جامعـــة خليفـــة
Khalifa University

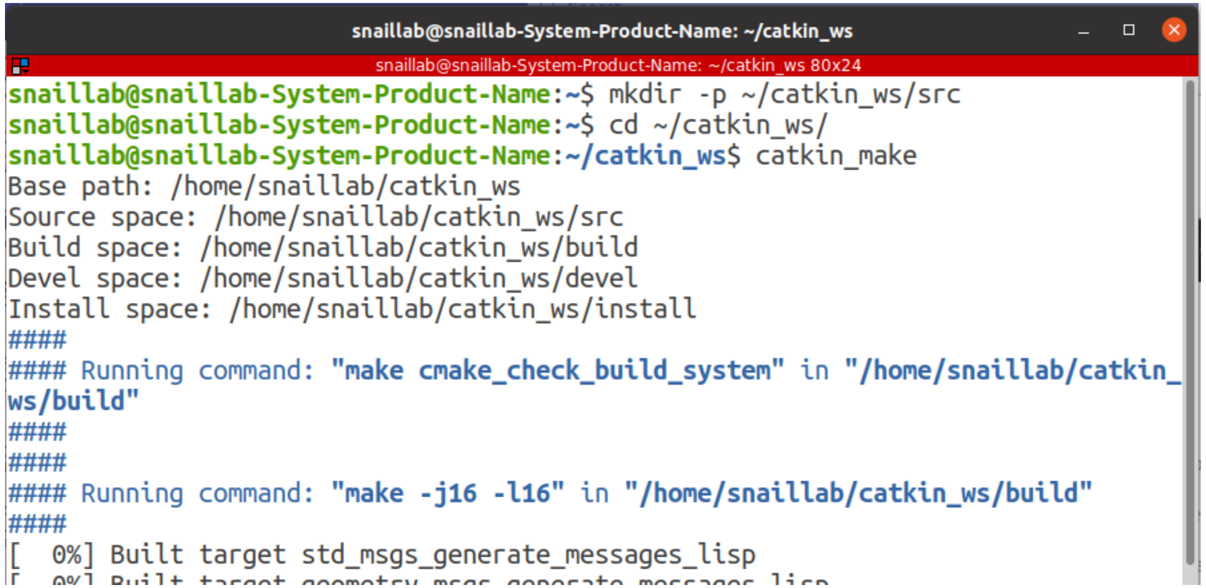
Day 3 & 4

THURSDAY, JULY 21, 2022

DIRECTIONS-How to Create a ROS Workspace

Open up a new terminal window

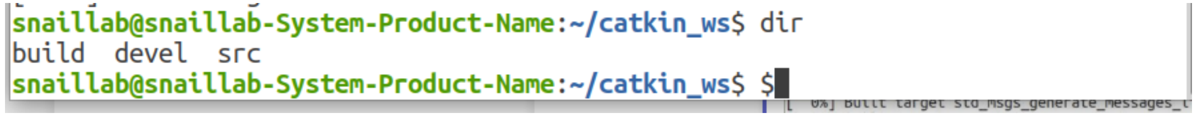
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make



snaillab@snaillab-System-Product-Name: ~/catkin_ws
snaillab@snaillab-System-Product-Name: ~/catkin_ws 80x24

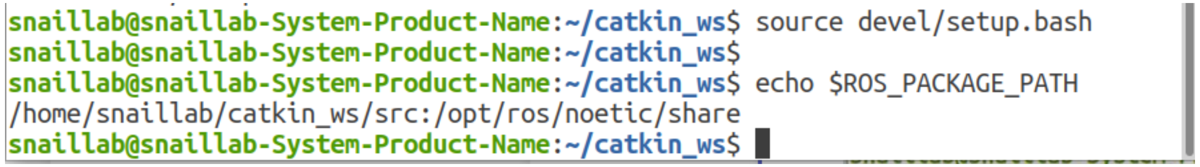
```
snaillab@snaillab-System-Product-Name:~$ mkdir -p ~/catkin_ws/src
snaillab@snaillab-System-Product-Name:~$ cd ~/catkin_ws/
snaillab@snaillab-System-Product-Name:~/catkin_ws$ catkin_make
Base path: /home/snaillab/catkin_ws
Source space: /home/snaillab/catkin_ws/src
Build space: /home/snaillab/catkin_ws/build
Devel space: /home/snaillab/catkin_ws/devel
Install space: /home/snaillab/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/snaillab/catkin_
ws/build"
####
####
#### Running command: "make -j16 -l16" in "/home/snaillab/catkin_ws/build"
####
[  0%] Built target std_msgs_generate_messages_lisp
[  0%] Built target geometry_msgs_generate_messages_lisp
```

Type the **dir** command, and you will see three folders inside of this directory: build, devel, and src.



```
snaillab@snaillab-System-Product-Name:~/catkin_ws$ dir
build  devel  src
snaillab@snaillab-System-Product-Name:~/catkin_ws$ $
```

Now we need to source the setup.bash file. This file sets the path of the workspace so that packages and code inside the workspace can be found.



```
snaillab@snaillab-System-Product-Name:~/catkin_ws$ source devel/setup.bash
snaillab@snaillab-System-Product-Name:~/catkin_ws$
snaillab@snaillab-System-Product-Name:~/catkin_ws$ echo $ROS_PACKAGE_PATH
/home/snaillab/catkin_ws/src:/opt/ros/noetic/share
snaillab@snaillab-System-Product-Name:~/catkin_ws$
```

DIRECTIONS-How to Create a ROS Workspace

Open up a new terminal window

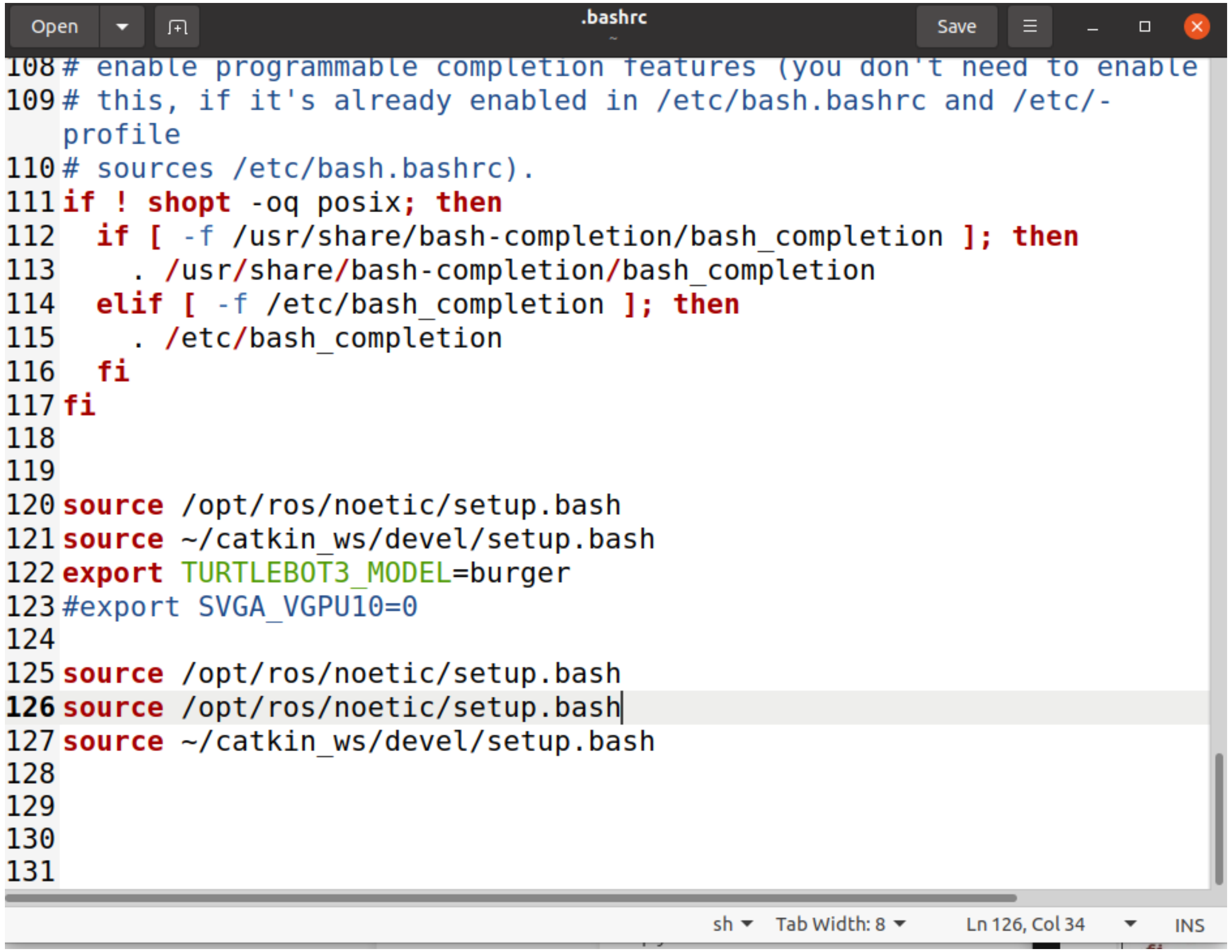
So we don’t have to source the setup.bash file every time we open a new Linux terminal, let’s add the ~/catkin_ws/devel/setup.bash command to the .bashrc file. Open a new Linux terminal window.

Type the following command to edit the .bashrc text file:

```
gedit ~/.bashrc
```

Add this line to the end of the .bashrc file:

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

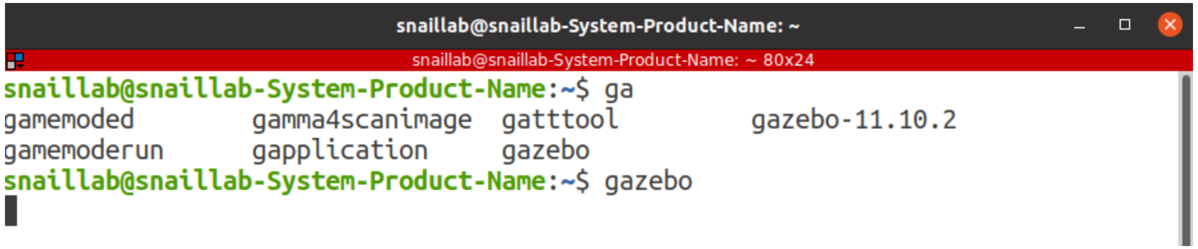


```
# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/-
profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi


source /opt/ros/noetic/setup.bash
source ~/catkin_ws/devel/setup.bash
export TURTLEBOT3_MODEL=burger
#export SVGA_VGPU10=0

source /opt/ros/noetic/setup.bash
source /opt/ros/noetic/setup.bash
source ~/catkin_ws/devel/setup.bash
```

DIRECTIONS-How to Launch Gazebo in Ubuntu

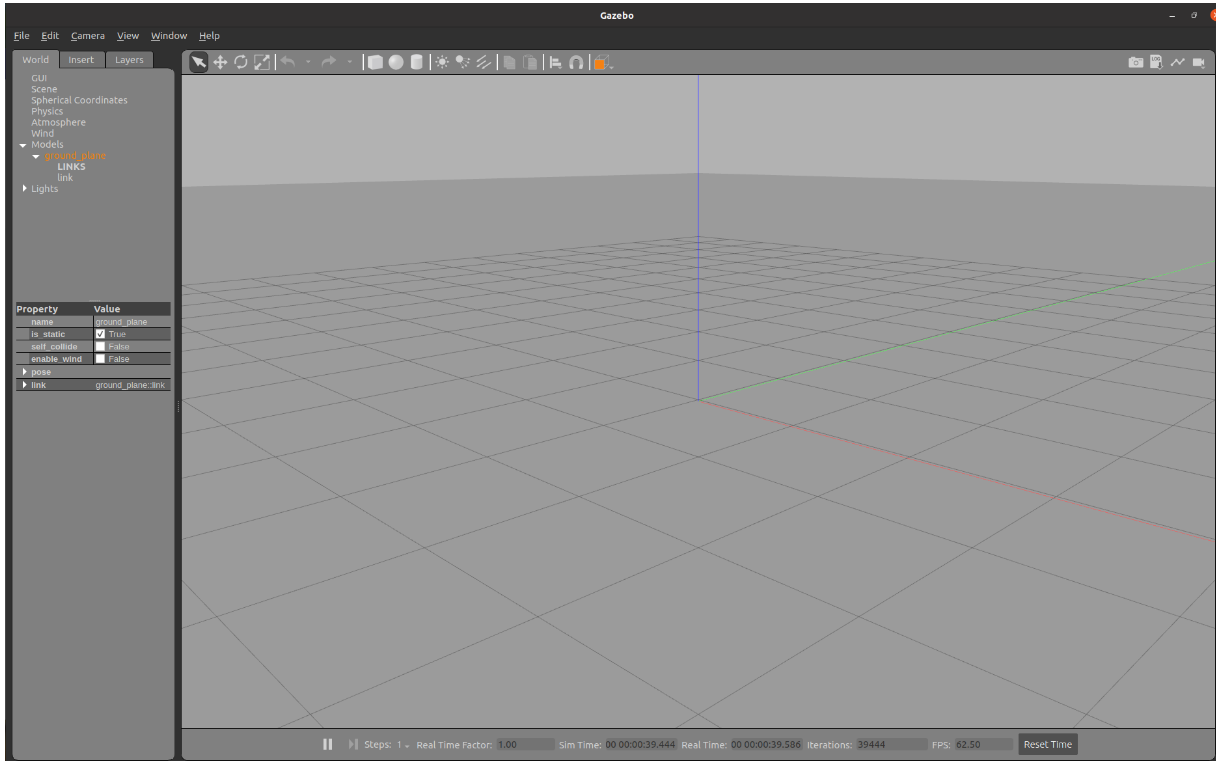
Gazebo is a 3D simulator that is a really good tool if you want to simulate your robot in a complex outdoor or indoor environment.

To launch Gazebo for the first time, open up a new terminal window, and type the following command.



snaillab@snaillab-System-Product-Name: ~
snaillab@snaillab-System-Product-Name: ~ 80x24

```
snaillab@snaillab-System-Product-Name:~$ ga
gamemoded          gamma4scanimage  gatttool          gazebo-11.10.2
gamemoderun        gapplication     gazebo
snaillab@snaillab-System-Product-Name:~$ gazebo
```



Property	Value
name	ground_plane
is_static	☑ True
self_collide	□ False
enable_wind	□ False
pose	
link	ground_plane::link

Steps: 1 Real Time Factor: 1.00 Sim Time: 00 00:00:39.444 Real Time: 00 00:00:39.586 Iterations: 39444 FPS: 62.50

DIRECTIONS-How to Launch Rviz and RQT in ROS

ROS also has some really cool graphical user interface (GUI) tools that enable you to interact with ROS in a more visual way than we have done so far.

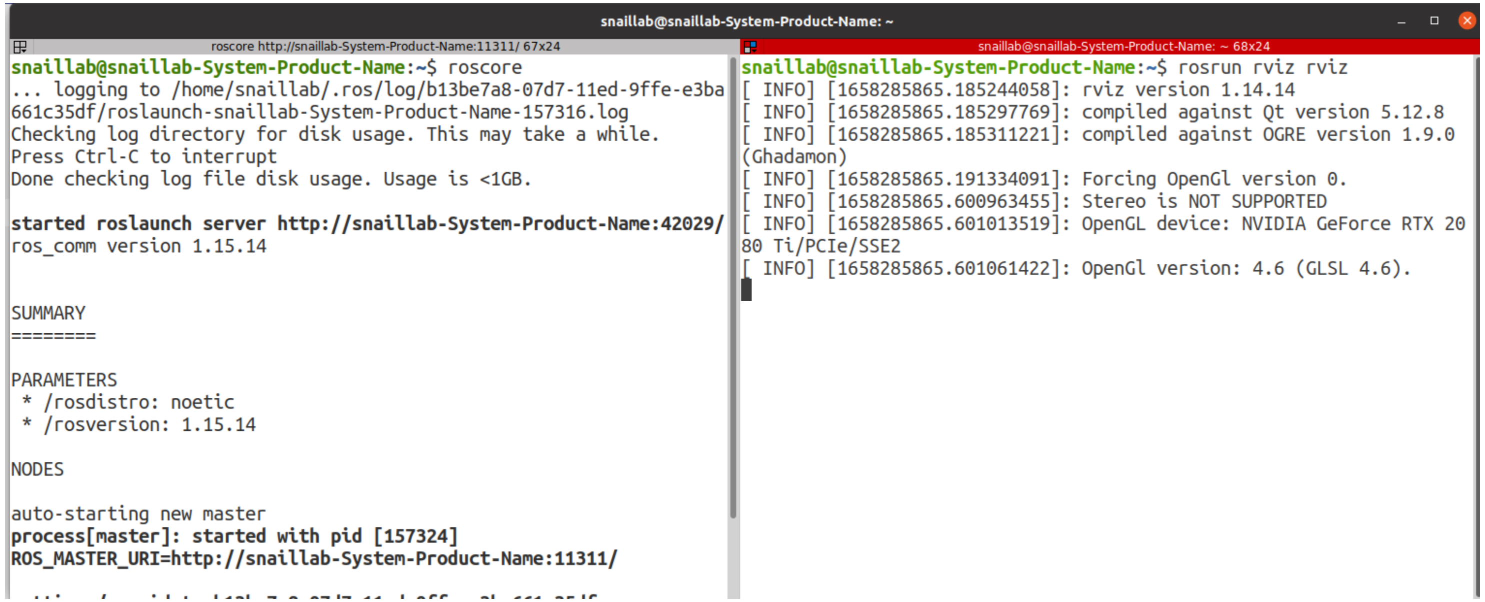
Two of these tools are rviz and rqt.

- rviz is a 3D visualizer for ROS
- rqt is a ROS visualization tool based on Qt, a free and open-source widget toolkit for creating GUIs.

DIRECTIONS-How to Launch Rviz

To launch rviz, open a new terminal window and type:

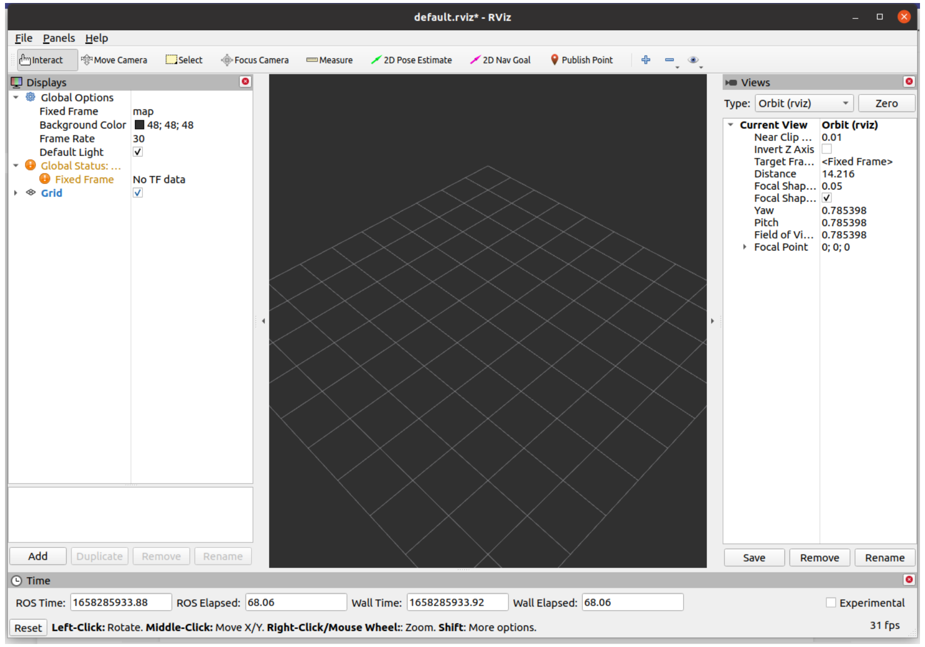
Open up a new terminal tab and type:



snaillab@snaillab-System-Product-Name: ~
roscore http://snaillab-System-Product-Name:11311/ 67x24
snaillab@snaillab-System-Product-Name: ~ 68x24

```
snaillab@snaillab-System-Product-Name:~$ roscore
... logging to /home/snaillab/.ros/log/b13be7a8-07d7-11ed-9ffe-e3ba
661c35df/roslaunch-snaillab-System-Product-Name-157316.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://snaillab-System-Product-Name:42029/
ros_comm version 1.15.14


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.14

NODES

auto-starting new master
process[master]: started with pid [157324]
ROS_MASTER_URI=http://snaillab-System-Product-Name:11311/
```

```
snaillab@snaillab-System-Product-Name:~$ rosrun rviz rviz
[ INFO] [1658285865.185244058]: rviz version 1.14.14
[ INFO] [1658285865.185297769]: compiled against Qt version 5.12.8
[ INFO] [1658285865.185311221]: compiled against OGRE version 1.9.0
(Ghadamon)
[ INFO] [1658285865.191334091]: Forcing OpenGl version 0.
[ INFO] [1658285865.600963455]: Stereo is NOT SUPPORTED
[ INFO] [1658285865.601013519]: OpenGL device: NVIDIA GeForce RTX 20
80 Ti/PCIe/SSE2
[ INFO] [1658285865.601061422]: OpenGl version: 4.6 (GLSL 4.6).
```



Global Options
Fixed Frame map
Background Color 48; 48; 48
Frame Rate 30
Default Light ☑
Global Status: ...
Fixed Frame No TF data
Grid ☑

Type: Orbit (rviz)

Current View Orbit (rviz)
Near Clip ... 0.01
Invert Z Axis ☐
Target Fra... <Fixed Frame>
Distance 14.216
Focal Shap... 0.05
Focal Shap... ☑
Yaw 0.785398
Pitch 0.785398
Field of Vi... 0.785398
Focal Point 0; 0; 0

ROS Time: 1658285933.88 ROS Elapsed: 68.06 Wall Time: 1658285933.92 Wall Elapsed: 68.06 ☐ Experimental

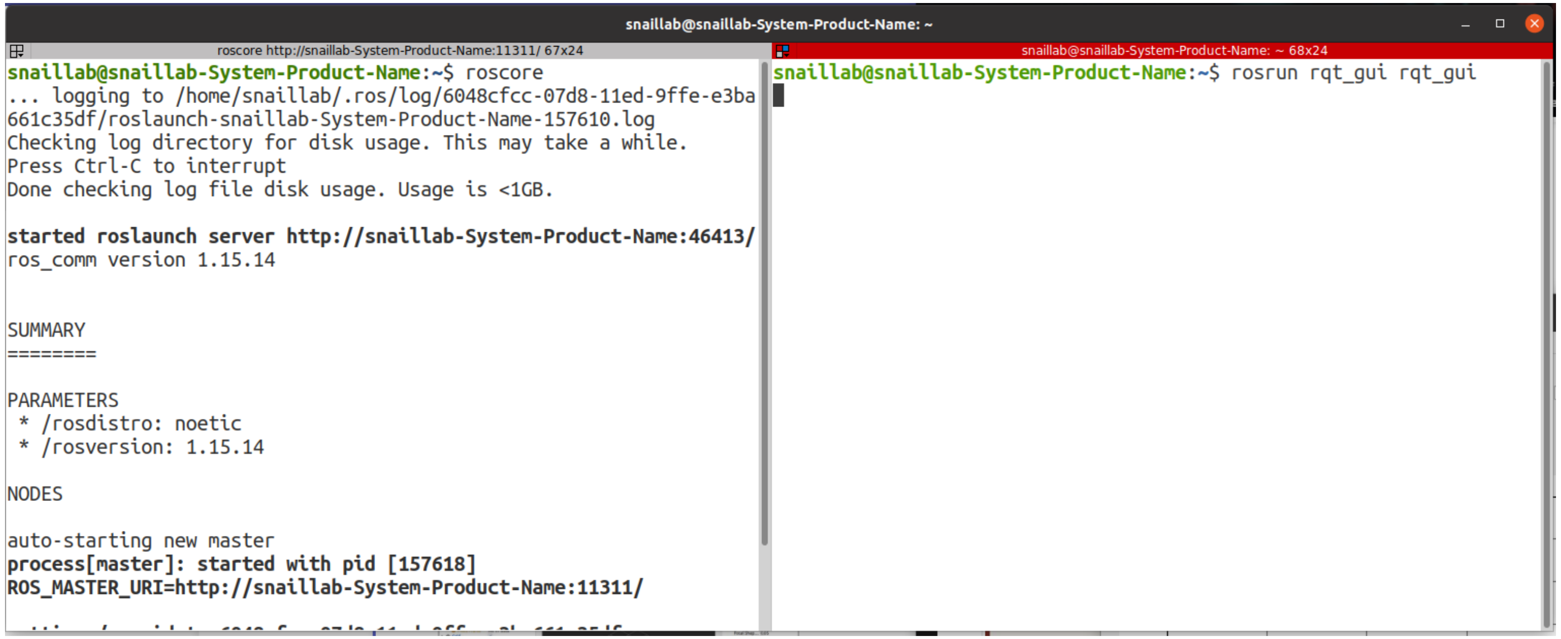
Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:**: Zoom. **Shift:** More options.

31 fps

DIRECTIONS-How to Launch RQT

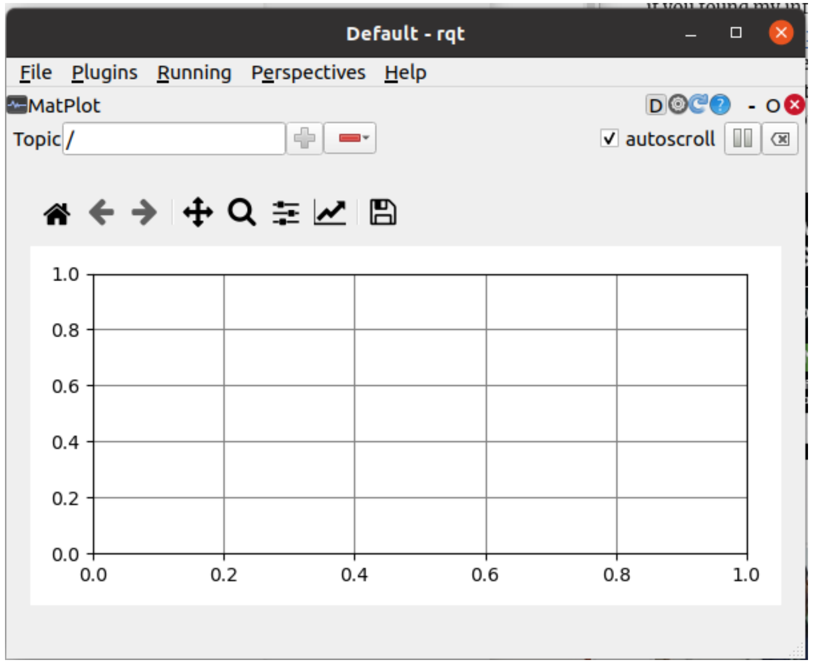
To launch rqt, open a new terminal window and type:

Open up a new terminal tab and type:


snaillab@snaillab-System-Product-Name: ~
roscore http://snaillab-System-Product-Name:11311/ 67x24
snaillab@snaillab-System-Product-Name: ~ 68x24

```
snaillab@snaillab-System-Product-Name:~$ roscore
... logging to /home/snaillab/.ros/log/6048cfcc-07d8-11ed-9ffe-e3ba
661c35df/roslaunch-snaillab-System-Product-Name-157610.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://snaillab-System-Product-Name:46413/
ros_comm version 1.15.14


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.14

NODES

auto-starting new master
process[master]: started with pid [157618]
ROS_MASTER_URI=http://snaillab-System-Product-Name:11311/
```

```
snaillab@snaillab-System-Product-Name:~$ rosrun rqt_gui rqt_gui
```



list of available Plugins by going to the Plugins option. Let’s go to Plugins -> Visualization -> Plot to get a blank plot.

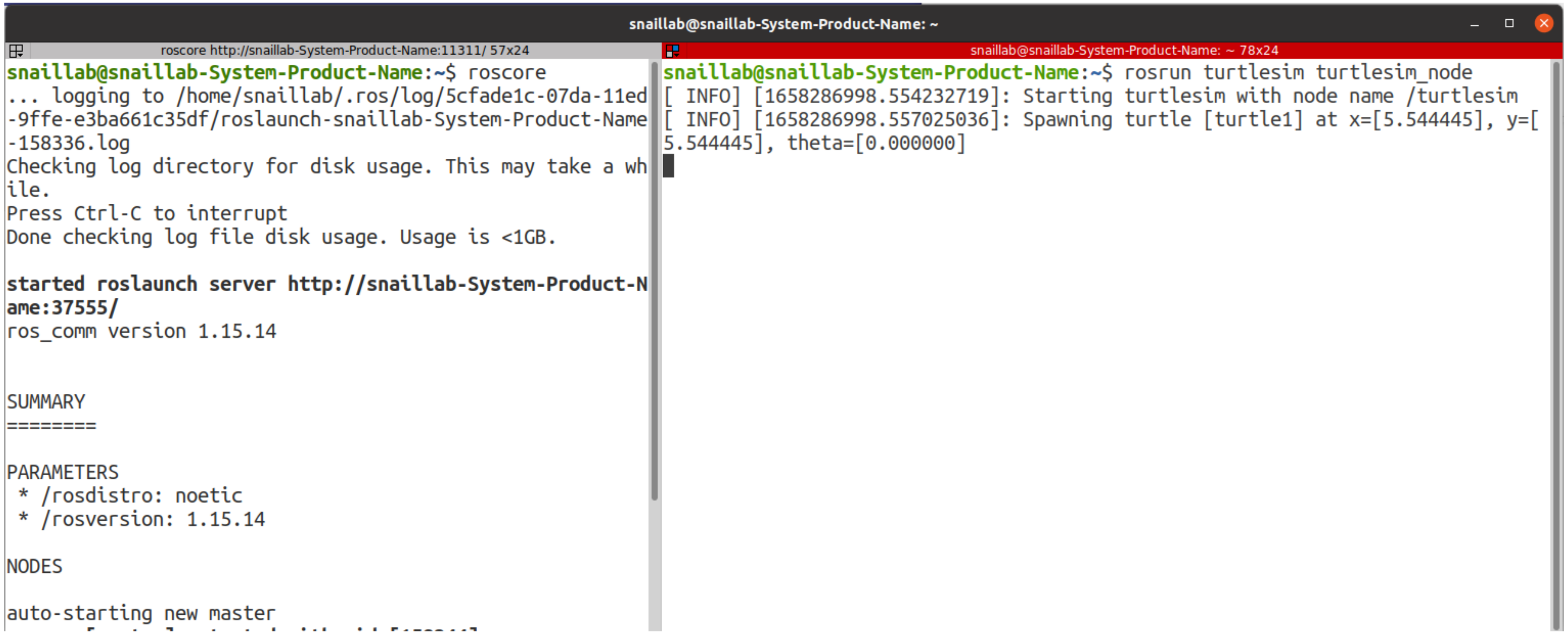
DIRECTIONS- Launch the Turtlesim Robot Simulation in ROS

In this section, we will work with the **turtlesim** application. This application comes pre-installed with ROS and consists of a 2D simulation of a turtle. You can move the turtle around and do a lot of other cool stuff as described [here at the turtlesim ROS Wiki page](#).

Let’s run this program now with rospy, the Python library for ROS.

Let’s launch turtlesim now. Open up a new terminal window, and type:

Open a new terminal tab, and launch the **turtlesim** application.


snaillab@snaillab-System-Product-Name: ~
roscore http://snaillab-System-Product-Name:11311/ 57x24
snaillab@snaillab-System-Product-Name: ~ 78x24

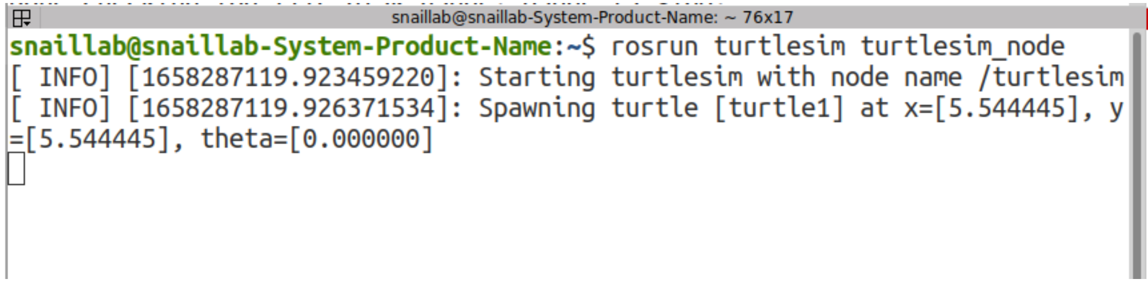
```
snaillab@snaillab-System-Product-Name:~$ roscore
... logging to /home/snaillab/.ros/log/5cfade1c-07da-11ed
-9ffe-e3ba661c35df/roslaunch-snaillab-System-Product-Name
-158336.log
Checking log directory for disk usage. This may take a wh
ile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://snaillab-System-Product-N
ame:37555/
ros_comm version 1.15.14


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.14

NODES

auto-starting new master
```

```
snaillab@snaillab-System-Product-Name:~$ rosrun turtlesim turtlesim_node
[ INFO] [1658286998.554232719]: Starting turtlesim with node name /turtlesim
[ INFO] [1658286998.557025036]: Spawning turtle [turtle1] at x=[5.544445], y=[
5.544445], theta=[0.000000]
```

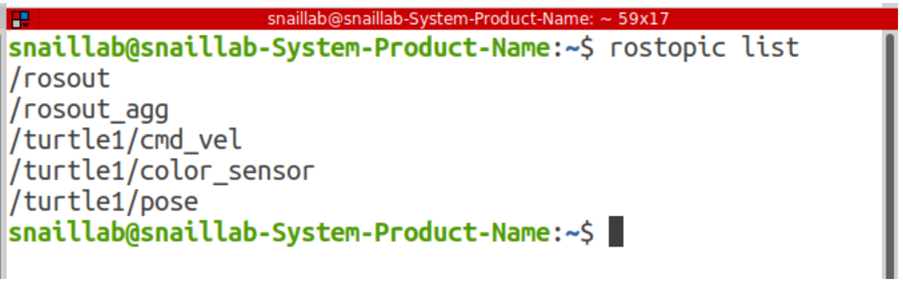
DIRECTIONS- Launch the Turtlesim Robot Simulation in ROS

Let’s see the list of topics. Remember that a topic in ROS is a named bus (or channel) over which a node publishes messages for other nodes to receive.

Open up a new terminal window, and type:



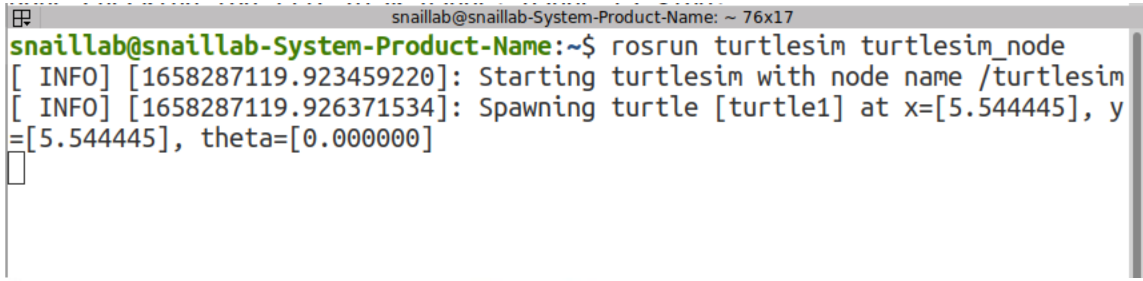
```
snaillab@snaillab-System-Product-Name:~$ rosrun turtlesim turtlesim_node
[ INFO] [1658287119.923459220]: Starting turtlesim with node name /turtlesim
[ INFO] [1658287119.926371534]: Spawning turtle [turtle1] at x=[5.544445], y
=[5.544445], theta=[0.000000]
```



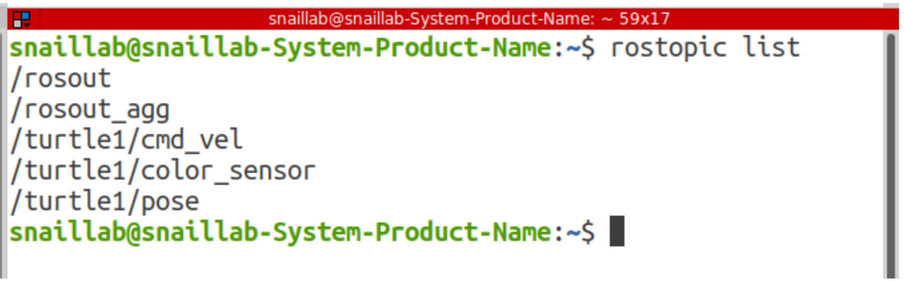
```
snaillab@snaillab-System-Product-Name:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
snaillab@snaillab-System-Product-Name:~$
```

ROS nodes communicate with each other is the ROS Topics model, in which a Publisher Node sends messages via a Topic to one or more registered Subscriber nodes.

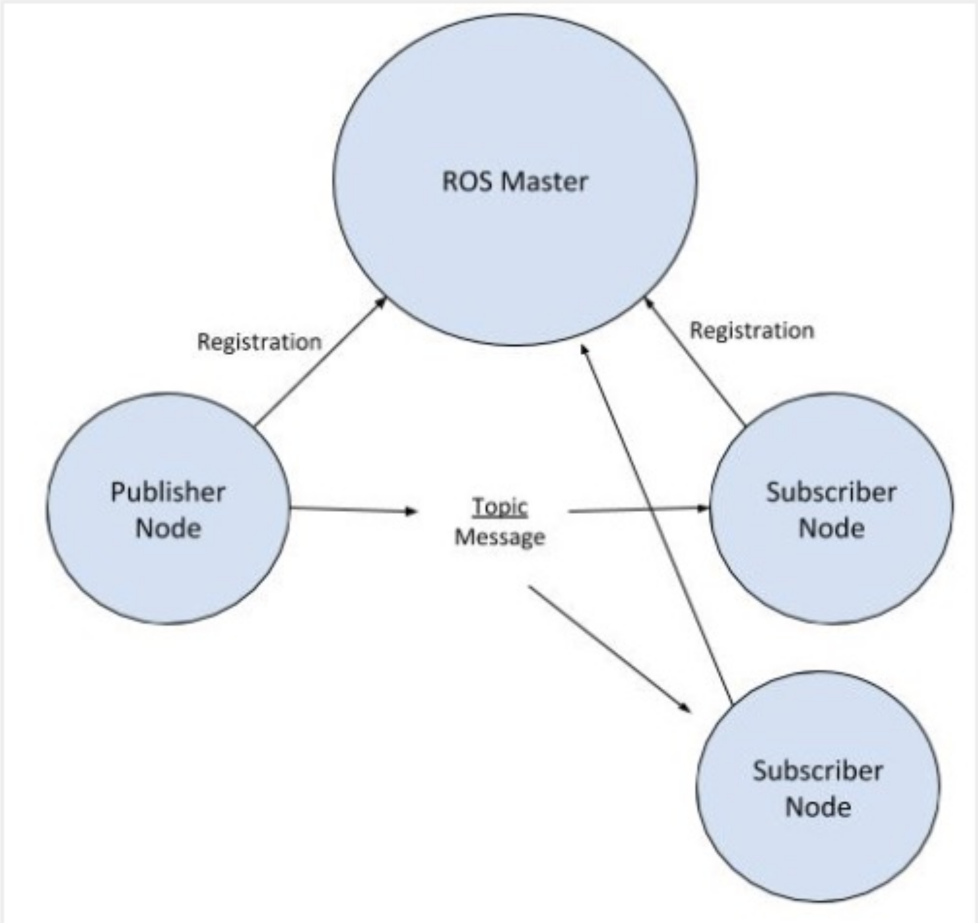
DIRECTIONS- Launch the Turtlesim Robot Simulation in ROS



```
snaillab@snaillab-System-Product-Name: ~ 76x17
snaillab@snaillab-System-Product-Name:~$ rosrun turtlesim turtlesim_node
[ INFO] [1658287119.923459220]: Starting turtlesim with node name /turtlesim
[ INFO] [1658287119.926371534]: Spawning turtle [turtle1] at x=[5.544445], y
=[5.544445], theta=[0.000000]
```



```
snaillab@snaillab-System-Product-Name: ~ 59x17
snaillab@snaillab-System-Product-Name:~$ rostopic list
/rosout
/rosout_agg
/turtle1/cmd_vel
/turtle1/color_sensor
/turtle1/pose
snaillab@snaillab-System-Product-Name:~$ 
```

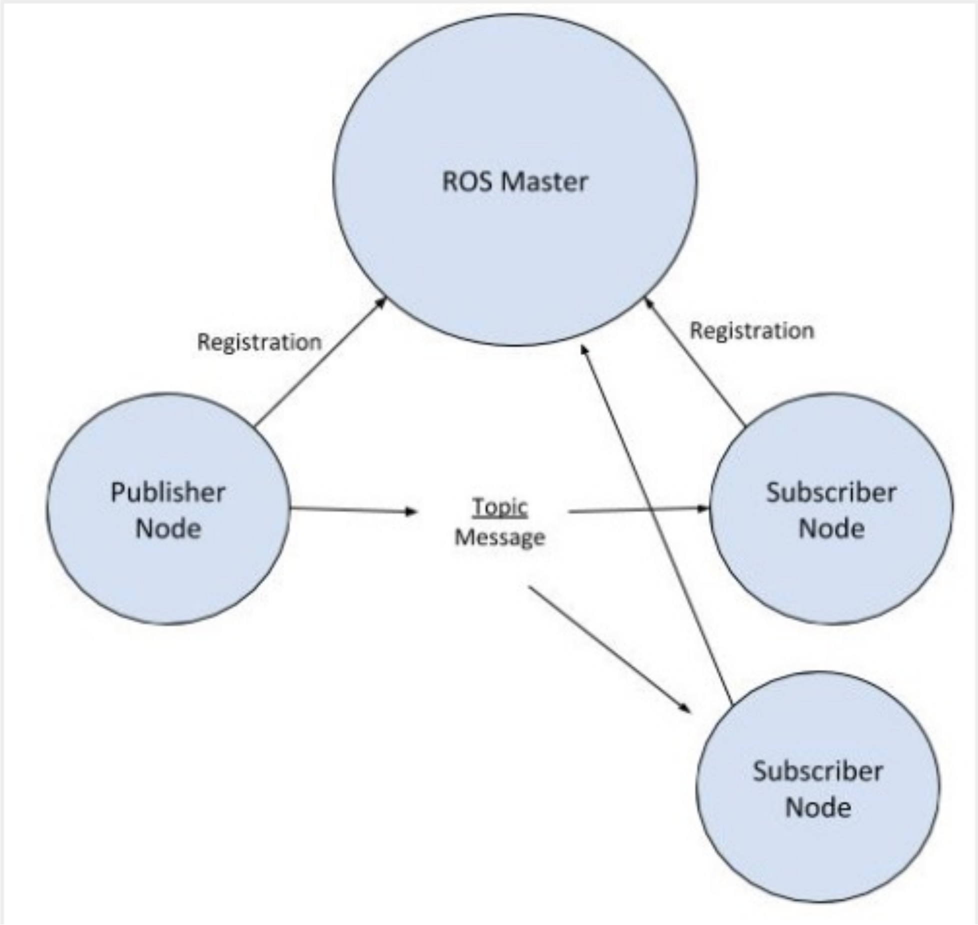


ROS nodes communicate with each other is the ROS Topics model, in which a Publisher Node sends messages via a Topic to one or more registered Subscriber nodes.

Notice that the flow of information between nodes is one-way, from Publisher to Subscriber.

DIRECTIONS- Launch the Turtlesim Robot Simulation in ROS

Notice that the flow of information between nodes is one-way, from Publisher to Subscriber.



What do we do in a situation where we have a node that wants to request information from another node and receive an immediate reply?

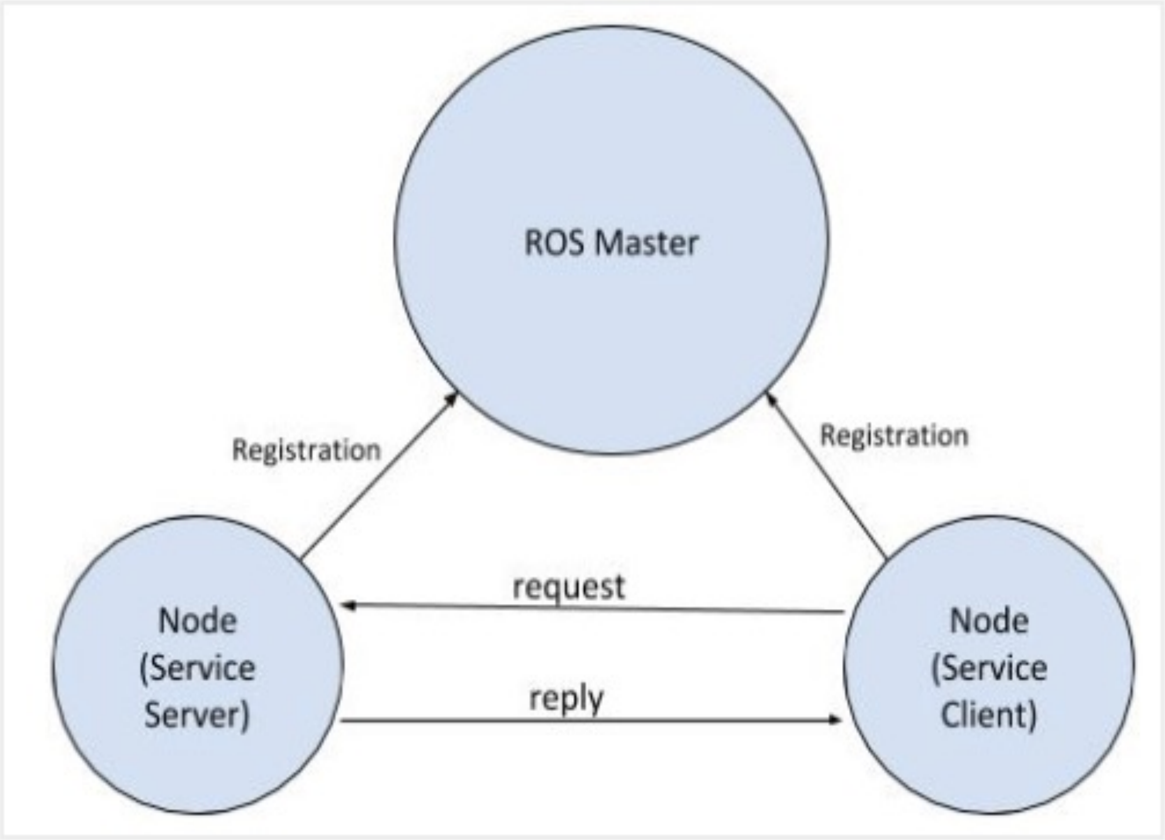
How is this two-way communication implemented in ROS?

Request/reply in ROS is performed via ROS Services.

DIRECTIONS- Launch the Turtlesim Robot Simulation in ROS

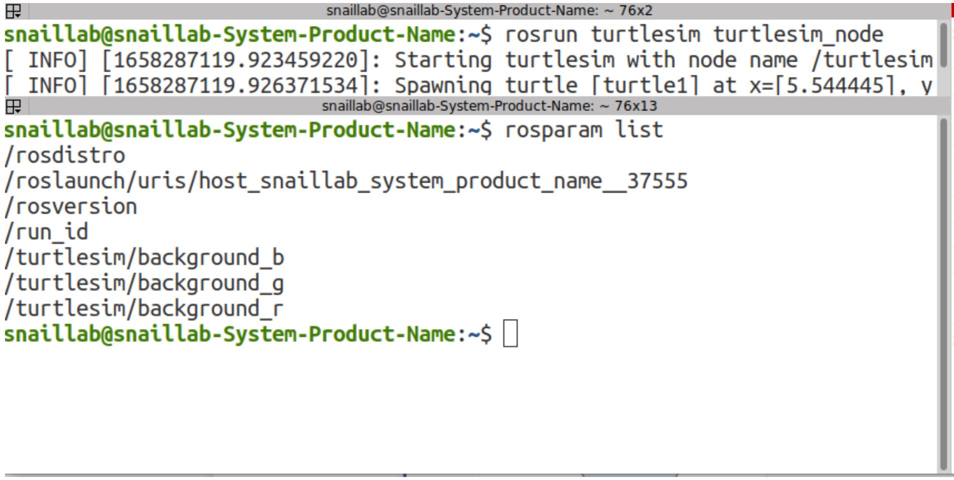
How is this two-way communication implemented in ROS?

- A ROS Service consists of a pair of messages: one for the request and one for the reply.
- A service-providing ROS node (i.e. Service Server) offers a service (e.g. read sensor data).
- A client node (i.e. Service Client) calls the service by sending a request message to the service provider.
- The client node then awaits the reply. Here is what the **ROS Service model**



DIRECTIONS- Launch the Turtlesim Robot Simulation in ROS

How is this two-way communication implemented in ROS?

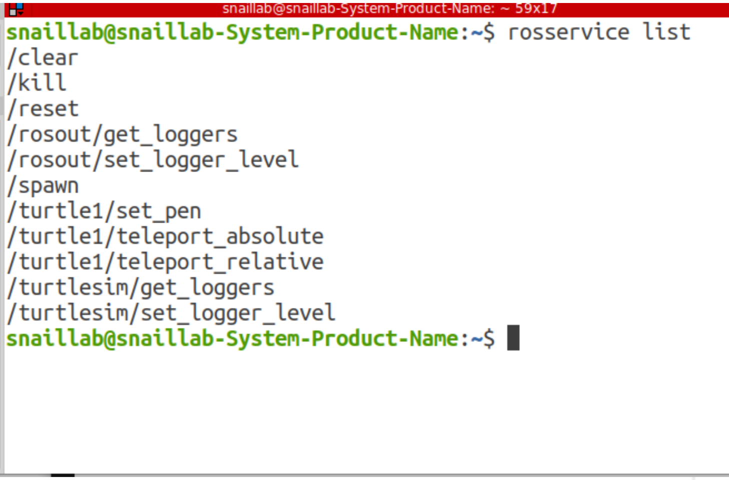


snaillab@snaillab-System-Product-Name: ~ 76x2

```
snaillab@snaillab-System-Product-Name:~$ rosrun turtlesim turtlesim_node
[ INFO] [1658287119.923459220]: Starting turtlesim with node name /turtlesim
[ INFO] [1658287119.926371534]: Spawning turtle [turtle1] at x=[5.544445], y
```

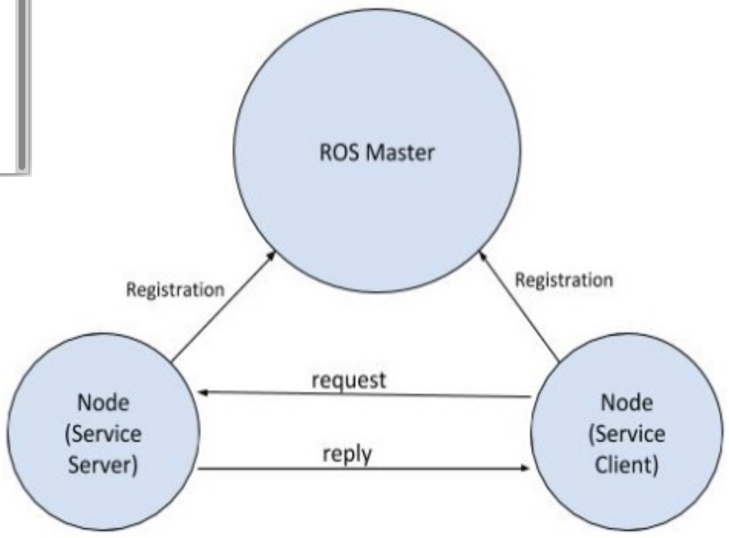
snaillab@snaillab-System-Product-Name: ~ 76x13

```
snaillab@snaillab-System-Product-Name:~$ rosparam list
/rosdistro
/roslaunch/uris/host_snaillab_system_product_name__37555
/rosversion
/run_id
/turtlesim/background_b
/turtlesim/background_g
/turtlesim/background_r
snaillab@snaillab-System-Product-Name:~$
```



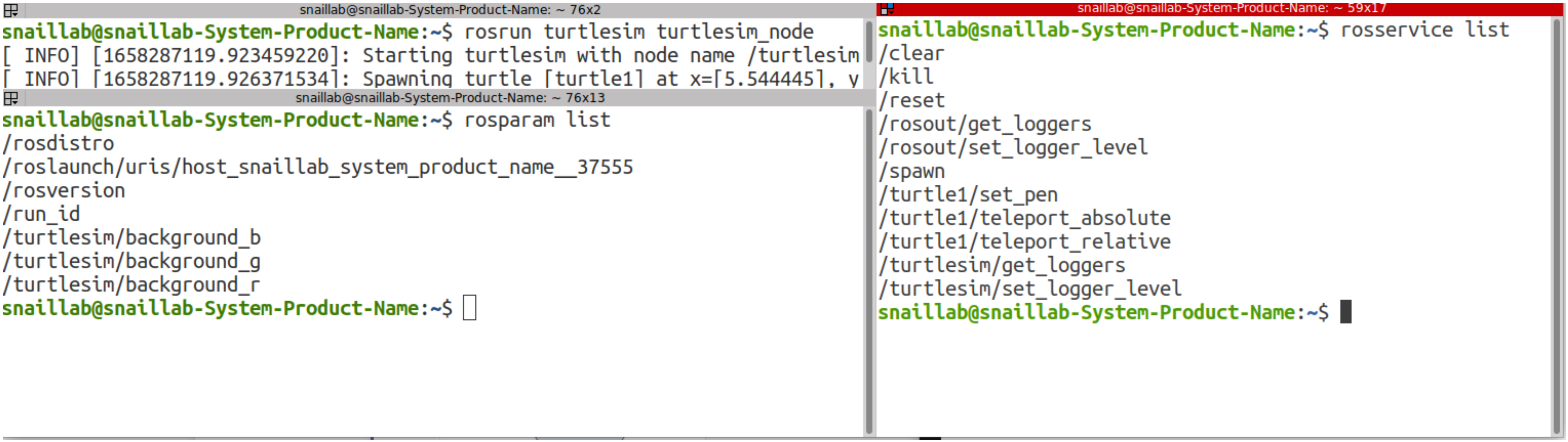
snaillab@snaillab-System-Product-Name: ~ 59x17

```
snaillab@snaillab-System-Product-Name:~$ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/spawn
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
snaillab@snaillab-System-Product-Name:~$
```



DIRECTIONS- Launch the Turtlesim Robot Simulation in ROS

Let’s see a list of **ROS parameters**. Think of parameters as the global settings for our current ROS environment (e.g. things like the background color of the turtlesim screen, version of ROS we are using, etc.).



snaillab@snaillab-System-Product-Name: ~ 76x2

```
snaillab@snaillab-System-Product-Name:~$ rosrun turtlesim turtlesim_node
[ INFO] [1658287119.923459220]: Starting turtlesim with node name /turtlesim
[ INFO] [1658287119.926371534]: Spawning turtle [turtle1] at x=[5.544445], y
```

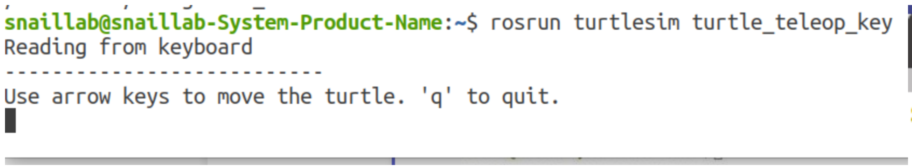
snaillab@snaillab-System-Product-Name: ~ 76x13

```
snaillab@snaillab-System-Product-Name:~$ rosparam list
/rosdistro
/roslaunch/uris/host_snaillab_system_product_name__37555
/rosversion
/run_id
/turtlesim/background_b
/turtlesim/background_g
/turtlesim/background_r
snaillab@snaillab-System-Product-Name:~$ 
```

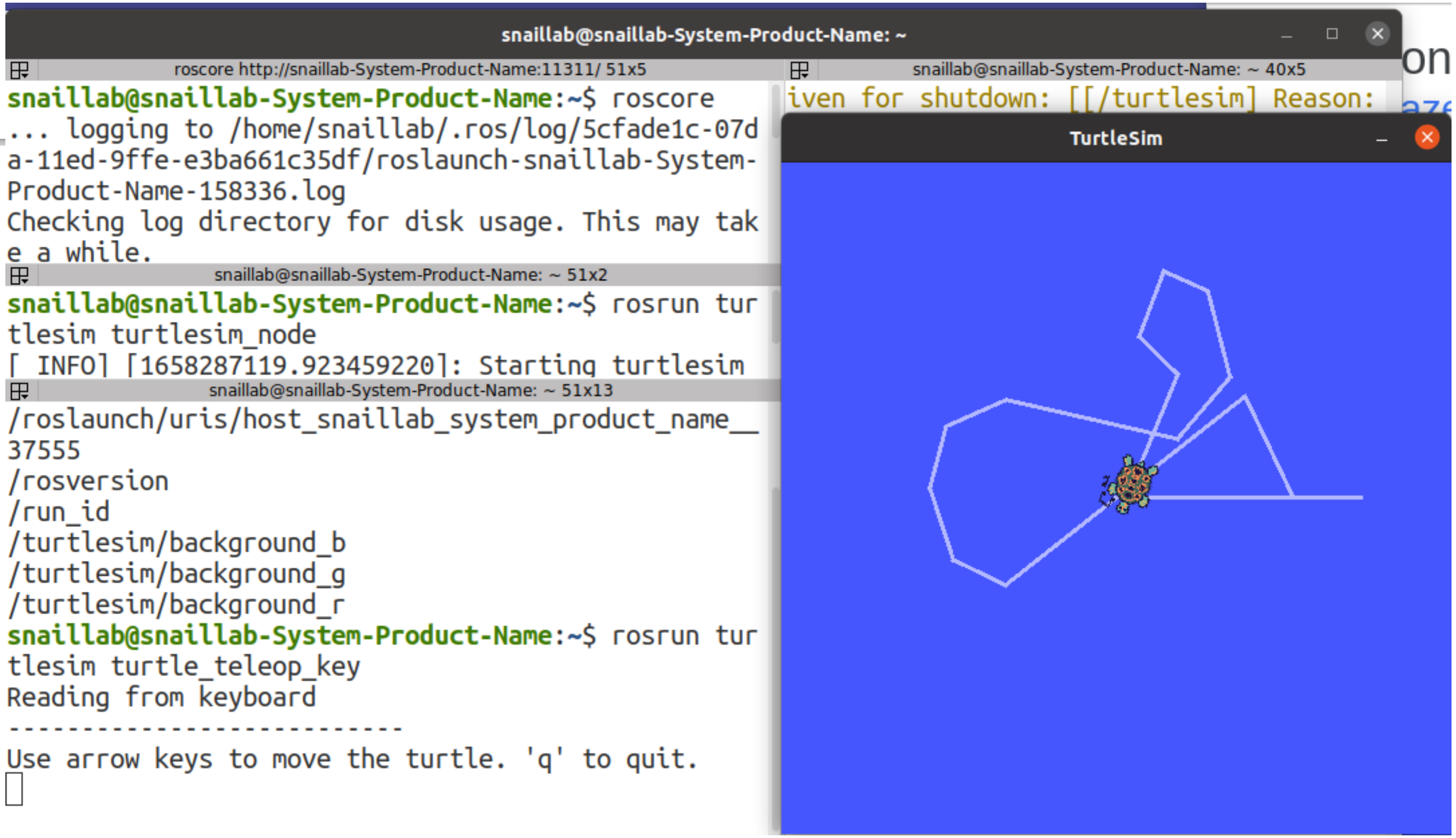
```
snaillab@snaillab-System-Product-Name:~$ rosservice list
/clear
/kill
/reset
/rosout/get_loggers
/rosout/set_logger_level
/spawn
/turtle1/set_pen
/turtle1/teleport_absolute
/turtle1/teleport_relative
/turtlesim/get_loggers
/turtlesim/set_logger_level
snaillab@snaillab-System-Product-Name:~$ 
```

DIRECTIONS- Launch the Turtlesim Robot Simulation in ROS

open up a new terminal tab and type this command



```
snaillab@snaillab-System-Product-Name:~$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle. 'q' to quit.
```



snaillab@snaillab-System-Product-Name: ~
roscore http://snaillab-System-Product-Name:11311/ 51x5
snaillab@snaillab-System-Product-Name: ~ 40x5
snaillab@snaillab-System-Product-Name: ~ 51x2
snaillab@snaillab-System-Product-Name: ~ 51x13
TurtleSim
on
aze

```
snaillab@snaillab-System-Product-Name:~$ roscore
... logging to /home/snaillab/.ros/log/5cfade1c-07d
a-11ed-9ffe-e3ba661c35df/roslaunch-snaillab-System-
Product-Name-158336.log
Checking log directory for disk usage. This may tak
e a while.
```

```
iven for shutdown: [[/turtlesim] Reason:
```

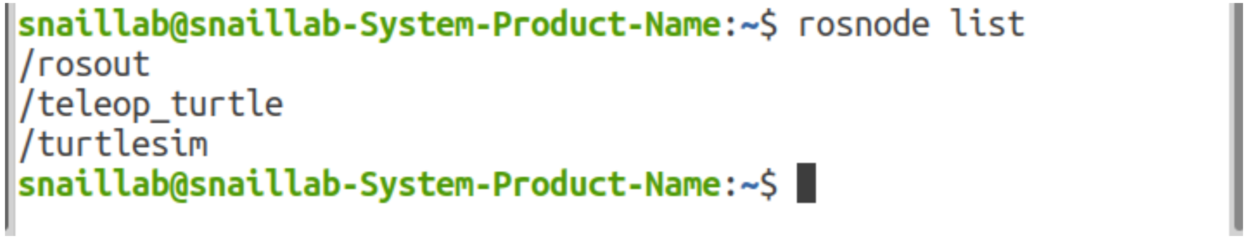
```
snaillab@snaillab-System-Product-Name:~$ rosrun tur
tlesim turtlesim_node
[ INFO] [1658287119.923459220]: Starting turtlesim
```

```
/roslaunch/uris/host_snaillab_system_product_name__
37555
/rosversion
/run_id
/turtlesim/background_b
/turtlesim/background_g
/turtlesim/background_r
snaillab@snaillab-System-Product-Name:~$ rosrun tur
tlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle. 'q' to quit.
```

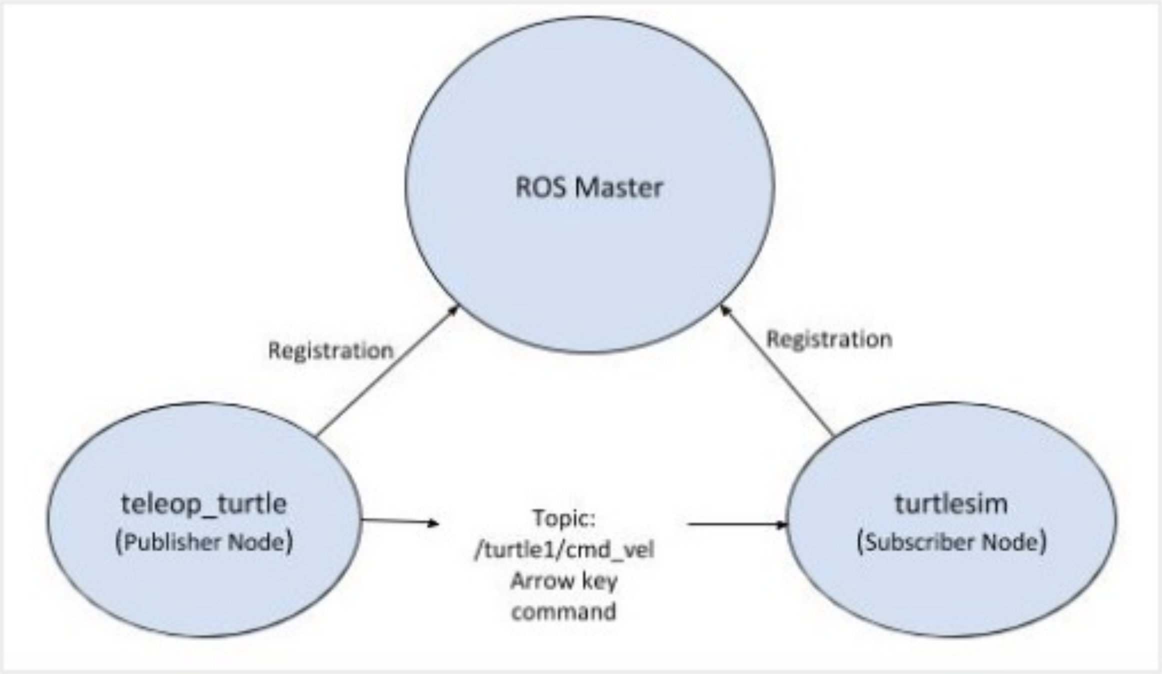
Each time you press an arrow key, the teleop_turtle node publishes a message to the /turtle1/cmd_vel topic. turtlesim node is subscribed to that topic. It receives the message and moves the turtle.

DIRECTIONS- Launch the Turtlesim Robot Simulation in ROS

open up a new terminal tab and type this command



```
snaillab@snaillab-System-Product-Name:~$ rosnode list
/rosout
/teleop_turtle
/turtlesim
snaillab@snaillab-System-Product-Name:~$ 
```



DIRECTIONS- Launch the Turtlesim Robot Simulation in ROS

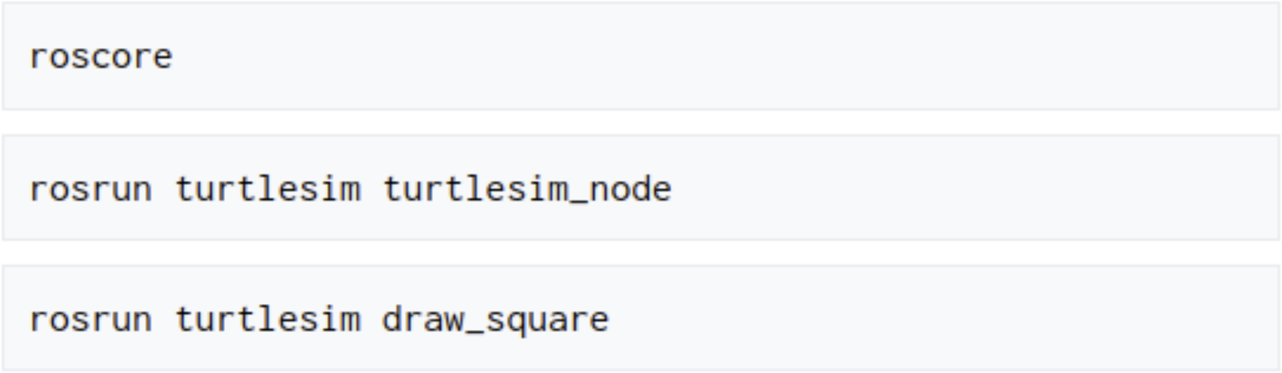
Exercise: Now draw a square with turtlesim.

Press Ctrl+C to stop the simulation. Close all terminal windows and start a new terminal window. Type:

DIRECTIONS- Launch the Turtlesim Robot Simulation in ROS

Exercise: Now draw a square with turtlesim.

Press Ctrl+C to stop the simulation. Close all terminal windows and start a new terminal window. Type:

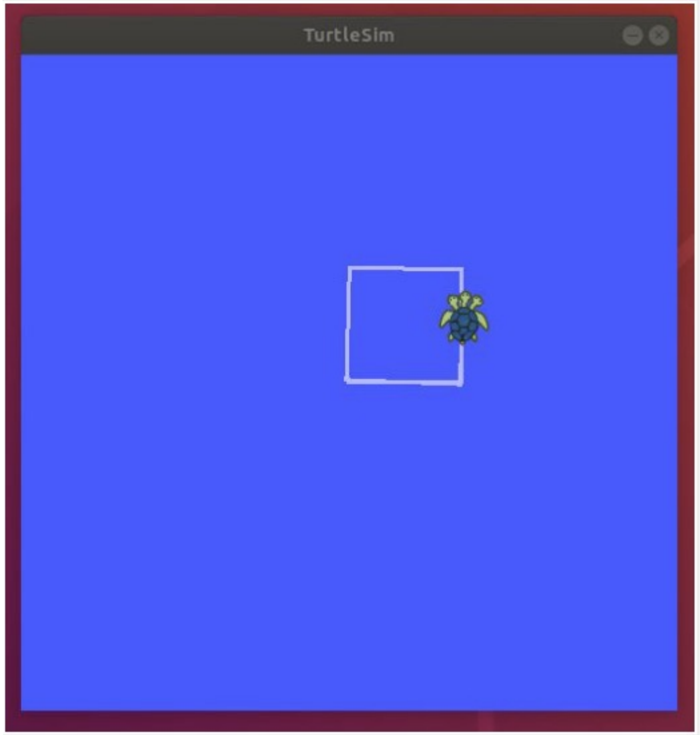


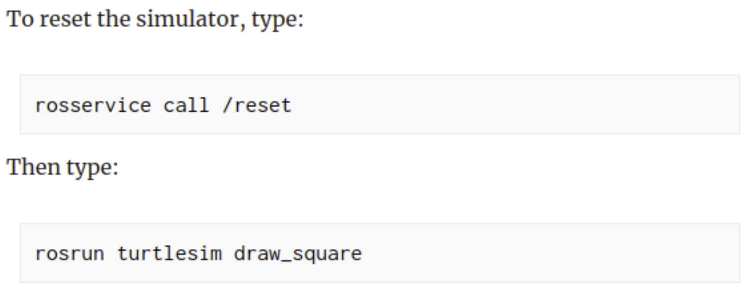
```
roscore
```

```
rosrun turtlesim turtlesim_node
```

```
rosrun turtlesim draw_square
```

The robot will go around and around along a square-shaped path.





To reset the simulator, type:

```
rosservice call /reset
```

Then type:

```
rosrun turtlesim draw_square
```

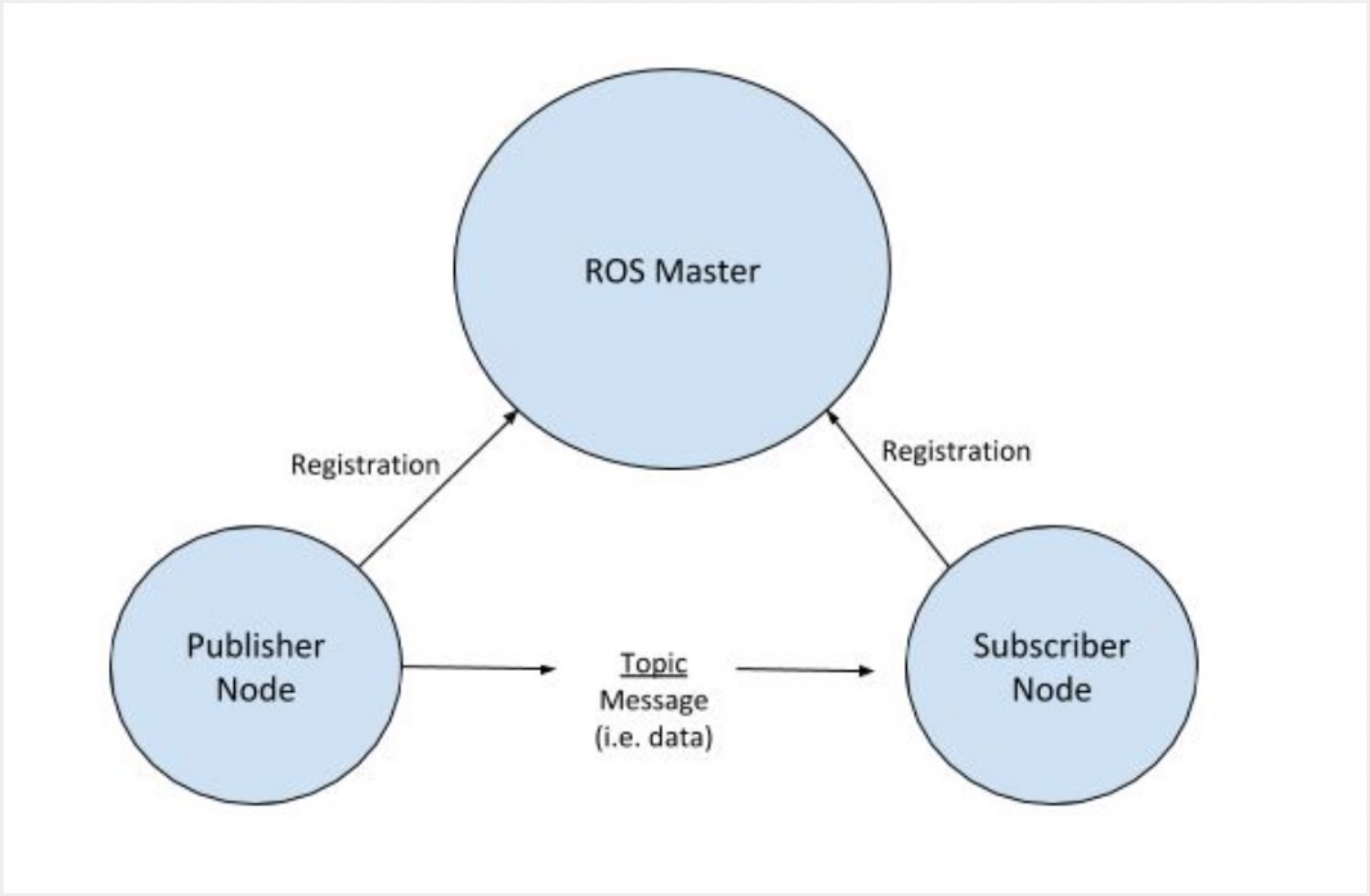
DIRECTIONS- Create a Hello World Project in ROS

- In this project is to get two pieces of ROS software (called **nodes**) to talk to each other. You can think of nodes as small single-purpose programs within a larger robotic system.
- One way nodes communicate with each other is by using **messages**. These messages are passed via channels called **topics**.
- Nodes that send data are known as **publisher nodes**, and nodes that receive data are known as **subscriber nodes.**
- The node that keeps track (i.e. a register) of which nodes are publisher nodes and which nodes are subscriber nodes is called the **ROS Master**.
- Without the ROS Master, nodes would not be able to communicate with each other.

DIRECTIONS- Create a Hello World Project in ROS

- Nodes that are interested in a particular piece of data subscribe to the relevant topic; nodes that generate data publish to the relevant topic.
- There can be multiple publishers and subscribers to a topic.
- You can think of topics like a middle man between publishers (nodes that generate data) and subscribers (nodes that receive data).
- The communication is anonymous, so nodes do not know what nodes they are sending data to/receiving data from.

DIRECTIONS- Create a Hello World Project in ROS



DIRECTIONS- Create a Hello World Project in ROS

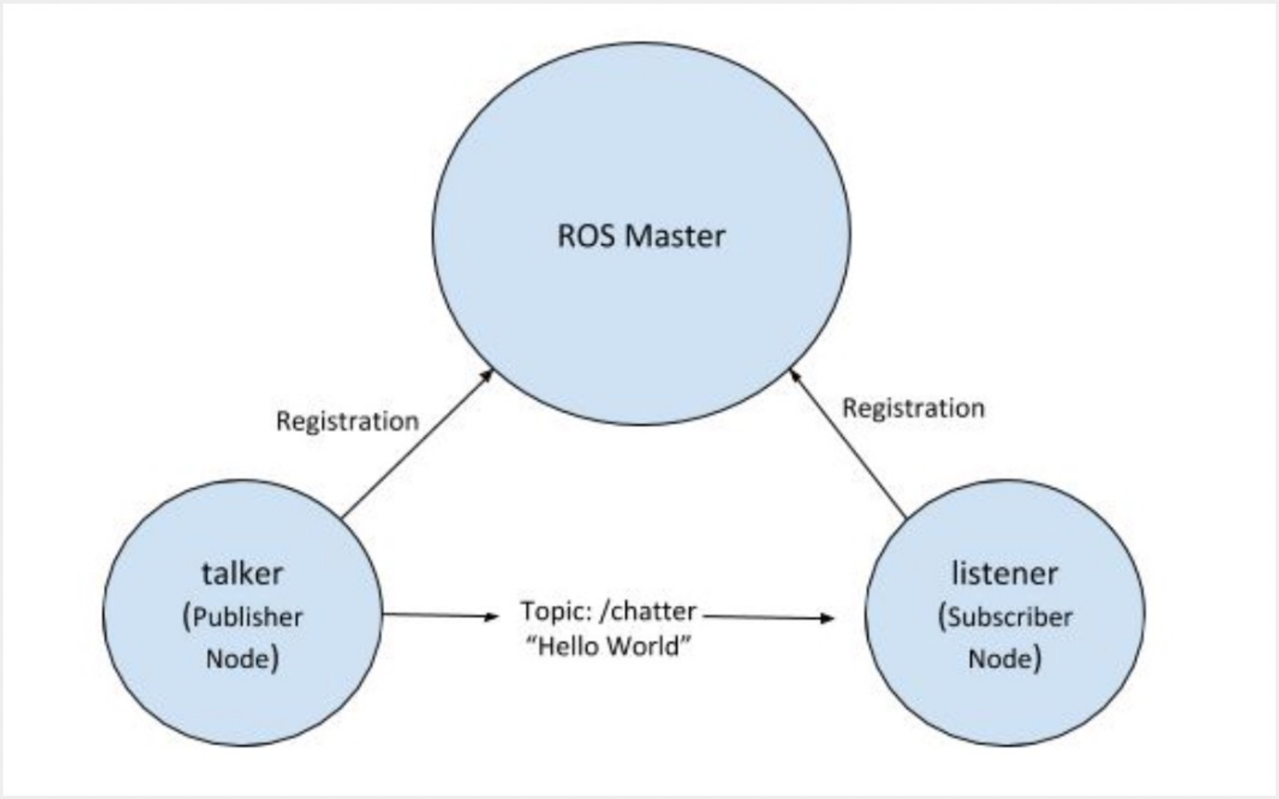
A good analogy

- Think of YouTube (or even other social media sites like Twitter or Instagram).
    - YouTubers (publisher nodes) publish videos (messages) to a channel (topic), and you (subscriber node) can subscribe to that channel (topic) so that you receive all the videos (messages) on that channel (topic).
    - YouTube (**ROS Master**) keeps track of who is a **publisher** and who is a **subscriber**.
    - One thing to keep in mind is that (in contrast to **YouTube**) **in ROS** there ***can be multiple publishers to the same topic, and publishers and subscribers don't know each other.***

DIRECTIONS- Create a Hello World Project in ROS

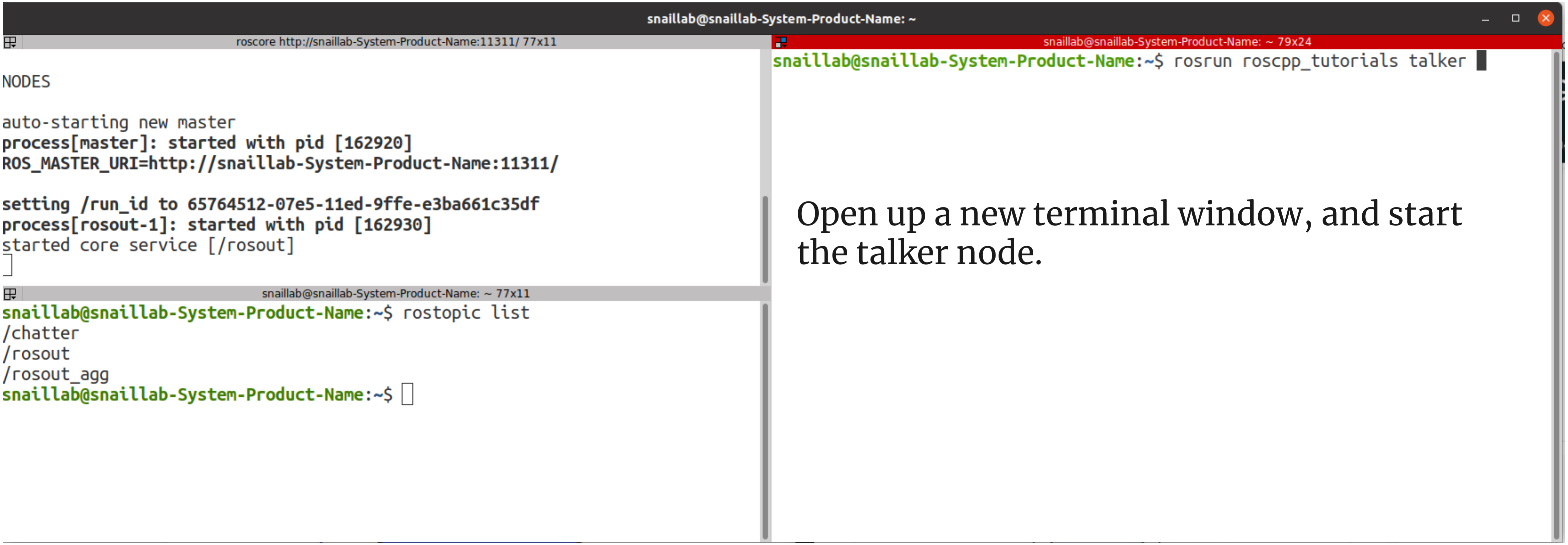
Develop an application that consists of **two nodes**: **talker** and **listener**.

- The talker node will publish a **“Hello World”** message to the **/chatter topic**.
- The listener will **subscribe** to the **/chatter topic** so that it can **receive** the **“Hello World”** message..



DIRECTIONS- Create a Hello World Project in ROS

To launch ROS, open a new Linux terminal window and type the following command:



snaillab@snaillab-System-Product-Name: ~
roscore http://snaillab-System-Product-Name:11311/ 77x11
snaillab@snaillab-System-Product-Name: ~ 79x24
snaillab@snaillab-System-Product-Name: ~ 77x11

```
NODES

auto-starting new master
process[master]: started with pid [162920]
ROS_MASTER_URI=http://snaillab-System-Product-Name:11311/

setting /run_id to 65764512-07e5-11ed-9ffe-e3ba661c35df
process[rosout-1]: started with pid [162930]
started core service [/rosout]
```

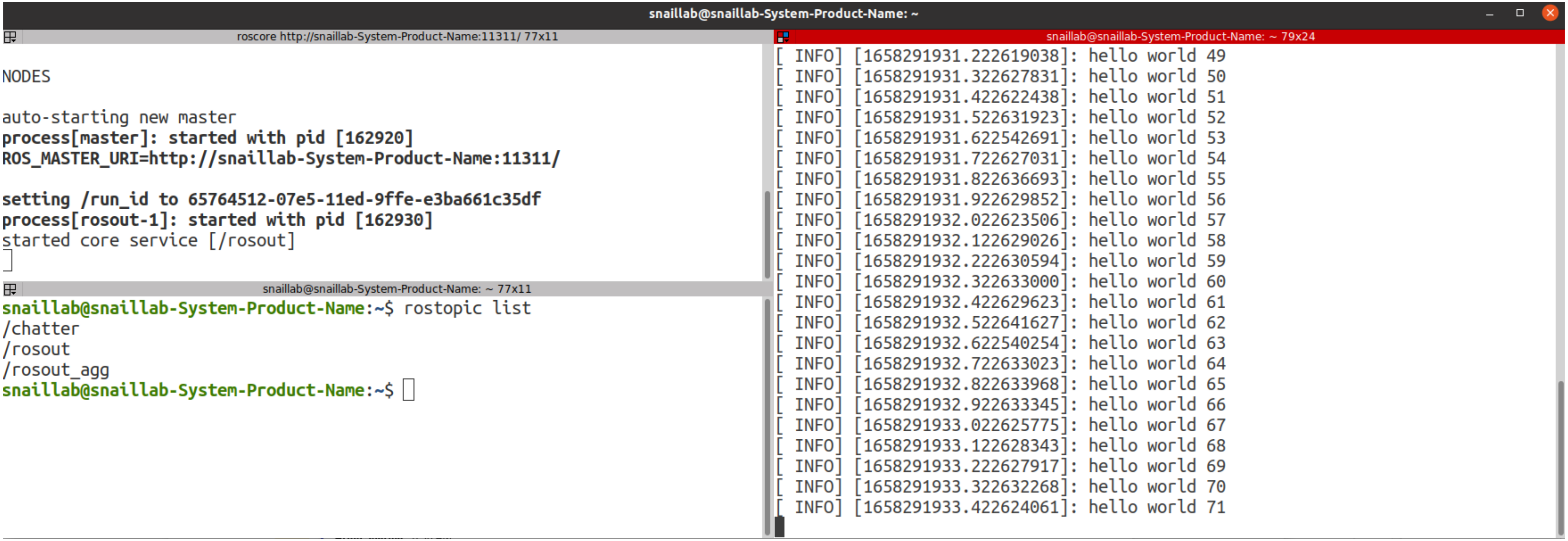
```
snaillab@snaillab-System-Product-Name:~$ rostopic list
/chatter
/rosout
/rosout_agg
snaillab@snaillab-System-Product-Name:~$
```

```
snaillab@snaillab-System-Product-Name:~$ rosrun roscpp_tutorials talker
```

Open up a new terminal window, and start the talker node.

DIRECTIONS- Create a Hello World Project in ROS

You should see hello world messages repeatedly printing to the screen.


snaillab@snaillab-System-Product-Name: ~
roscore http://snaillab-System-Product-Name:11311/ 77x11
snaillab@snaillab-System-Product-Name: ~ 77x11
snaillab@snaillab-System-Product-Name: ~ 79x24

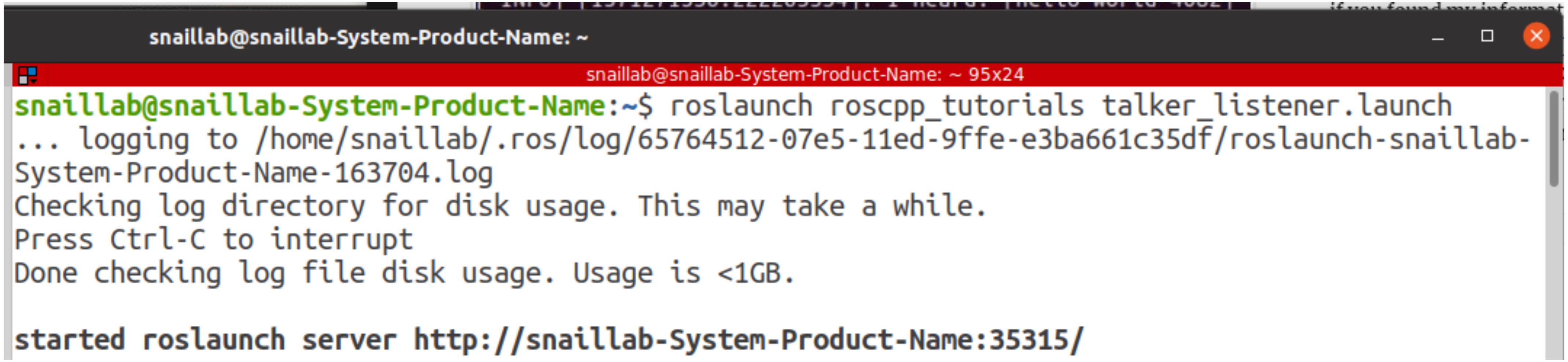
```
NODES

auto-starting new master
process[master]: started with pid [162920]
ROS_MASTER_URI=http://snaillab-System-Product-Name:11311/

setting /run_id to 65764512-07e5-11ed-9ffe-e3ba661c35df
process[rosout-1]: started with pid [162930]
started core service [/rosout]
```

```
snaillab@snaillab-System-Product-Name:~$ rostopic list
/chatter
/rosout
/rosout_agg
snaillab@snaillab-System-Product-Name:~$
```

```
[ INFO] [1658291931.222619038]: hello world 49
[ INFO] [1658291931.322627831]: hello world 50
[ INFO] [1658291931.422622438]: hello world 51
[ INFO] [1658291931.522631923]: hello world 52
[ INFO] [1658291931.622542691]: hello world 53
[ INFO] [1658291931.722627031]: hello world 54
[ INFO] [1658291931.822636693]: hello world 55
[ INFO] [1658291931.922629852]: hello world 56
[ INFO] [1658291932.022623506]: hello world 57
[ INFO] [1658291932.122629026]: hello world 58
[ INFO] [1658291932.222630594]: hello world 59
[ INFO] [1658291932.322633000]: hello world 60
[ INFO] [1658291932.422629623]: hello world 61
[ INFO] [1658291932.522641627]: hello world 62
[ INFO] [1658291932.622540254]: hello world 63
[ INFO] [1658291932.722633023]: hello world 64
[ INFO] [1658291932.822633968]: hello world 65
[ INFO] [1658291932.922633345]: hello world 66
[ INFO] [1658291933.022625775]: hello world 67
[ INFO] [1658291933.122628343]: hello world 68
[ INFO] [1658291933.222627917]: hello world 69
[ INFO] [1658291933.322632268]: hello world 70
[ INFO] [1658291933.422624061]: hello world 71
```

Use this command on a new terminal tab (File -> New Tab) to see a list of current active topics.

DIRECTIONS- Create a Hello World Project in ROS


snaillab@snaillab-System-Product-Name: ~
snaillab@snaillab-System-Product-Name: ~ 95x24

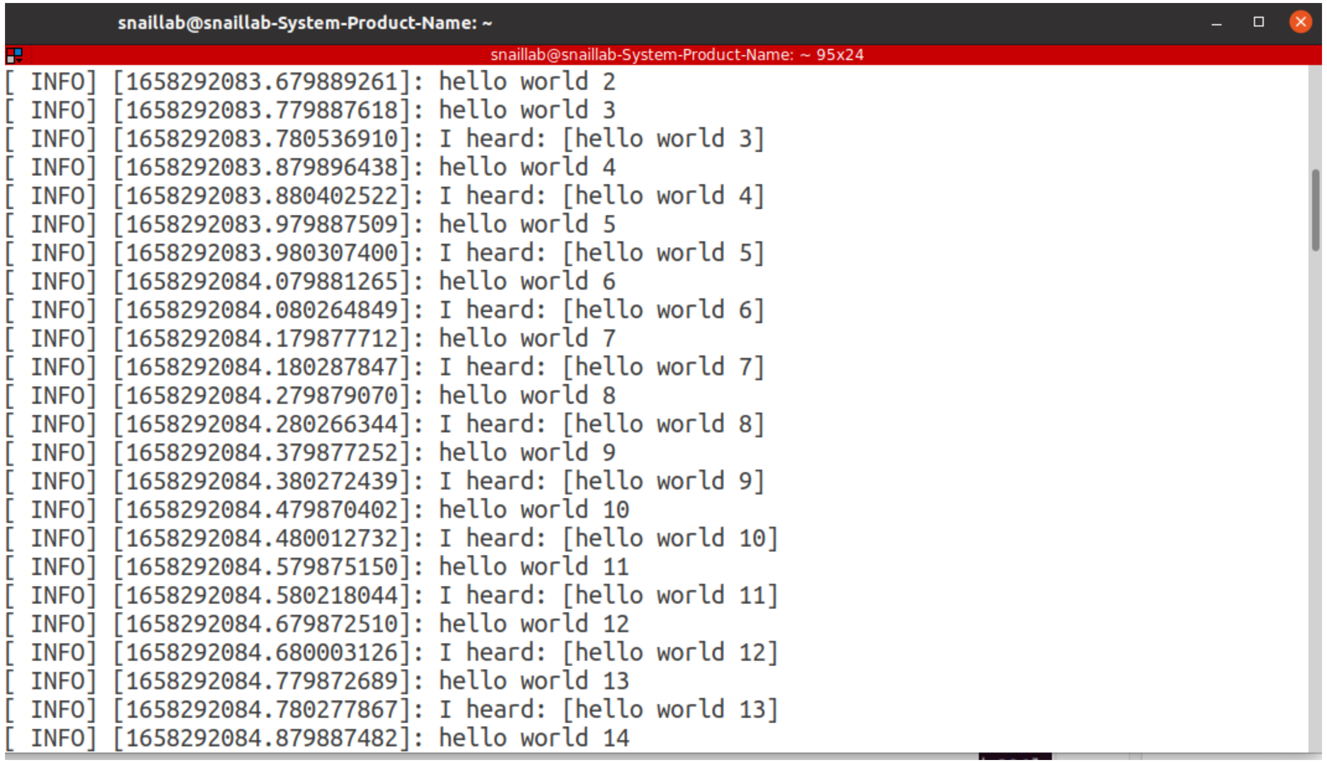
```
snaillab@snaillab-System-Product-Name:~$ roslaunch roscpp_tutorials talker_listener.launch
... logging to /home/snaillab/.ros/log/65764512-07e5-11ed-9ffe-e3ba661c35df/roslaunch-snaillab-
System-Product-Name-163704.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://snaillab-System-Product-Name:35315/
```

let’s start the **listener** node. The listener node will subscribe to the **/chatter** topic so that it can receive the hello world messages published by **talker.**

```
rosrun roscpp_tutorials listener
```

DIRECTIONS- Create a Hello World Project in ROS

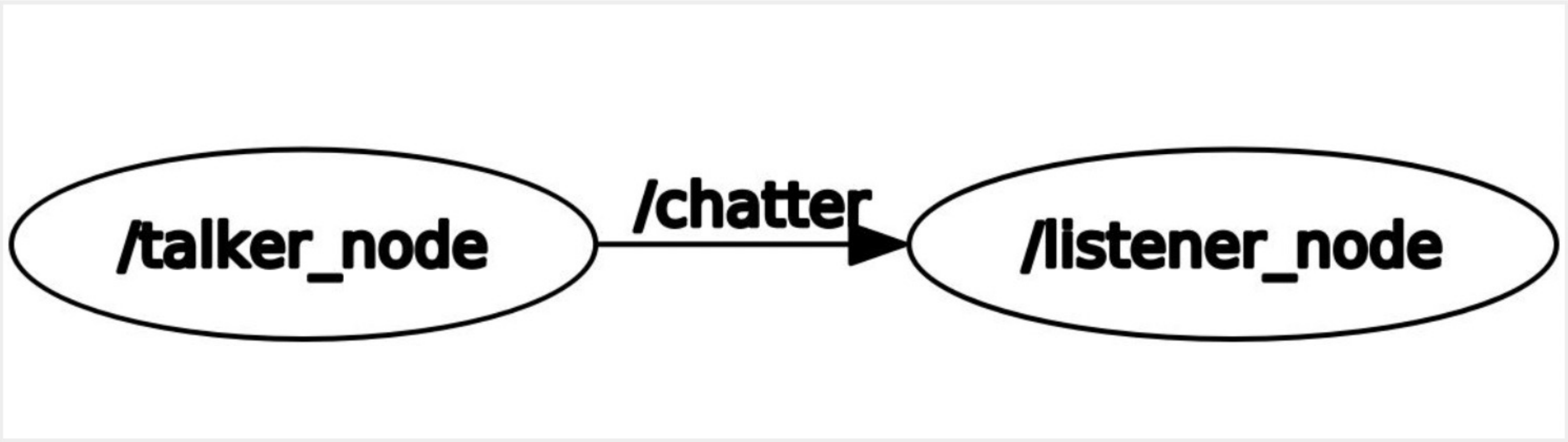
```
roslaunch roscpp_tutorials talker_listener.launch
```



snaillab@snaillab-System-Product-Name: ~
snaillab@snaillab-System-Product-Name: ~ 95x24

```
[ INFO] [1658292083.679889261]: hello world 2
[ INFO] [1658292083.779887618]: hello world 3
[ INFO] [1658292083.780536910]: I heard: [hello world 3]
[ INFO] [1658292083.879896438]: hello world 4
[ INFO] [1658292083.880402522]: I heard: [hello world 4]
[ INFO] [1658292083.979887509]: hello world 5
[ INFO] [1658292083.980307400]: I heard: [hello world 5]
[ INFO] [1658292084.079881265]: hello world 6
[ INFO] [1658292084.080264849]: I heard: [hello world 6]
[ INFO] [1658292084.179877712]: hello world 7
[ INFO] [1658292084.180287847]: I heard: [hello world 7]
[ INFO] [1658292084.279879070]: hello world 8
[ INFO] [1658292084.280266344]: I heard: [hello world 8]
[ INFO] [1658292084.379877252]: hello world 9
[ INFO] [1658292084.380272439]: I heard: [hello world 9]
[ INFO] [1658292084.479870402]: hello world 10
[ INFO] [1658292084.480012732]: I heard: [hello world 10]
[ INFO] [1658292084.579875150]: hello world 11
[ INFO] [1658292084.580218044]: I heard: [hello world 11]
[ INFO] [1658292084.679872510]: hello world 12
[ INFO] [1658292084.680003126]: I heard: [hello world 12]
[ INFO] [1658292084.779872689]: hello world 13
[ INFO] [1658292084.780277867]: I heard: [hello world 13]
[ INFO] [1658292084.879887482]: hello world 14
```

DIRECTIONS- How to Visualize Nodes Using the RQt GUI Tool



The rqt tool in ROS enables us to visualize the node connections while a launch file is running

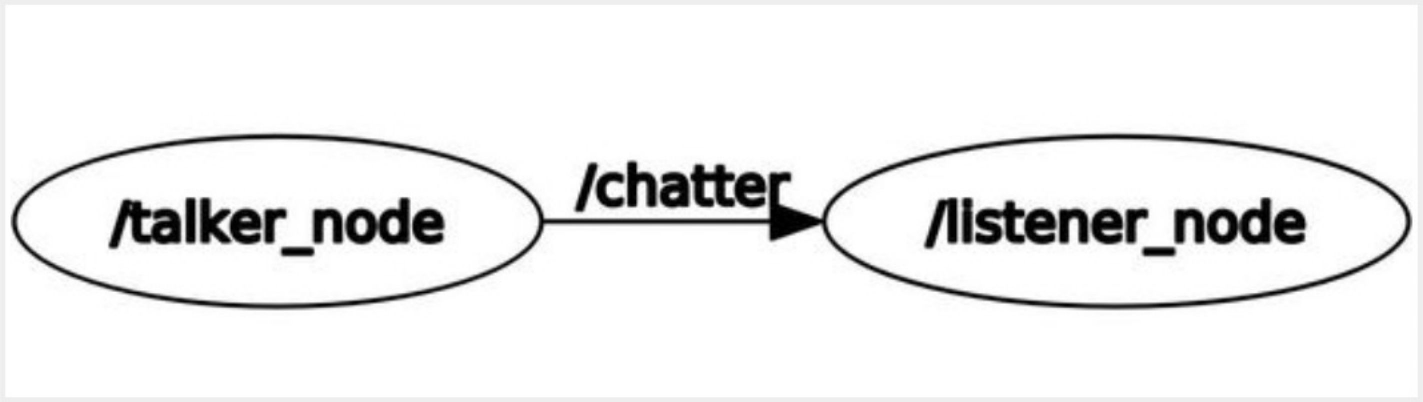
DIRECTIONS- How to Visualize Nodes Using the RQt GUI Tool

Type:

```
roslaunch hello_world talker_listener.launch
```

Then in a new terminal window, type:

```
rqt_graph
```



You can see that the listener__node is subscribed to the **/chatter** topic. The talker__node is publishing to the /chatter topic. You might also see another node called /rosout

DIRECTIONS- Working With rosbags in ROS Noetic

We’ll learn the basics of **rosbag**. rosbag is a tool that enables you to record messages that are published to a ROS topic. You can also replay the messages you recorded using rosbag.

The primary use cases for rosbags are testing and troubleshooting your robotics applications as well as developing new functionality.

how to record and replay messages using rosbags

Most Common rosbag Commands

The main component of rosbags is the bag file. A bag file is a formatted file that contains timestamped ROS messages.
The syntax for **creating a bag file** is as follows:

DIRECTIONS- Working With rosbags in ROS Noetic

The main component of rosbags is the bag file. A bag file is a formatted file that contains timestamped ROS messages.
The syntax for **creating a bag file** is as follows:

```
rosbag record -O filename.bag topic-names
```

For example, if you want to record messages that are published to the **/turtle1/cmd_vel** and **/turtle1/pose** topics, you would type this command:

```
rosbag record -O filename.bag /turtle1/cmd_vel /turtle1/pose
```

If you want to **record the messages of all published topics** that are currently active, you would use the following command:

```
rosbag record -a
```

DIRECTIONS- Working With rosbags in ROS Noetic

Example: how to record and replay messages using rosbags.

Open a new terminal window, and launch ROS.

```
roscore
```

In another terminal tab, type the following command to launch the turtle simulation:

```
rosrun turtlesim turtlesim_node
```

In another terminal tab, type the following command to get a turtle to repeatedly move in a square-shaped pattern:

```
rosrun turtlesim draw_square
```

DIRECTIONS- Working With rosbags in ROS Noetic

Open another terminal tab, and check out the topics that are currently active:

```
rostopic list -v
```

Let’s record the messages that are publishing to the **/turtle1/cmd_vel** and **/turtle1/pose** topics. We’ll store these messages in a bag file.

In a new terminal tab, create a new folder:

```
mkdir ~/bagfiles
```

Move inside the directory you just created.

```
cd ~/bagfiles
```

DIRECTIONS- Working With rosbags in ROS Noetic

Start recording.

```
rosbag record -O turtle_square_sim.bag /turtle1/cmd_vel /turtle1/pose
```

Go back to the terminal where you launched the draw_square node (don’t shutdown turtlesim or the ROS Master though).

Press CTRL + C to get the turtle to stop drawing. And In a new terminal tab, type:

```
cd ~/bagfiles
```

```
rosbag play turtle_square_sim.bag
```

DIRECTIONS- How to Launch the TurtleBot3 Simulation With ROS

In this tutorial, we will launch a virtual robot called **TurtleBot3**. TurtleBot3 is a low-cost, personal robot kit with open-source software.

http://wiki.ros.org/Robots/TurtleBot

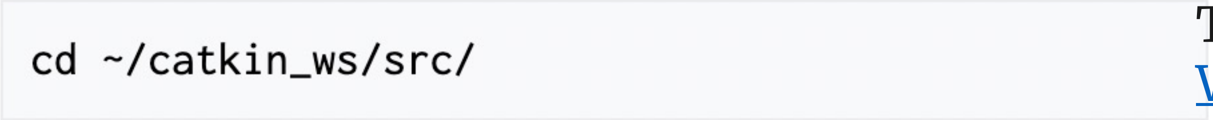
SLAM (Simultaneous localization and mapping) and autonomous navigation.

TurtleBot3 is designed to run using just ROS and Ubuntu. It is a popular robot for research and educational purposes.

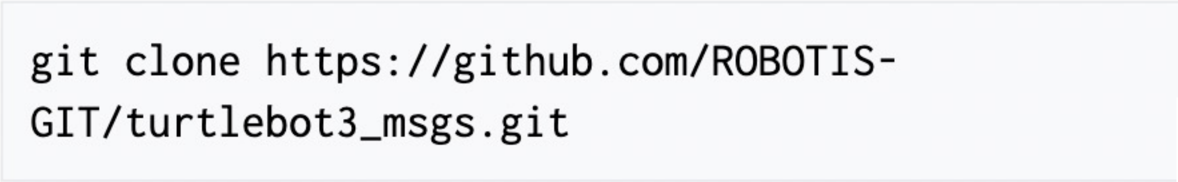
Reference
https://emanual.robotis.com/docs/en/platform/turtlebot3/simulation/#gazebo-simulation

DIRECTIONS- How to Launch the TurtleBot3 Simulation With ROS

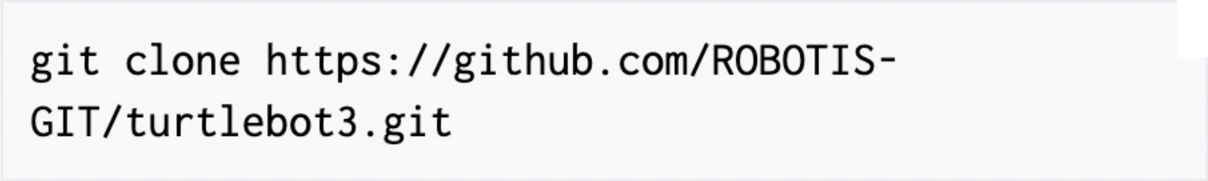
Open a terminal window and install the dependent packages. Enter the following commands, one right after the other:



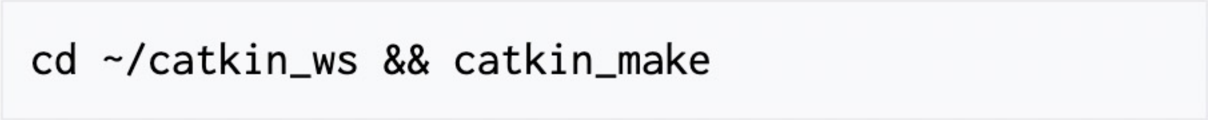
```
cd ~/catkin_ws/src/
```



```
git clone https://github.com/ROBOTIS-
GIT/turtlebot3_msgs.git
```

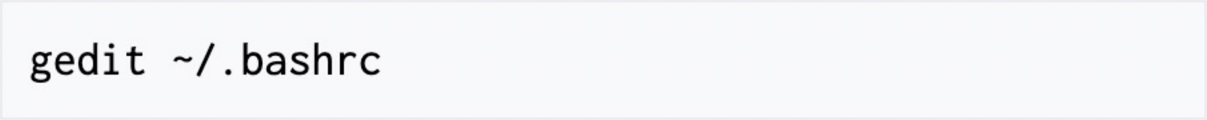


```
git clone https://github.com/ROBOTIS-
GIT/turtlebot3.git
```



```
cd ~/catkin_ws && catkin_make
```

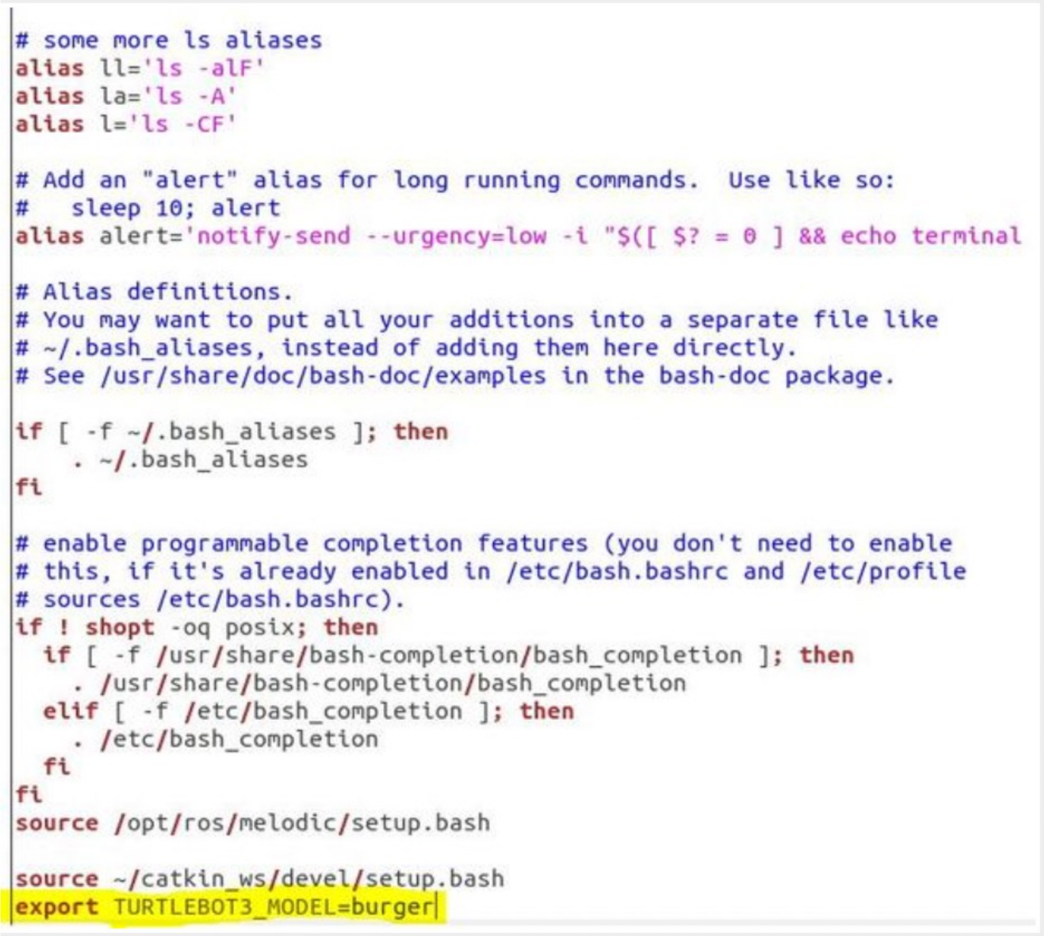
TurtleBot3 has three models, Burger, Waffle, and Waffle Pi, so you have to set which model you want to use before you launch TurtleBot3. Type this command to open the bashrc file to add this setting:



```
gedit ~/.bashrc
```

DIRECTIONS- How to Launch the TurtleBot3 Simulation With ROS

Add this line at the bottom of the file:



```
# some more ls aliases
alias ll='ls -alF'
alias la='ls -A'
alias l='ls -CF'

# Add an "alert" alias for long running commands.  Use like so:
#   sleep 10; alert
alias alert='notify-send --urgency=low -i "$([ $? = 0 ] && echo terminal

# Alias definitions.
# You may want to put all your additions into a separate file like
# ~/.bash_aliases, instead of adding them here directly.
# See /usr/share/doc/bash-doc/examples in the bash-doc package.

if [ -f ~/.bash_aliases ]; then
    . ~/.bash_aliases
fi

# enable programmable completion features (you don't need to enable
# this, if it's already enabled in /etc/bash.bashrc and /etc/profile
# sources /etc/bash.bashrc).
if ! shopt -oq posix; then
  if [ -f /usr/share/bash-completion/bash_completion ]; then
    . /usr/share/bash-completion/bash_completion
  elif [ -f /etc/bash_completion ]; then
    . /etc/bash_completion
  fi
fi
source /opt/ros/melodic/setup.bash

source ~/catkin_ws/devel/setup.bash
export TURTLEBOT3_MODEL=burger
```

Save the file and close it.

Now, we need to download the TurtleBot3 simulation files.

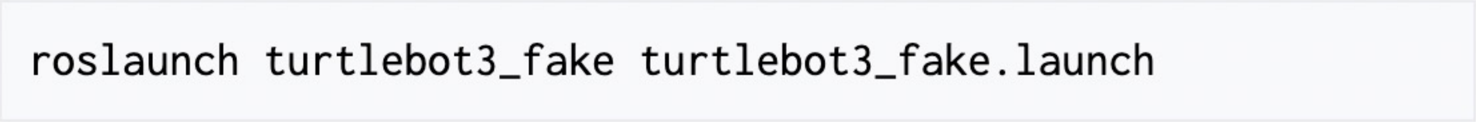
```
cd ~/catkin_ws/src/
```

```
git clone https://github.com/ROBOTIS-
GIT/turtlebot3_simulations.git
```

```
cd ~/catkin_ws && catkin_make
```

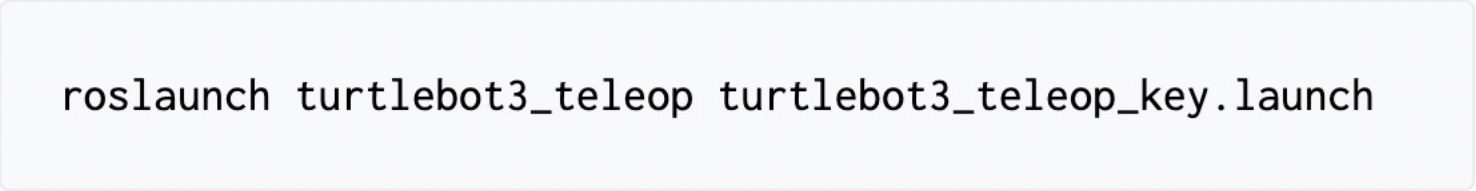
DIRECTIONS- Simulate TurtleBot3 Using RViz

let’s launch the virtual robot using **RViz**. Type this command in your terminal window:



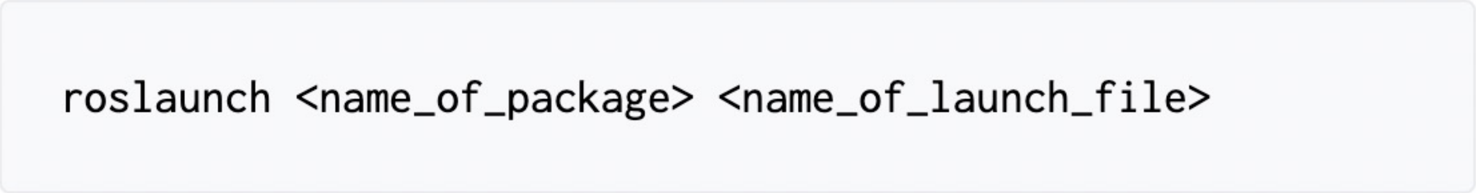
```
roslaunch turtlebot3_fake turtlebot3_fake.launch
```

If you want to move TurtleBot3 around the screen, open a new terminal window, and type the following command (everything on one line in the terminal):



```
roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch
```

roslaunch is the command in ROS that enables us to launch a program. The syntax is as follows:



```
roslaunch <name_of_package> <name_of_launch_file>
```

DIRECTIONS- How to Launch the TurtleBot3 Simulation With ROS

What is a Package?

ROS packages are the way software is organized in ROS. They are the smallest thing you can build in ROS.

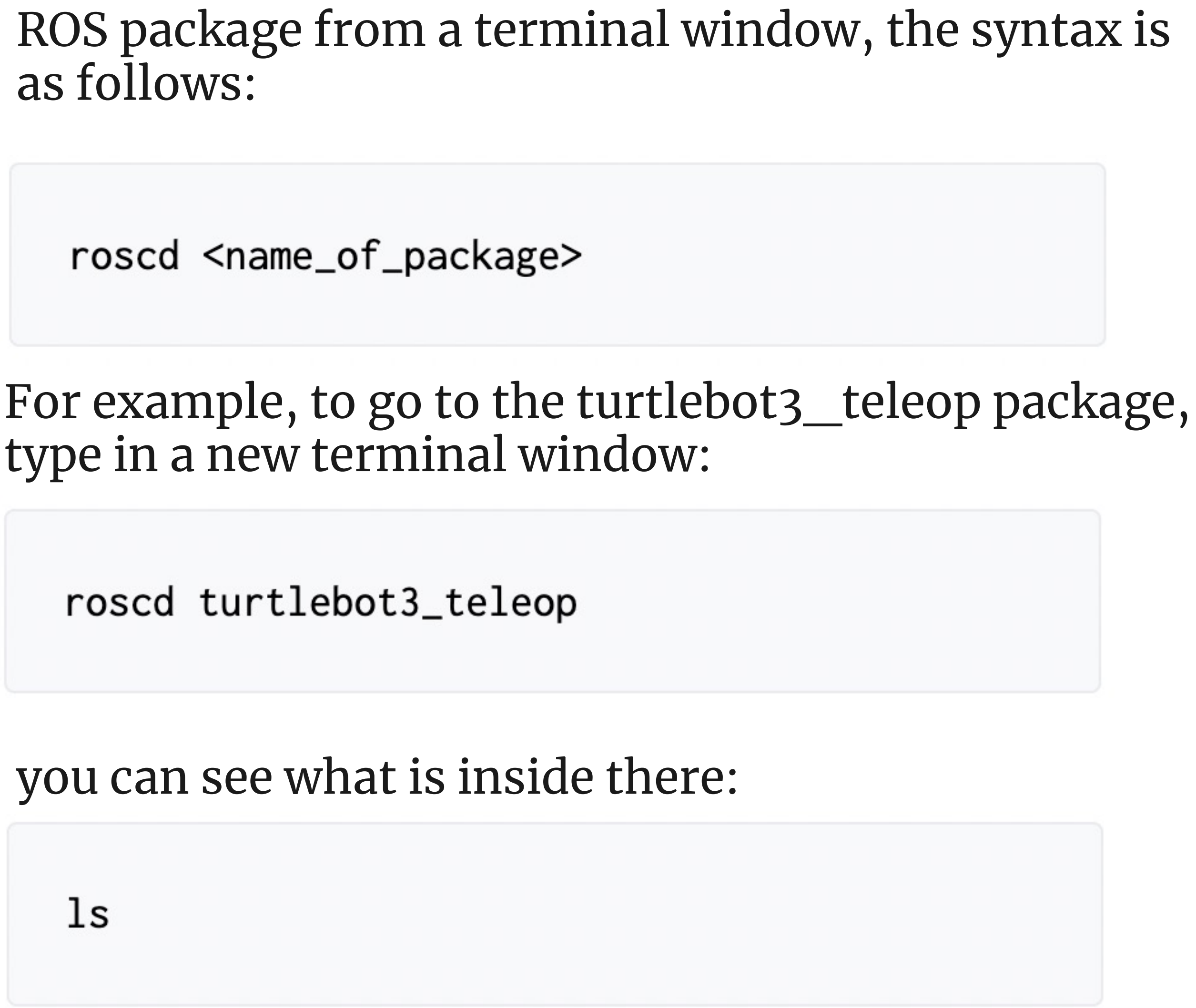
A package is a directory that contains all of the files, programs, libraries, and datasets needed to provide some useful functionality. ROS packages promote software reuse. **Every program that you write in ROS will need to be inside a package.**

The goal of a ROS package is to be large enough to be useful but not so large and complicated that nobody wants to reuse it for their own project.

ROS packages are organized as follows:

- launch folder: Contains launch files
- src folder: Contains the source code (C++, Python)
- CMakeLists.txt: List of cmake rules for compilation
- package.xml: Package information and dependencies

DIRECTIONS- How to Launch the TurtleBot3 Simulation With ROS



ROS package from a terminal window, the syntax is as follows:

```
roscd <name_of_package>
```

For example, to go to the turtlebot3_teleop package, type in a new terminal window:

```
roscd turtlebot3_teleop
```

you can see what is inside there:

```
ls
```

DIRECTIONS- How to Launch the TurtleBot3 Simulation With ROS

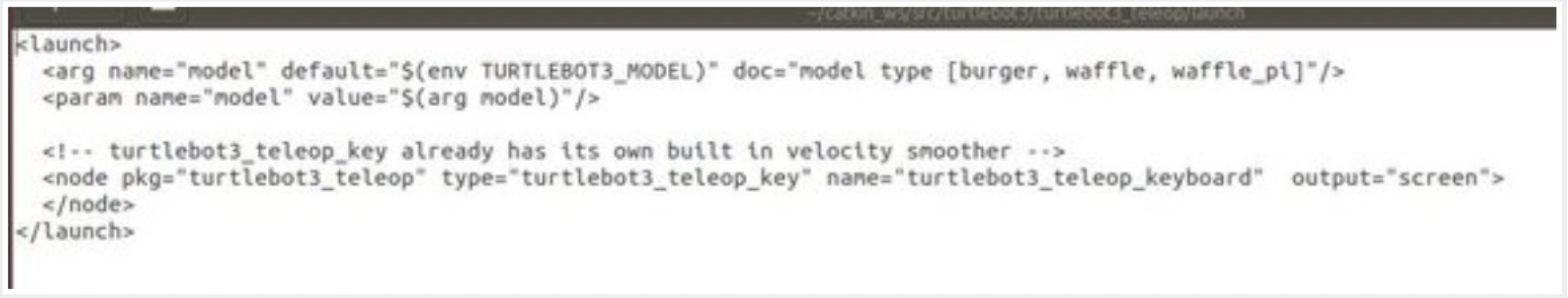
What is a Launch File?

From within the turtlebot3_teleop package, move inside the launch file.

```
cd launch
```

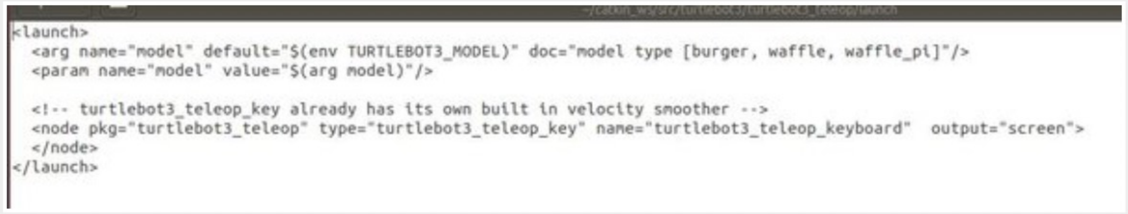
Let’s take a look inside it.

```
gedit turtlebot3_teleop_key.launch
```



```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <param name="model" value="$(arg model)"/>

  <!-- turtlebot3_teleop_key already has its own built in velocity smoother -->
  <node pkg="turtlebot3_teleop" type="turtlebot3_teleop_key" name="turtlebot3_teleop_keyboard"  output="screen">
  </node>
</launch>
```

DIRECTIONS- How to Launch the TurtleBot3 Simulation With ROS



```
<launch>
  <arg name="model" default="$(env TURTLEBOT3_MODEL)" doc="model type [burger, waffle, waffle_pi]"/>
  <param name="model" value="$(arg model)"/>

  <!-- turtlebot3_teleop_key already has its own built in velocity smoother -->
  <node pkg="turtlebot3_teleop" type="turtlebot3_teleop_key" name="turtlebot3_teleop_keyboard"  output="screen">
  </node>
</launch>
```

All launch files start off with the **<launch>** tag and end with the **</launch>** tag. Inside these tags, you have the <node> tag that contains the following parameters:

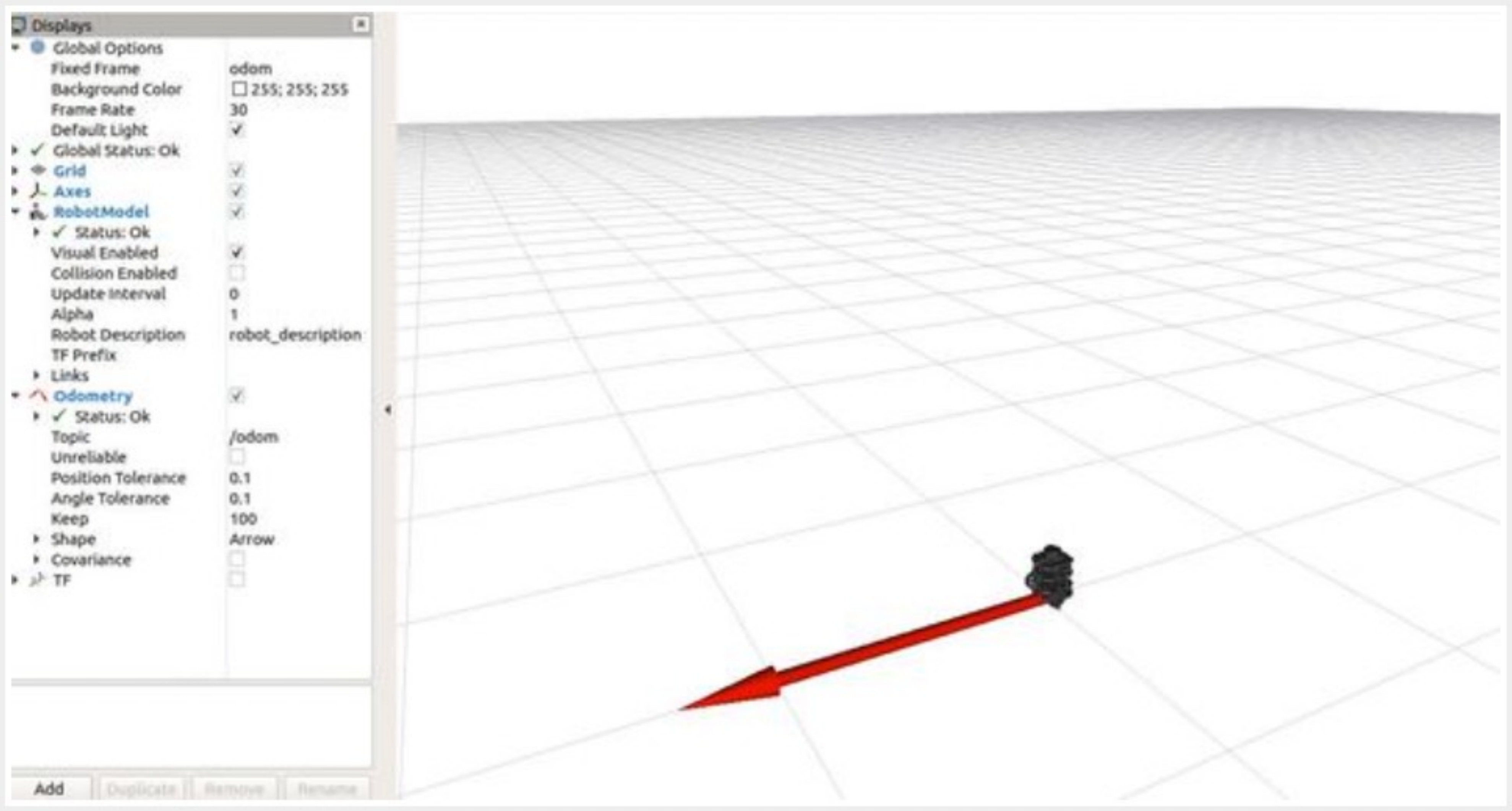
1.pkg=”package_name”: This is the name of the package that has the code we want ROS to execute.

2.type=”python_file_name.py”: This is the name of the program we’d like to execute.

3.name=”node_name”: This is the name of the ROS node we want to launch our program.

4.output=”type_of_output”: Where you will print the output of the program.

DIRECTIONS- How to Launch the TurtleBot3 Simulation With ROS



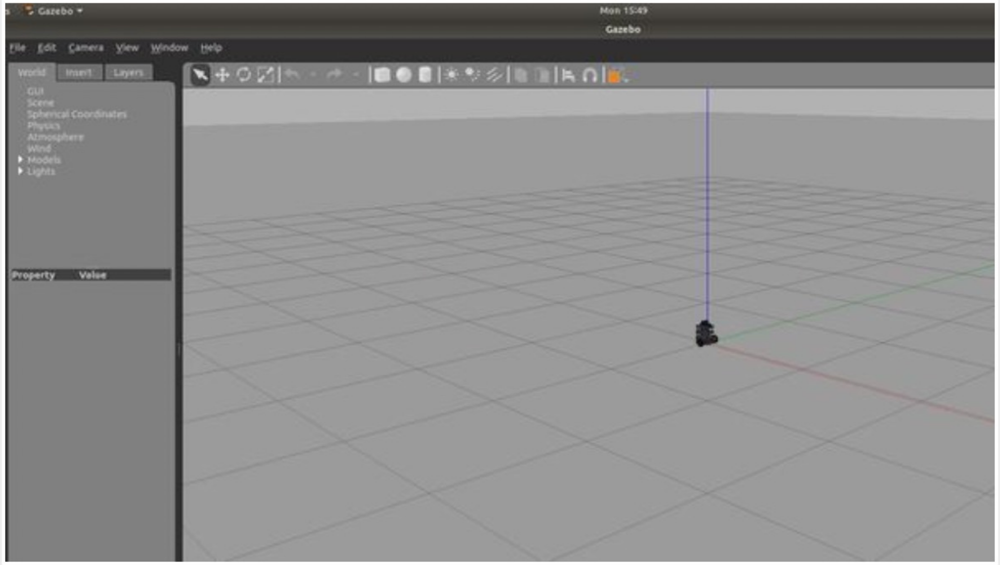
Displays

Global Options	
Fixed Frame	odom
Background Color	□ 255; 255; 255
Frame Rate	30
Default Light	✓
✓ Global Status: Ok	
Grid	✓
Axes	✓
RobotModel	✓
✓ Status: Ok	
Visual Enabled	✓
Collision Enabled	□
Update Interval	0
Alpha	1
Robot Description	robot_description
TF Prefix	
Links	
Odometry	✓
✓ Status: Ok	
Topic	/odom
Unreliable	□
Position Tolerance	0.1
Angle Tolerance	0.1
Keep	100
Shape	Arrow
Covariance	□
TF	□

DIRECTIONS- Simulate TurtleBot3 Using Gazebo

Now let’s use **Gazebo** to do the TurtleBot3 simulation.
First, let’s launch TurtleBot3 in an empty environment. Type this command (everything goes on one line):

```
roslaunch turtlebot3_gazebo turtlebot3_empty_world.launch
```

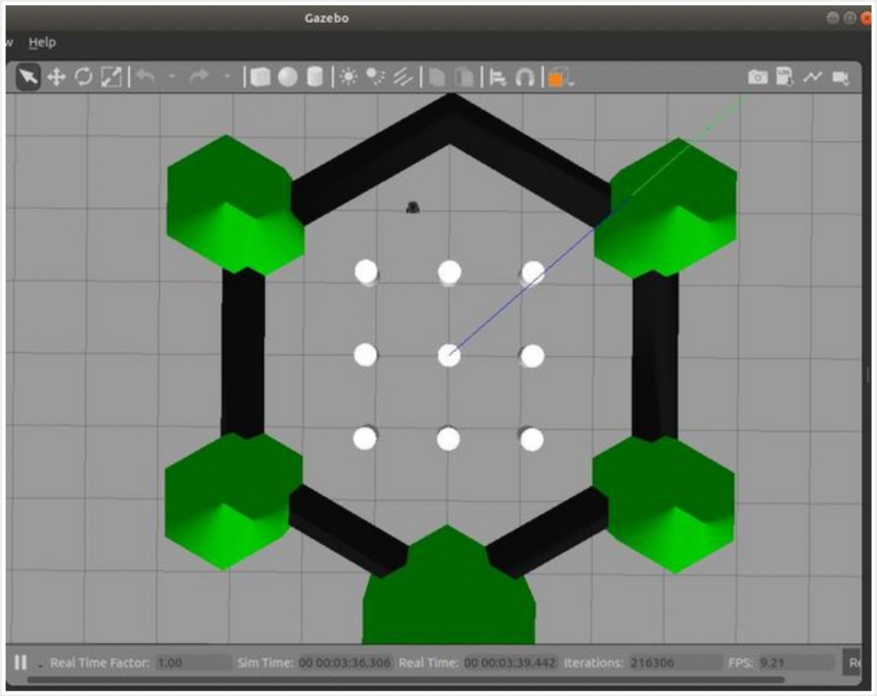


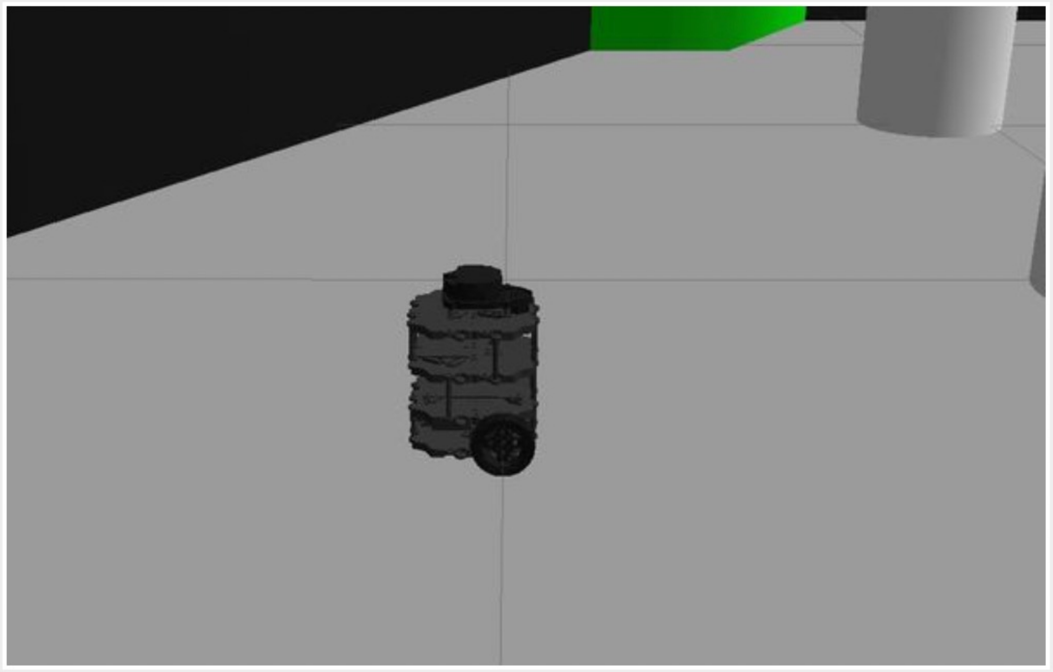
DIRECTIONS- How to Change the Simulation Environment for TurtleBot3

This environment is often used for testing SLAM and navigation algorithms. Simultaneous localization and mapping (SLAM) concerns the problem of a robot building or updating a map of an unknown environment while simultaneously keeping track its location in that environment.

In a new terminal window type:

```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```





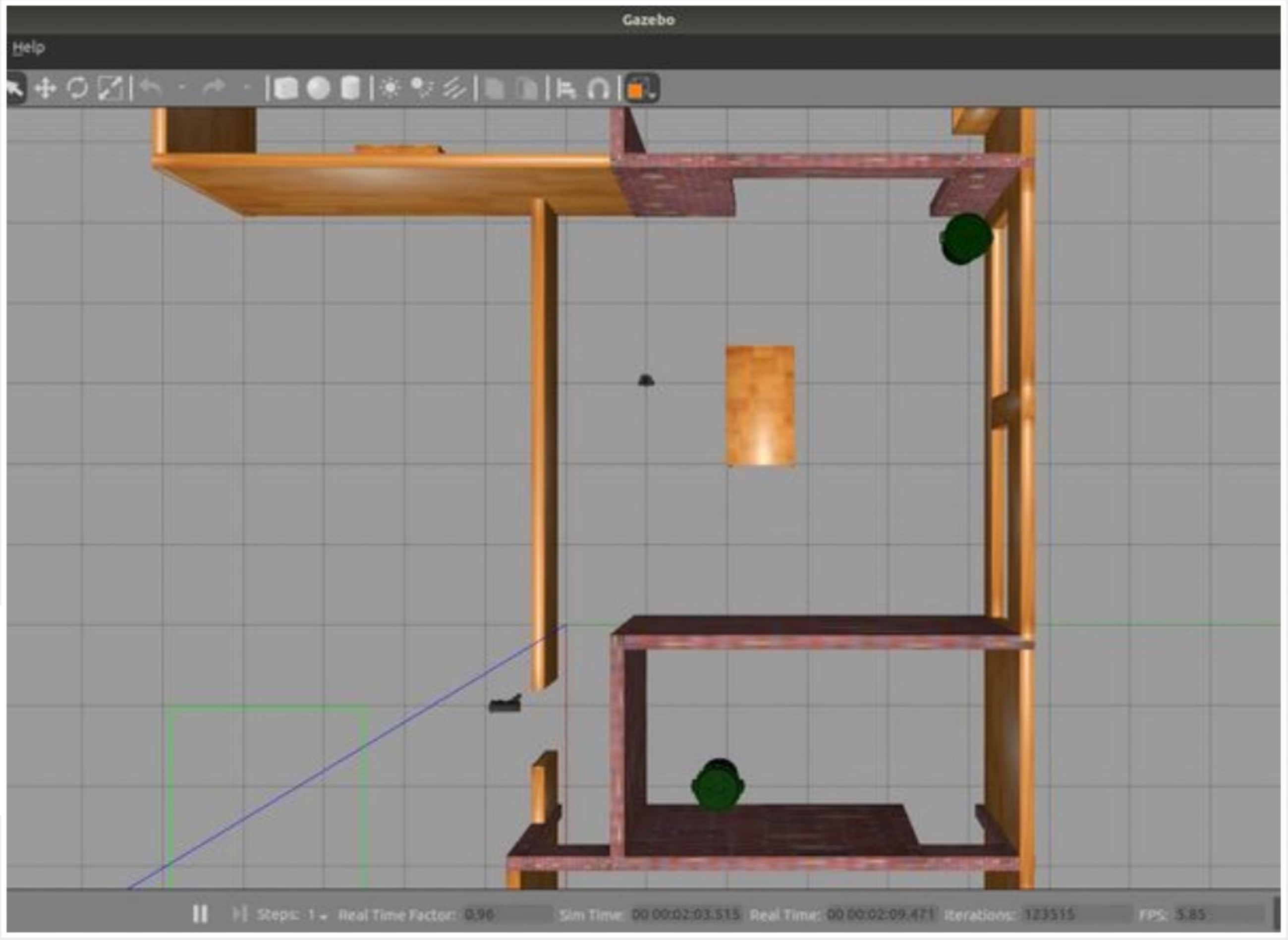
DIRECTIONS- How to Change the Simulation Environment for TurtleBot3

TurtleBot3 inside a house. Type this command and wait a few minutes for the environment to load.

```
roslaunch turtlebot3_gazebo
turtlebot3_house.launch
```

To move the TurtleBot with your keyboard, use this command in another terminal tab:

```
roslaunch turtlebot3_teleop
turtlebot3_teleop_key.launch
```

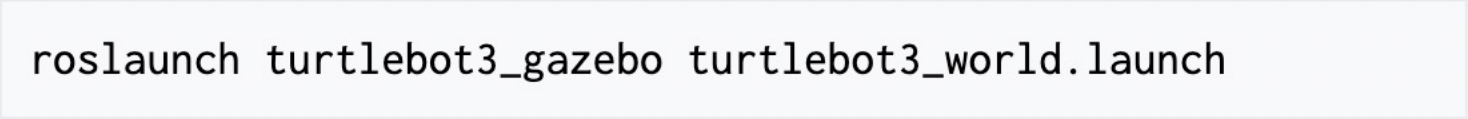


Steps: 1 Real Time Factor: 0.96 Sim Time: 00 00:02:03.515 Real Time: 00 00:02:09.471 Iterations: 123515 FPS: 5.85

DIRECTIONS- Autonomous Navigation and Obstacle Avoidance With TurtleBot3

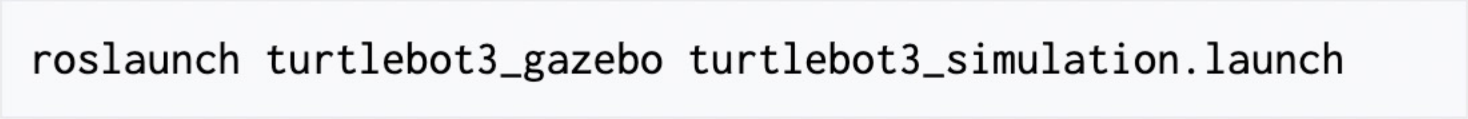
let’s implement obstacle avoidance for the TurtleBot3 robot. The goal is to have TurtleBot3 autonomously navigate around a room and avoid colliding into objects.

Open a new terminal and type:



```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

In another terminal window type:



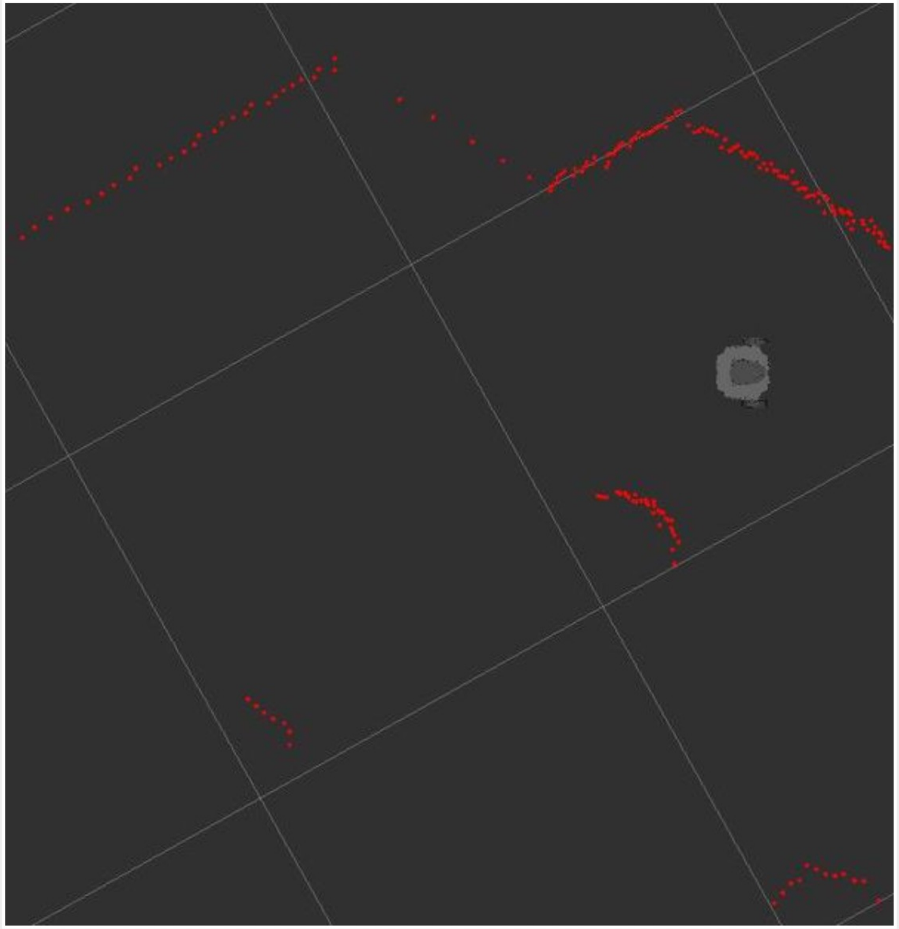
```
roslaunch turtlebot3_gazebo turtlebot3_simulation.launch
```

TurtleBot3 autonomously moving about the world and avoiding obstacles along the way.

DIRECTIONS- Autonomous Navigation and Obstacle Avoidance With TurtleBot3

We can open RViz to visualize the LaserScan topic while TurtleBot3 is moving about in the world. In a new terminal tab type:

```
roslaunch turtlebot3_gazebo turtlebot3_gazebo_rviz.launch
```



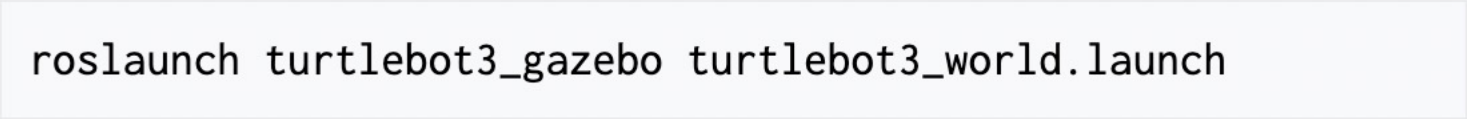
DIRECTIONS- Simulating SLAM With TurtleBot3

how we can simulate SLAM with TurtleBot3. As a refresher, Simultaneous localization and mapping (SLAM) concerns the problem of a robot building or updating a map of an unknown environment while simultaneously keeping track its location in that environment.

Install the SLAM module in a new terminal window.

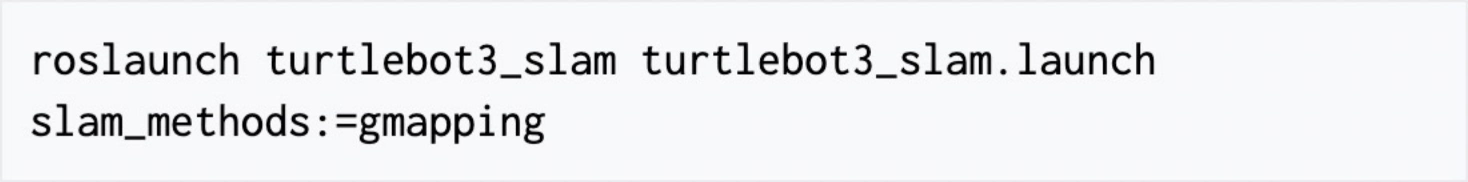
sudo apt install ros-noetic-slam-gmapping

Start Gazebo in a new terminal window.



```
roslaunch turtlebot3_gazebo turtlebot3_world.launch
```

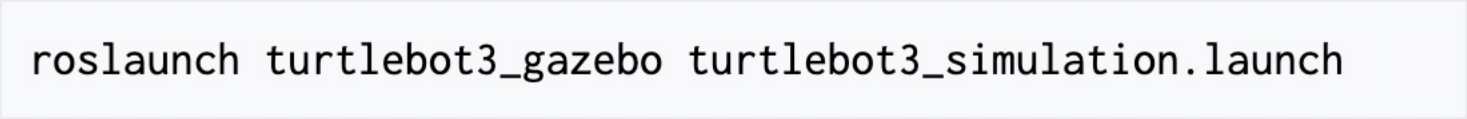
Start SLAM in a new terminal tab.



```
roslaunch turtlebot3_slam turtlebot3_slam.launch
slam_methods:=gmapping
```

DIRECTIONS- Autonomous Navigation and Obstacle Avoidance With TurtleBot3

Start autonomous navigation in a new terminal tab:

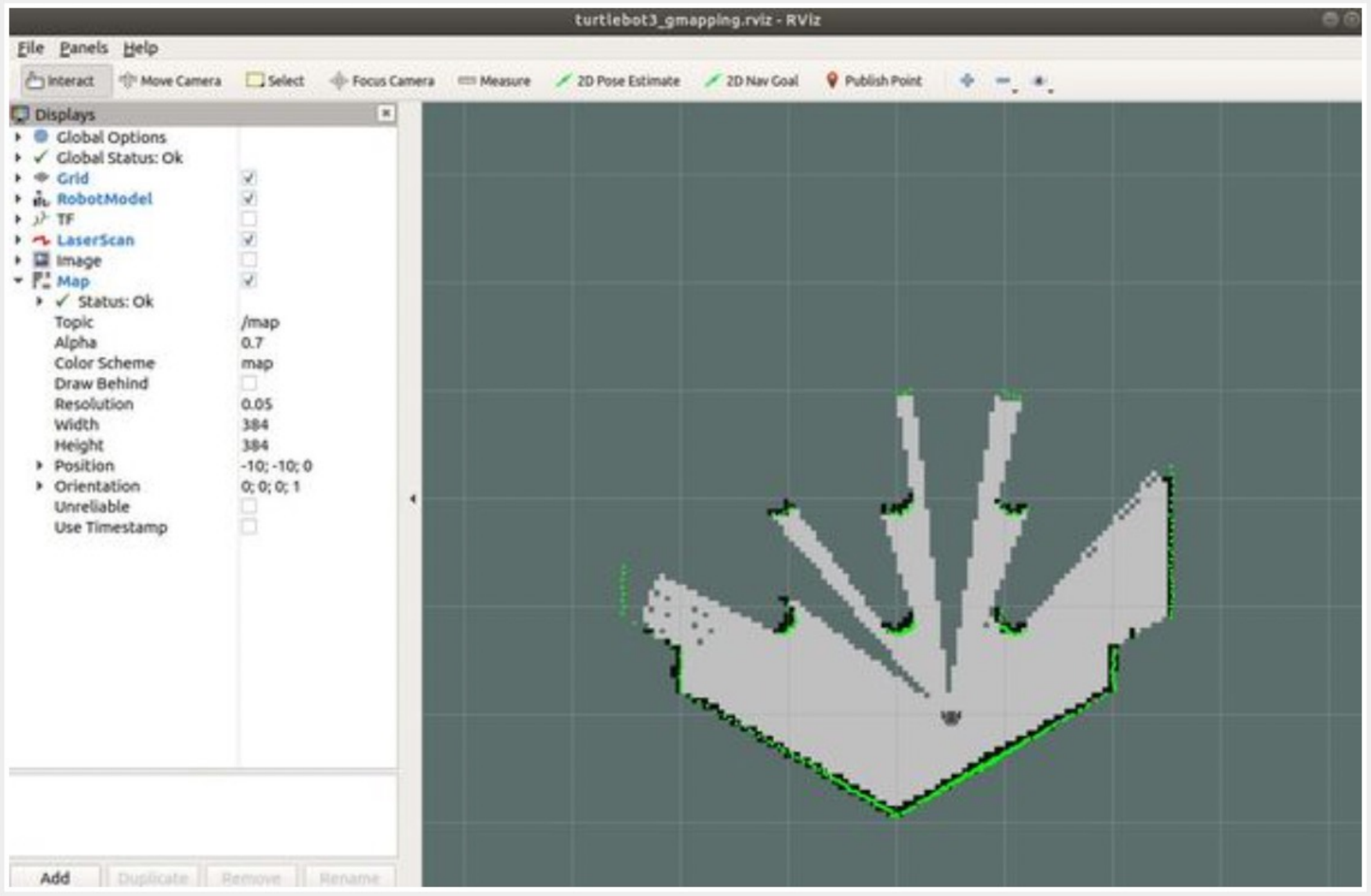


```
roslaunch turtlebot3_gazebo turtlebot3_simulation.launch
```

Watch the robot create a map of the environment as it autonomously moves from place to place!

DIRECTIONS- Autonomous Navigation and Obstacle Avoidance With TurtleBot3

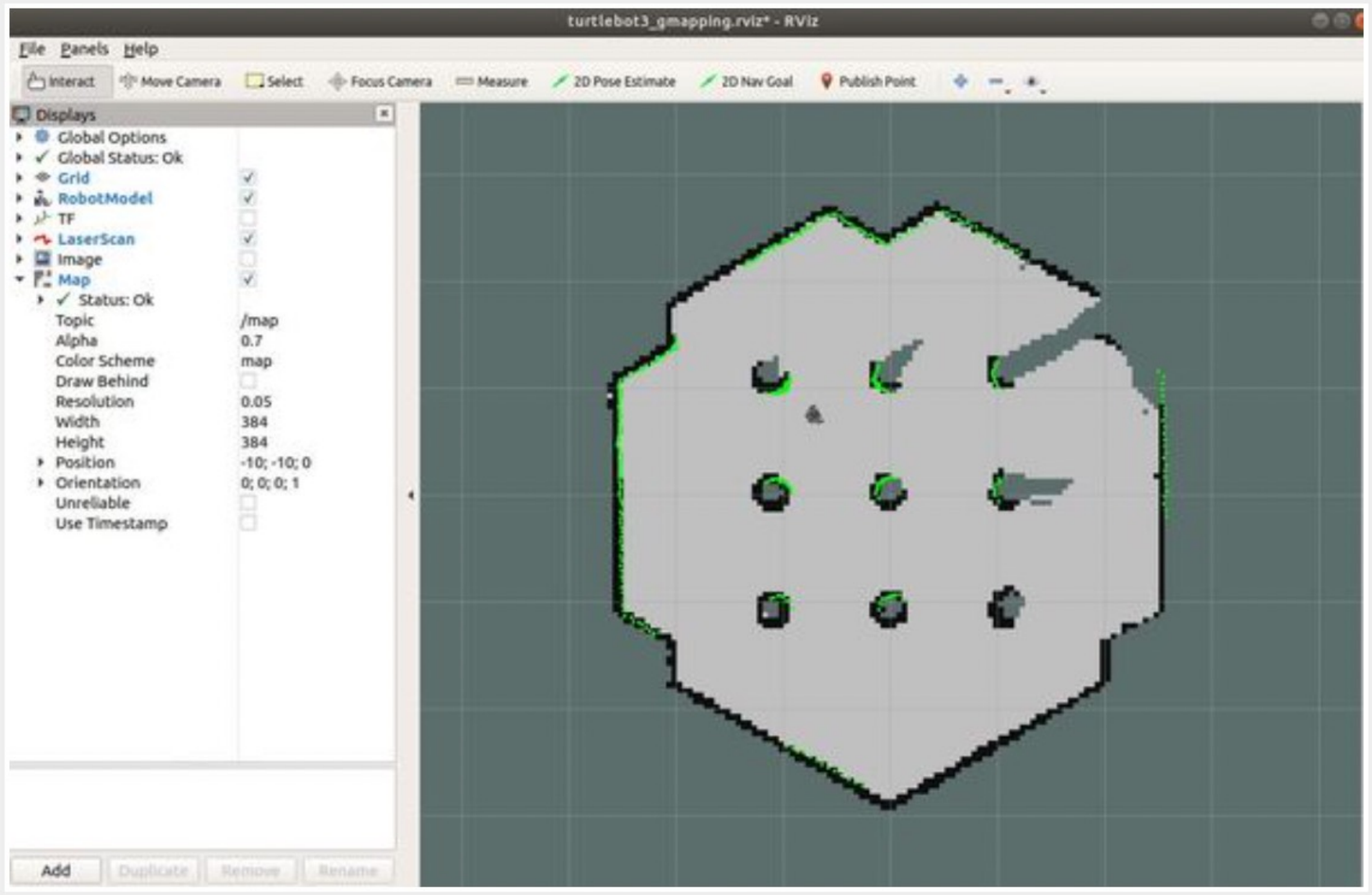
Watch the robot create a map of the environment as it autonomously moves from place to place!



Global Options	
✓ Global Status: Ok	
Grid	☑
RobotModel	☑
TF	□
LaserScan	☑
Image	□
Map	☑
✓ Status: Ok	
Topic	/map
Alpha	0.7
Color Scheme	map
Draw Behind	□
Resolution	0.05
Width	384
Height	384
Position	-10; -10; 0
Orientation	0; 0; 0; 1
Unreliable	□
Use Timestamp	□

DIRECTIONS- Autonomous Navigation and Obstacle Avoidance With TurtleBot3

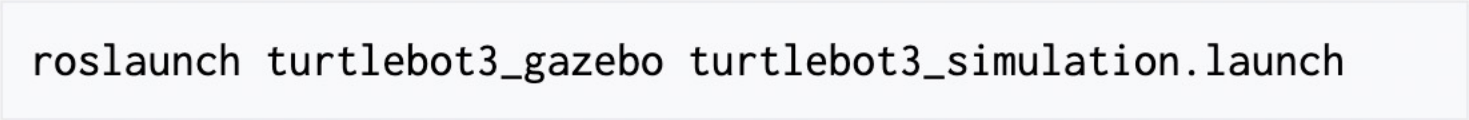
Watch the robot create a map of the environment as it autonomously moves from place to place!



Global Options	
✓ Global Status: Ok	
Grid	☑
RobotModel	☑
TF	□
LaserScan	☑
Image	□
Map	☑
✓ Status: Ok	
Topic	/map
Alpha	0.7
Color Scheme	map
Draw Behind	□
Resolution	0.05
Width	384
Height	384
Position	-10; -10; 0
Orientation	0; 0; 0; 1
Unreliable	□
Use Timestamp	□

DIRECTIONS- Autonomous Navigation and Obstacle Avoidance With TurtleBot3

Start autonomous navigation in a new terminal tab:



```
roslaunch turtlebot3_gazebo turtlebot3_simulation.launch
```

Watch the robot create a map of the environment as it autonomously moves from place to place!