ROBT 403 Laboratory Report 1

Daniil Filimonov, Abdirakhman Onabek, Kir Smolyarchuk

GitHub:

https://github.com/Meneralllka/FailTeamLab1/tree/master

1. Introduction to ROS (Robot Operating System)

The Robot Operating System (ROS) is an open-source, flexible framework for writing software for robots. Initially developed by Willow Garage in 2007, ROS has since become a standard in the robotics industry, supporting a vast range of robotic platforms and applications. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionalities, message-passing between processes, and package management.

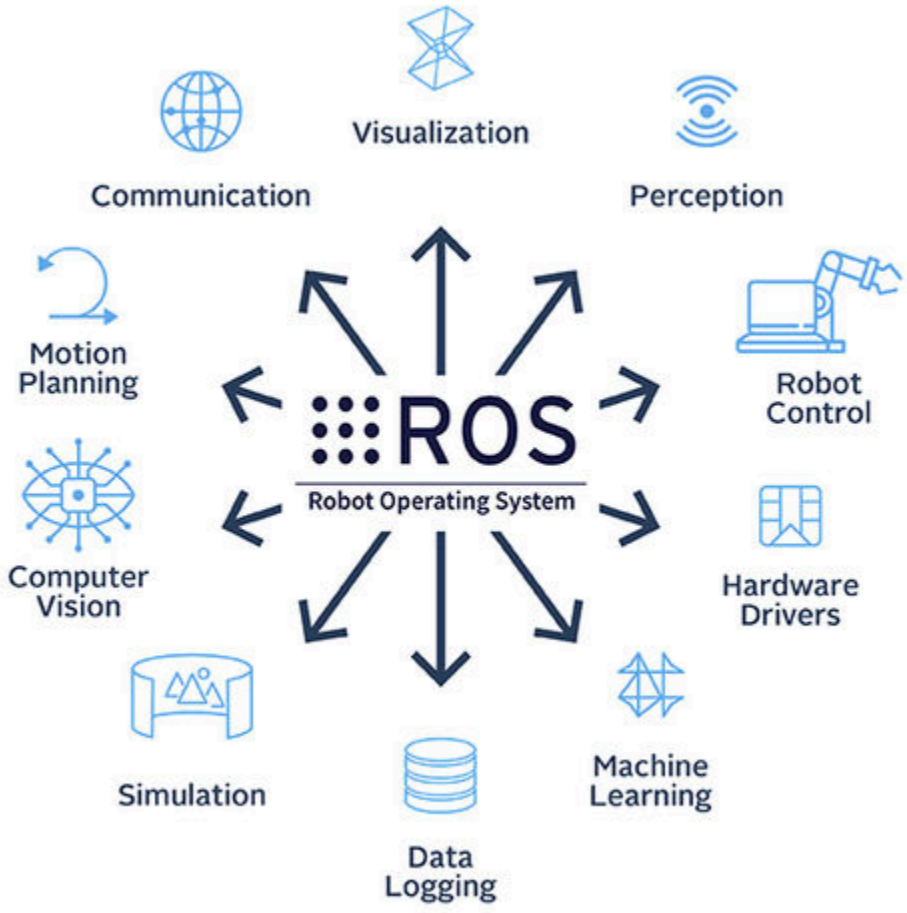


Figure 1. Versatility of ROS

ROS is designed to be highly modular, allowing developers to build complex robotic systems by connecting smaller, reusable components known as "nodes." Each node is responsible for a specific task, such as sensor data processing, motor control, or decision-making, and communicates with other nodes using a publish-subscribe messaging model.

A significant advantage of ROS is its extensive ecosystem, which includes a vast library of packages and tools developed by a global community of researchers and engineers. These resources enable rapid development and prototyping, making ROS an ideal choice for both academic research and commercial robotics applications.

2. Real-world applications

a. HERB (Home Exploring Robot Butler)

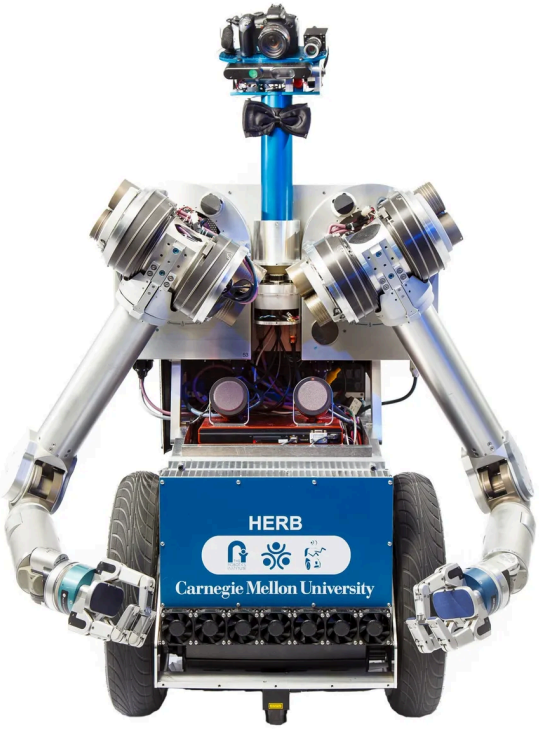


Figure 2. HERB Robot (Carnegie Mellon University)

Overview: HERB is a mobile manipulator robot developed by Carnegie Mellon University's Personal Robotics Lab. It is designed to assist with tasks in a domestic environment, such as fetching items, opening doors, and even serving food. HERB uses ROS for integrating its various sensors, planning algorithms, and manipulation tasks.

ROS Integration: HERB utilizes ROS to coordinate its perception, planning, and control systems. The robot's ability to interact with and manipulate objects in unstructured environments is heavily reliant on ROS's modular framework.

b. Husky UGV (Unmanned Ground Vehicle)

Figure 3. Husky UGV (Clearpath Robotics)

Overview: Husky is a rugged, all-terrain unmanned ground vehicle developed by Clearpath Robotics. It is designed for outdoor and industrial environments, where it can be used for research, mapping, and surveillance. Husky is known for its versatility and ability to carry heavy payloads.

ROS Integration: Husky uses ROS for its navigation, sensor integration, and control systems. The robot can be equipped with a variety of sensors and is commonly used in research projects that require autonomous navigation in challenging terrains.

3. Task 0 – ROS Tutorials

In this task, we had to do a 16 step basic tutorial from the official ROS web-site. Figure array below shows completion of all tasks.

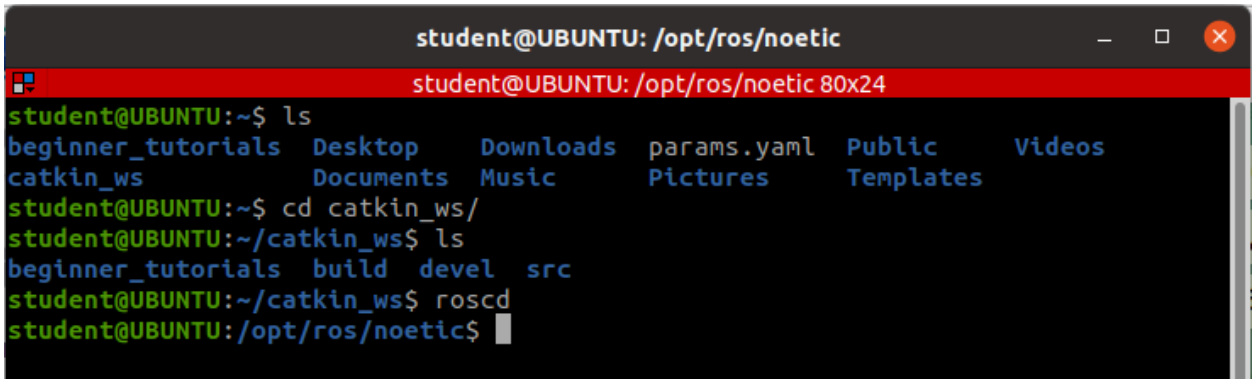


Figure 4. Setup ROS Environment (Tutorial 1)

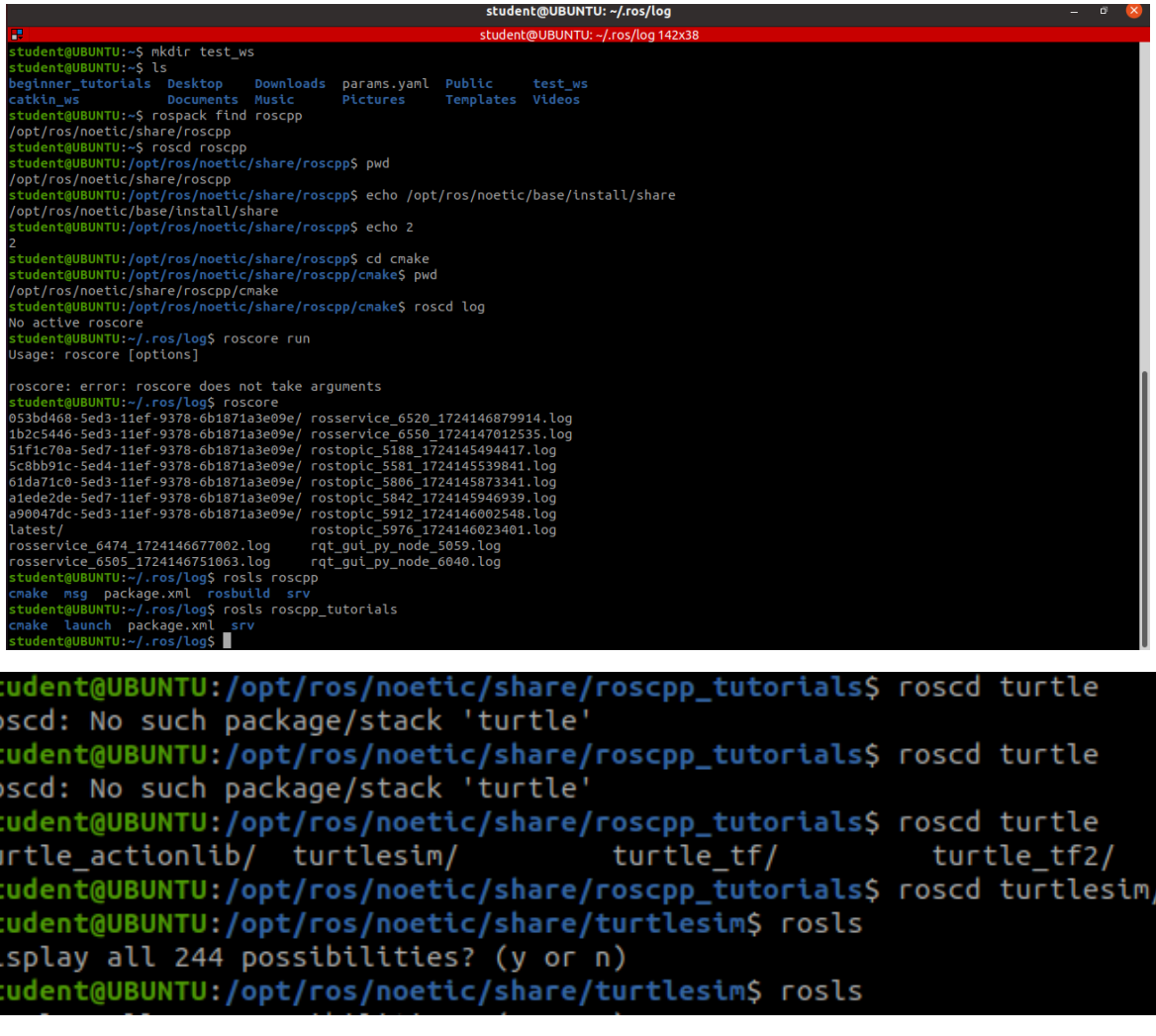


Figure 5. Navigating the ROS Filesystem (Tutorial 2)

```
student@UBUNTU:~/catkin_ws$ . ~/catkin_ws/devel/setup.bash
student@UBUNTU:~/catkin_ws$ rospack depends1 beginner_tutorials2
[rospack] Error: no such package beginner_tutorials2
student@UBUNTU:~/catkin_ws$ cd ~
student@UBUNTU:~$ rospack depends1 beginner_tutorials2
[rospack] Error: no such package beginner_tutorials2
student@UBUNTU:~$ rospack depends beginner_tutorials2
[rospack] Error: no such package beginner_tutorials2
student@UBUNTU:~$ rospack
cflags-only-I      depends-indent     depends-on1        help               libs-only-other    plugins            vcs
cflags-only-other  depends-manifests  depends-why        langs              list               profile            vcs0
depends            depends-msgsrv     export             libs-only-l        list-duplicates    rosdep
depends1           depends-on         find               libs-only-L        list-names         rosdep0
student@UBUNTU:~$ rospack depends1 beginner_tutorials
roscpp
rospy
std_msgs
student@UBUNTU:~$ rospack depends1 beginner_tutorials2
[rospack] Error: no such package beginner_tutorials2
student@UBUNTU:~$ roscd beginner_tutorials2
roscd: No such package/stack 'beginner_tutorials2'
student@UBUNTU:~$
```

```
student@UBUNTU:~$ catkin_create_pkg beginner_tutorials std_msgs rospy roscpp
usage: catkin_create_pkg [-h] [--meta] [-s [SYS_DEPS [SYS_DEPS ...]]] [-b [BOOST_COMPS [BOOST_COMPS ...]]] [-V PKG_VERSION] [-D DESCRIPTION]
                         [-l LICENSE] [-a AUTHOR] [-m MAINTAINER] [--rosdistro ROSDISTRO]
                         name [dependencies [dependencies ...]]
catkin_create_pkg: error: File exists: /home/student/beginner_tutorials/package.xml
student@UBUNTU:~$ catkin_create_pkg beginner_tutorials2 std_msgs rospy roscpp
Created file beginner_tutorials2/package.xml
Created file beginner_tutorials2/CMakeLists.txt
Created folder beginner_tutorials2/include/beginner_tutorials2
Created folder beginner_tutorials2/src
Successfully created files in /home/student/beginner_tutorials2. Please adjust the values in package.xml.
student@UBUNTU:~$ cd catkin_ws/
student@UBUNTU:~/catkin_ws$ catking_make

Command 'catking_make' not found, did you mean:

  command 'catkin_make' from deb catkin (0.8.0-1ubuntu2)

Try: apt install <deb name>

student@UBUNTU:~/catkin_ws$ catkin_make
Base path: /home/student/catkin_ws
Source space: /home/student/catkin_ws/src
Build space: /home/student/catkin_ws/build
Devel space: /home/student/catkin_ws/devel
Install space: /home/student/catkin_ws/install
####
#### Running command: "make cmake_check_build_system" in "/home/student/catkin_ws/build"
####
####
```

```
student@UBUNTU:~/catkin_ws/src/beginner_tutorials$ rospack depends1 rospy
genpy
roscpp
rosgraph
rosgraph_msgs
roslib
std_msgs
student@UBUNTU:~/catkin_ws/src/beginner_tutorials$ rospack depends beginner_tutorials
cpp_common
rostime
roscpp_traits
roscpp_serialization
catkin
genmsg
genpy
message_runtime
gencpp
geneus
gennodejs
genlisp
message_generation
rosbuild
rosconsole
std_msgs
rosgraph_msgs
xmlrpcpp
roscpp
rosgraph
ros_environment
rospack
roslib
rospy
```

```
student@UBUNTU:~$ roscd beginner_tutorials/
student@UBUNTU:~/catkin_ws/src/beginner_tutorials$ cat package.xml
<?xml version="1.0"?>
<package format="2">
  <name>beginner_tutorials</name>
  <version>0.0.0</version>
  <description>The beginner_tutorials package</description>

  <!-- One maintainer tag required, multiple allowed, one person per tag -->
  <!-- Example:  -->
  <!-- <maintainer email="jane.doe@example.com">Jane Doe</maintainer> -->
  <maintainer email="student@todo.todo">student</maintainer>


  <!-- One license tag required, multiple allowed, one license per tag -->
  <!-- Commonly used license strings: -->
  <!--   BSD, MIT, Boost Software License, GPLv2, GPLv3, LGPLv2.1, LGPLv3 -->
  <license>TODO</license>


  <!-- Url tags are optional, but multiple are allowed, one per tag -->
  <!-- Optional attribute type can be: website, bugtracker, or repository -->
  <!-- Example: -->
  <!-- <url type="website">http://wiki.ros.org/beginner_tutorials</url> -->


  <!-- Author tags are optional, multiple are allowed, one per tag -->
  <!-- Authors do not have to be maintainers, but could be -->
  <!-- Example: -->
  <!-- <author email="jane.doe@example.com">Jane Doe</author> -->


  <!-- The *depend tags are used to specify dependencies -->
  <!-- Dependencies can be catkin packages or system dependencies -->
  <!-- Examples: -->
```

Figure 6. Creating a ROS Package (Tutorial 3)

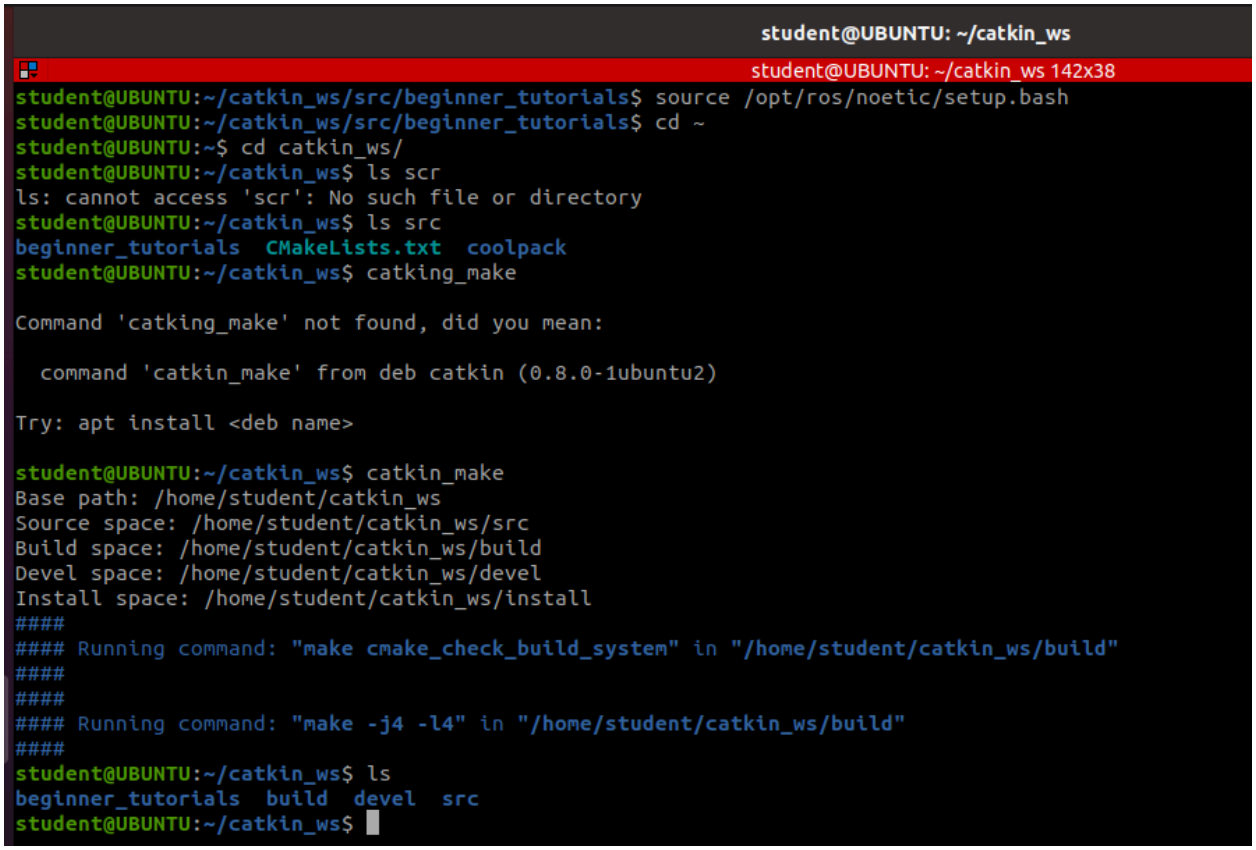
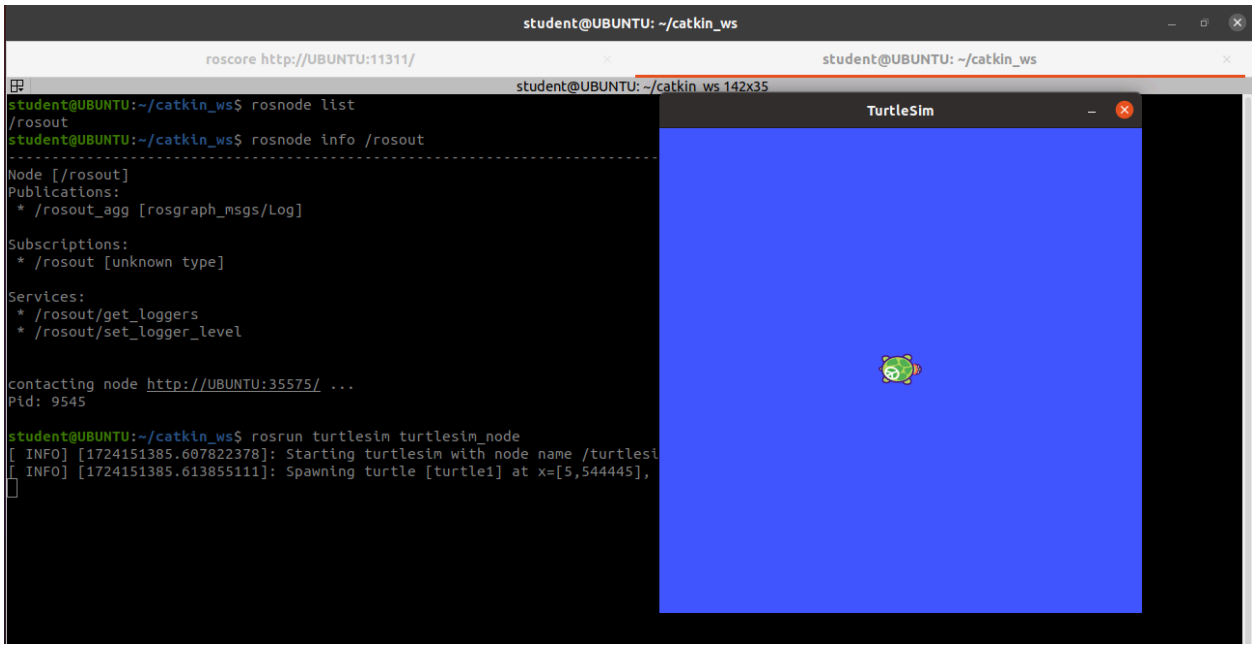


Figure 7. Building a ROS Package (Tutorial 4)



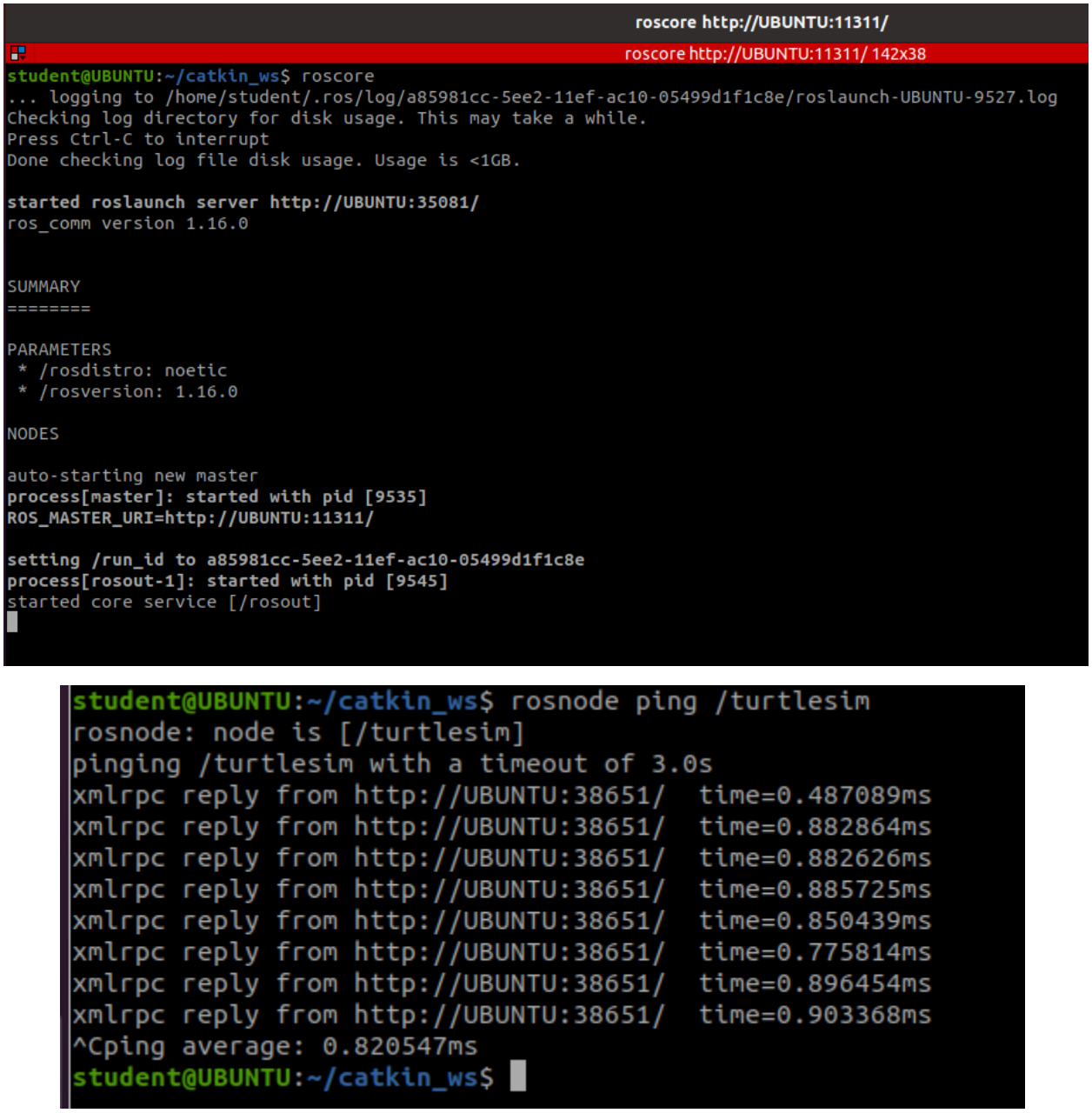
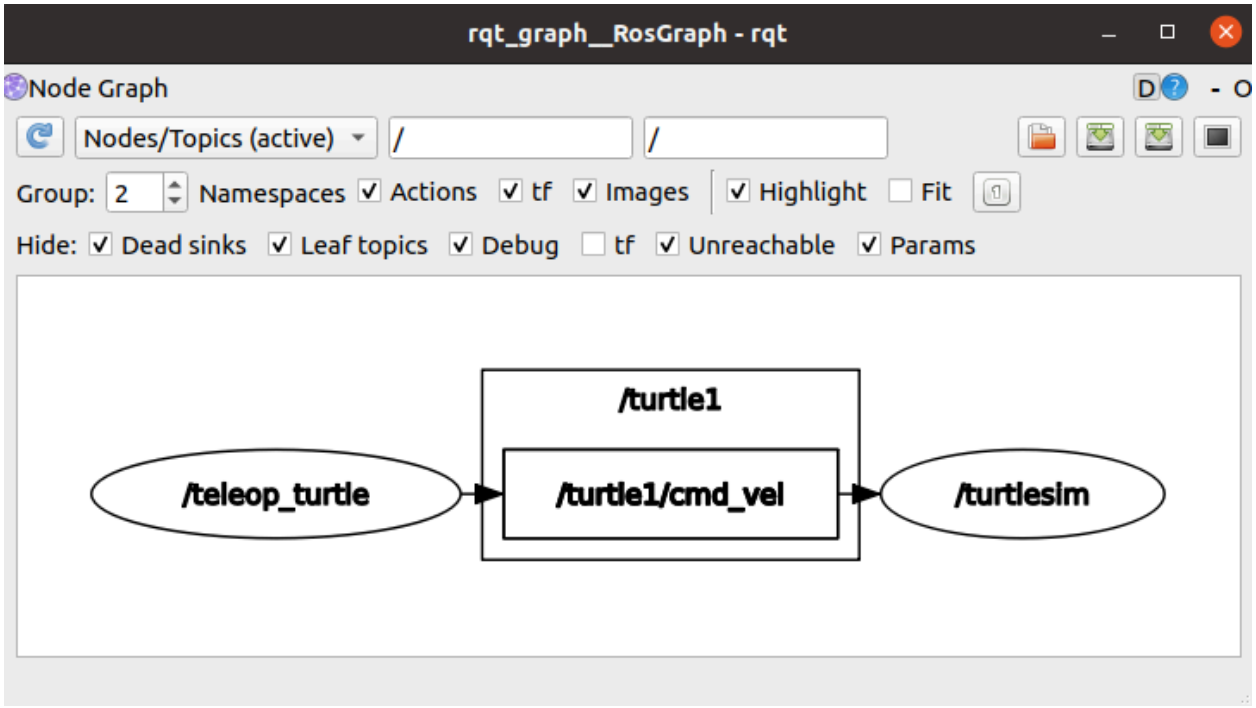
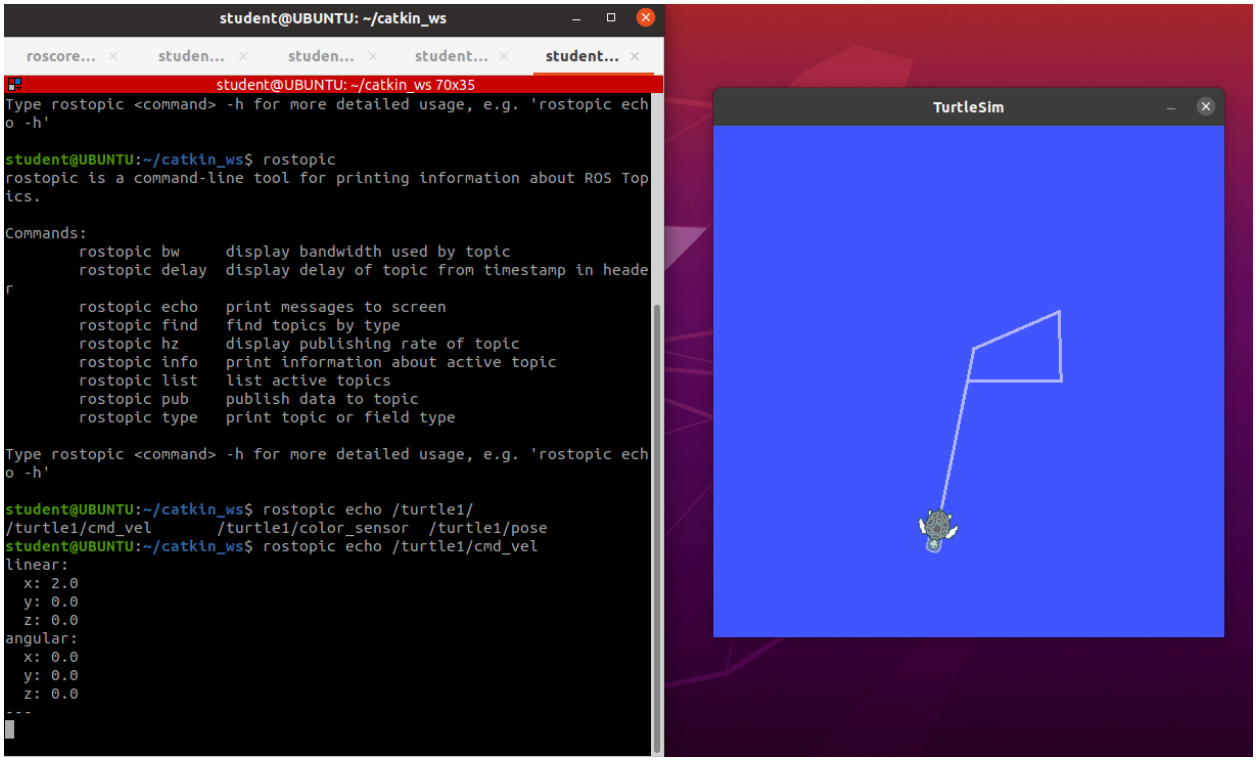
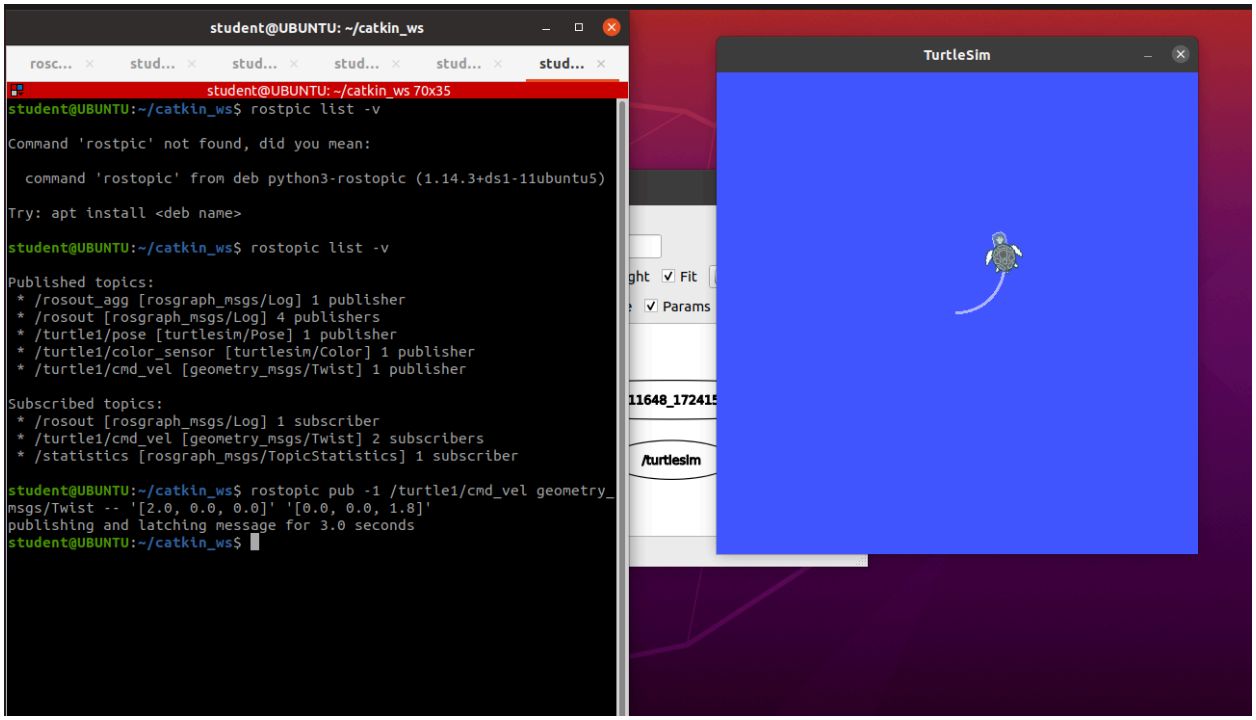


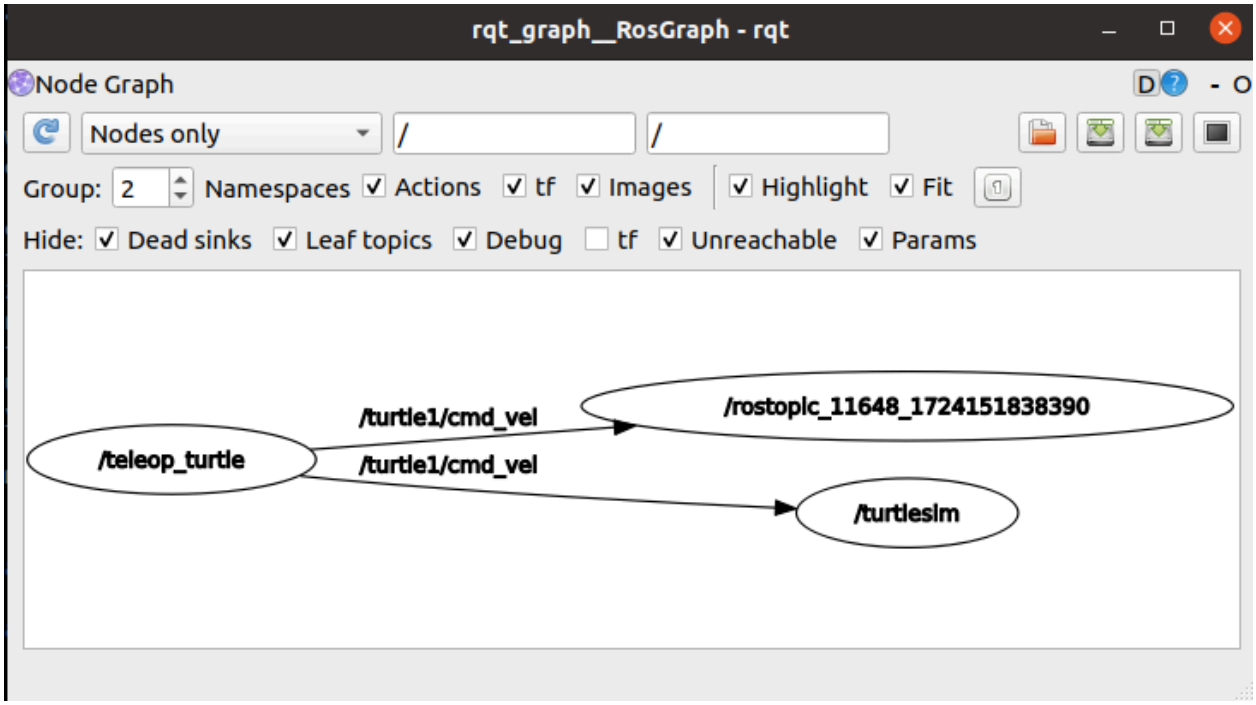
Figure 8. Understanding ROS Nodes (Tutorial 5)


rqt_graph__RosGraph - rqt
Node Graph
Nodes/Topics (active)
/
/
Group: 2
Namespaces
Actions
tf
Images
Highlight
Fit
Hide:
Dead sinks
Leaf topics
Debug
tf
Unreachable
Params
/turtle1
/teleop_turtle
/turtle1/cmd_vel
/turtlesim


student@UBUNTU: ~/catkin_ws
roscore...
studen...
studen...
student...
student...
student@UBUNTU: ~/catkin_ws 70x35
Type rostopic <command> -h for more detailed usage, e.g. 'rostopic ech
o -h'
student@UBUNTU:~/catkin_ws$ rostopic
rostopic is a command-line tool for printing information about ROS Top
ics.
Commands:
rostopic bw display bandwidth used by topic
rostopic delay display delay of topic from timestamp in heade
r
rostopic echo print messages to screen
rostopic find find topics by type
rostopic hz display publishing rate of topic
rostopic info print information about active topic
rostopic list list active topics
rostopic pub publish data to topic
rostopic type print topic or field type
Type rostopic <command> -h for more detailed usage, e.g. 'rostopic ech
o -h'
student@UBUNTU:~/catkin_ws$ rostopic echo /turtle1/
/turtle1/cmd_vel /turtle1/color_sensor /turtle1/pose
student@UBUNTU:~/catkin_ws$ rostopic echo /turtle1/cmd_vel
linear:
x: 2.0
y: 0.0
z: 0.0
angular:
x: 0.0
y: 0.0
z: 0.0

TurtleSim


student@UBUNTU: ~/catkin_ws
student@UBUNTU: ~/catkin_ws 70x35
student@UBUNTU:~/catkin_ws$ rostpic list -v
Command 'rostpic' not found, did you mean:
command 'rostopic' from deb python3-rostopic (1.14.3+ds1-11ubuntu5)
Try: apt install <deb name>
student@UBUNTU:~/catkin_ws$ rostopic list -v
Published topics:
* /rosout_agg [rosgraph_msgs/Log] 1 publisher
* /rosout [rosgraph_msgs/Log] 4 publishers
* /turtle1/pose [turtlesim/Pose] 1 publisher
* /turtle1/color_sensor [turtlesim/Color] 1 publisher
* /turtle1/cmd_vel [geometry_msgs/Twist] 1 publisher
Subscribed topics:
* /rosout [rosgraph_msgs/Log] 1 subscriber
* /turtle1/cmd_vel [geometry_msgs/Twist] 2 subscribers
* /statistics [rosgraph_msgs/TopicStatistics] 1 subscriber
student@UBUNTU:~/catkin_ws$ rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'
publishing and latching message for 3.0 seconds
student@UBUNTU:~/catkin_ws$
TurtleSim
Fit
Params
/turtlesim


rqt_graph__RosGraph - rqt
Node Graph
Nodes only
/
/
Group: 2
Namespaces
Actions
tf
Images
Highlight
Fit
Hide:
Dead sinks
Leaf topics
Debug
tf
Unreachable
Params
/turtle1/cmd_vel
/rostopic_11648_1724151838390
/teleop_turtle
/turtle1/cmd_vel
/turtlesim

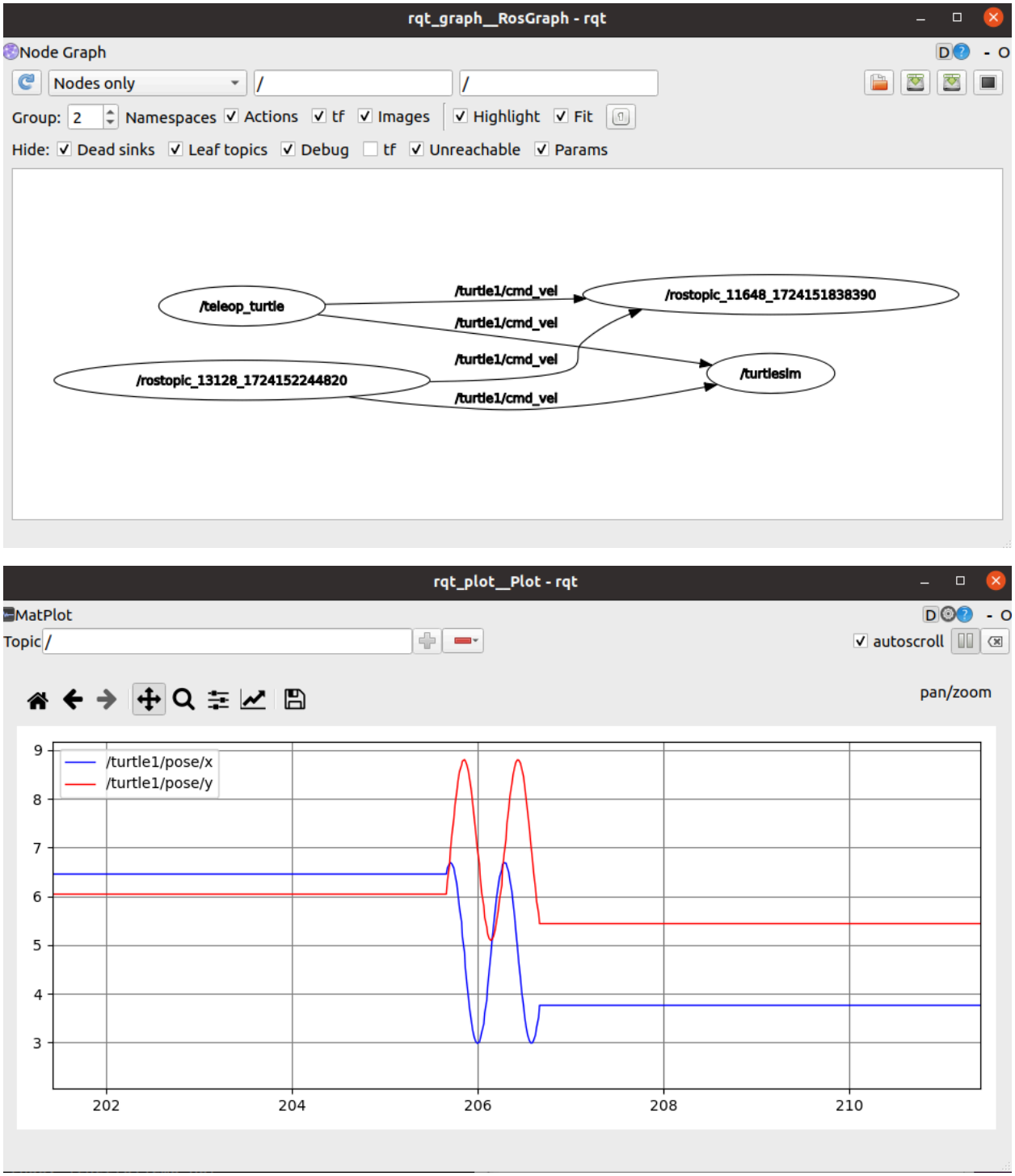
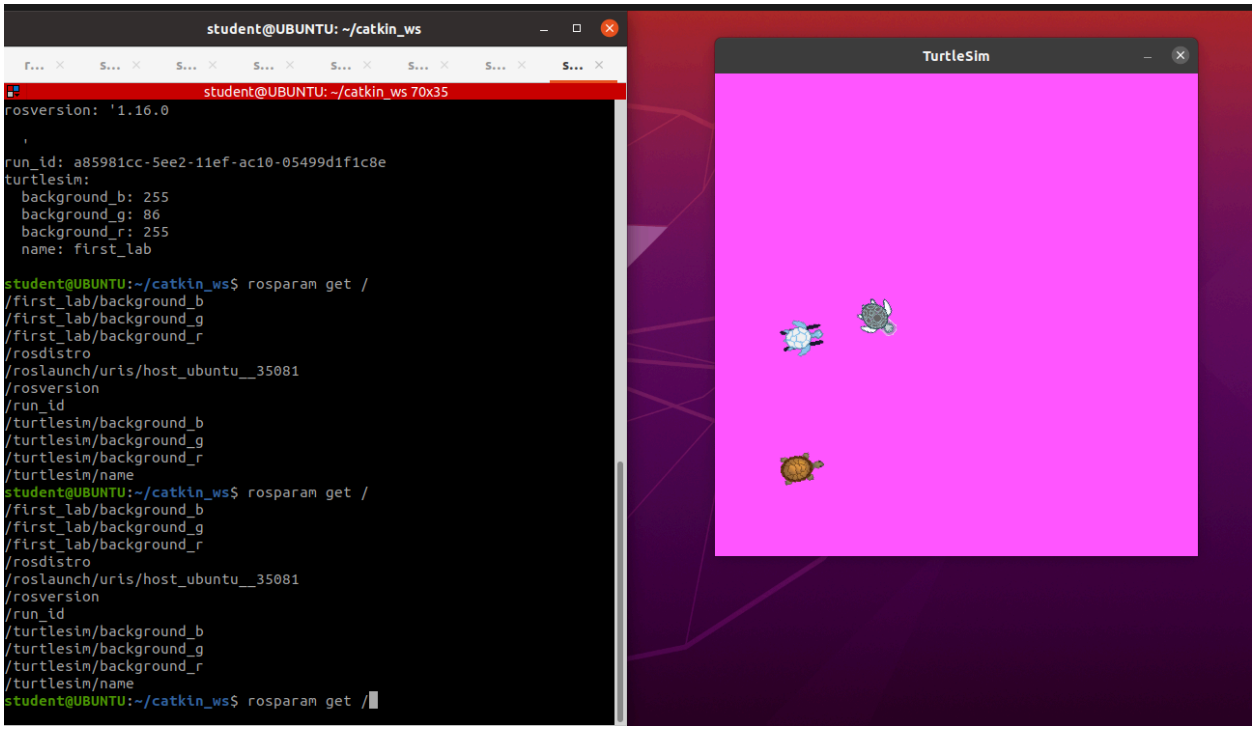
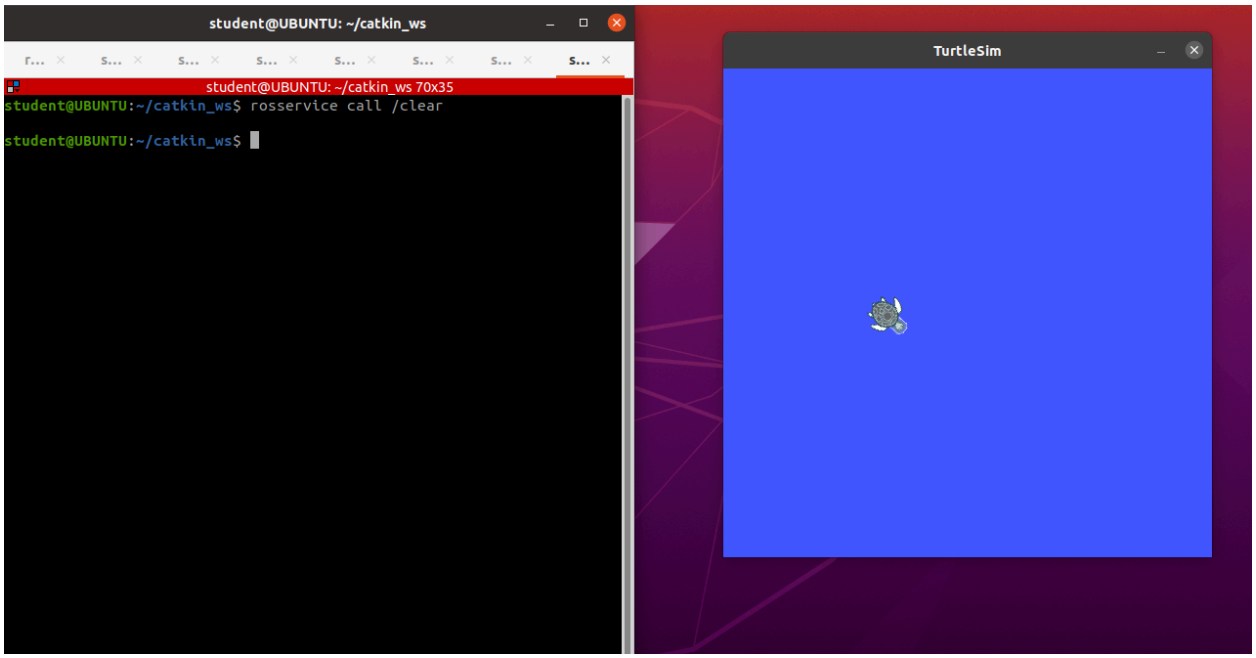


Figure 9. Understanding ROS Topics (Tutorial 6)


student@UBUNTU: ~/catkin_ws
student@UBUNTU: ~/catkin_ws 70x35
rosversion: '1.16.0
'
run_id: a85981cc-5ee2-11ef-ac10-05499d1f1c8e
turtlesim:
background_b: 255
background_g: 86
background_r: 255
name: first_lab
student@UBUNTU:~/catkin_ws$ rosparam get /
/first_lab/background_b
/first_lab/background_g
/first_lab/background_r
/rosdistro
/roslaunch/uris/host_ubuntu__35081
/rosversion
/run_id
/turtlesim/background_b
/turtlesim/background_g
/turtlesim/background_r
/turtlesim/name
student@UBUNTU:~/catkin_ws$ rosparam get /
/first_lab/background_b
/first_lab/background_g
/first_lab/background_r
/rosdistro
/roslaunch/uris/host_ubuntu__35081
/rosversion
/run_id
/turtlesim/background_b
/turtlesim/background_g
/turtlesim/background_r
/turtlesim/name
student@UBUNTU:~/catkin_ws$ rosparam get /
TurtleSim


student@UBUNTU: ~/catkin_ws
student@UBUNTU: ~/catkin_ws 70x35
student@UBUNTU:~/catkin_ws$ rosservice call /clear
student@UBUNTU:~/catkin_ws$
TurtleSim

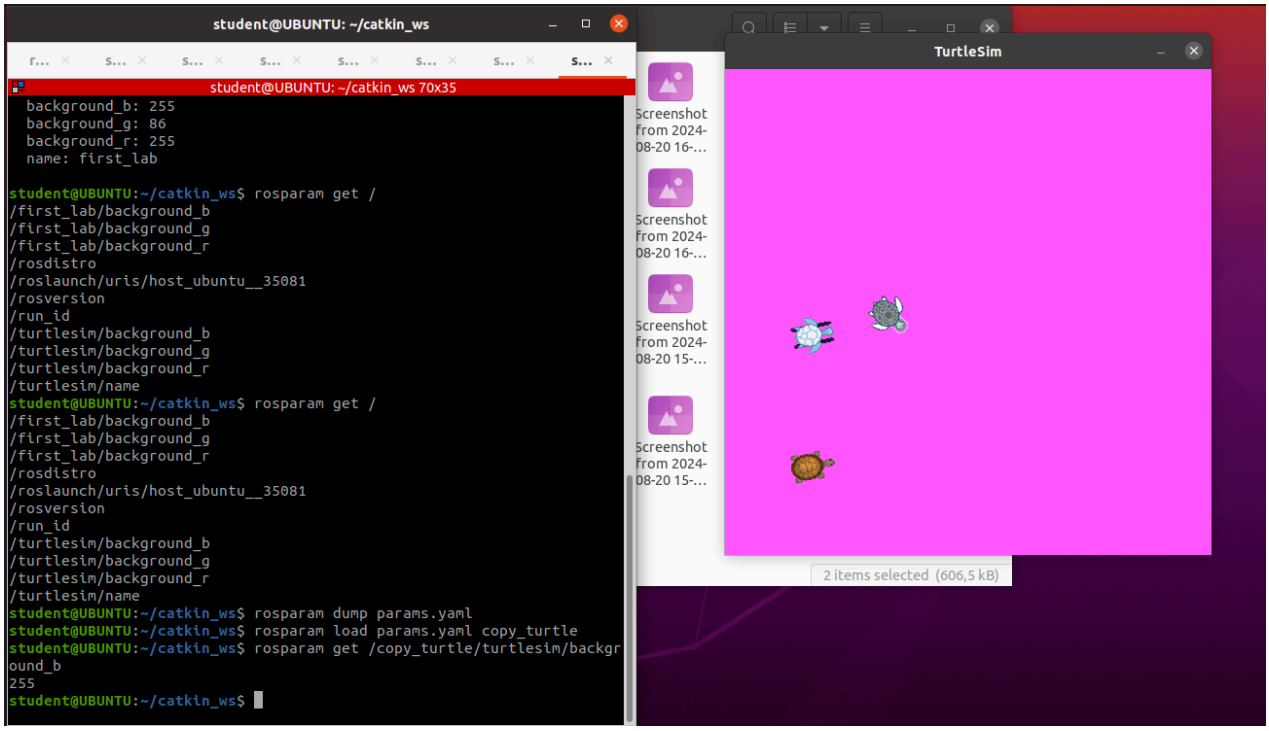
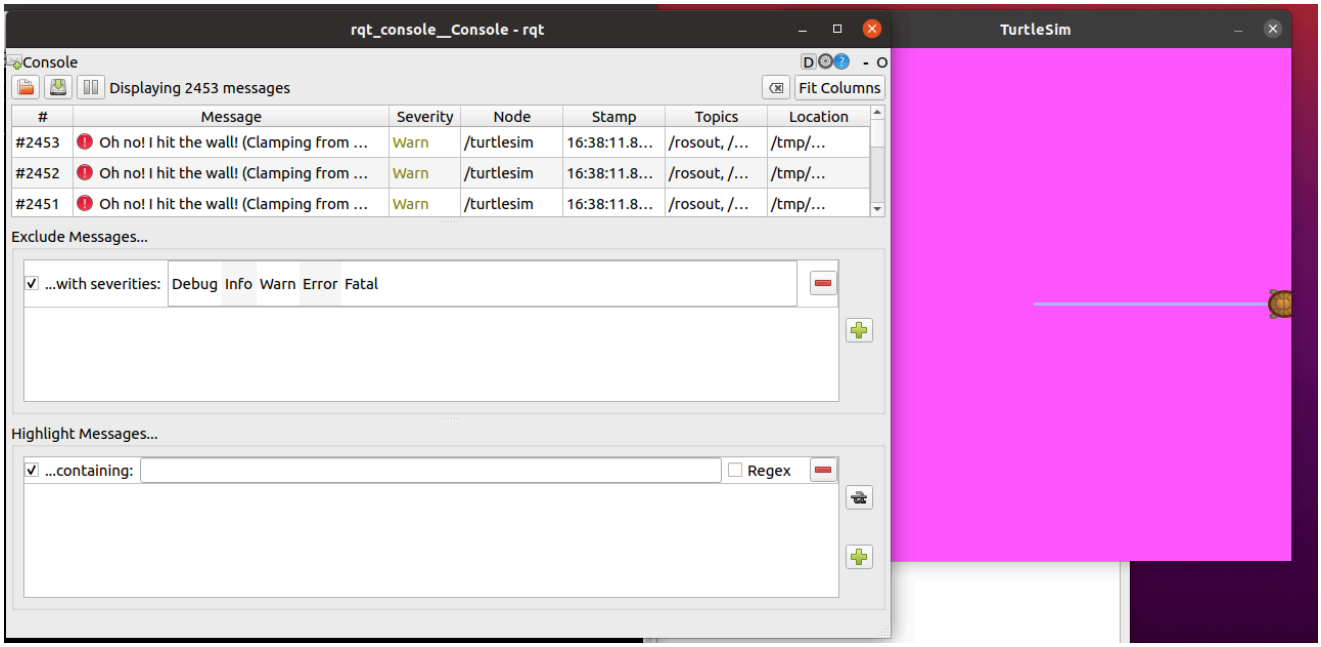
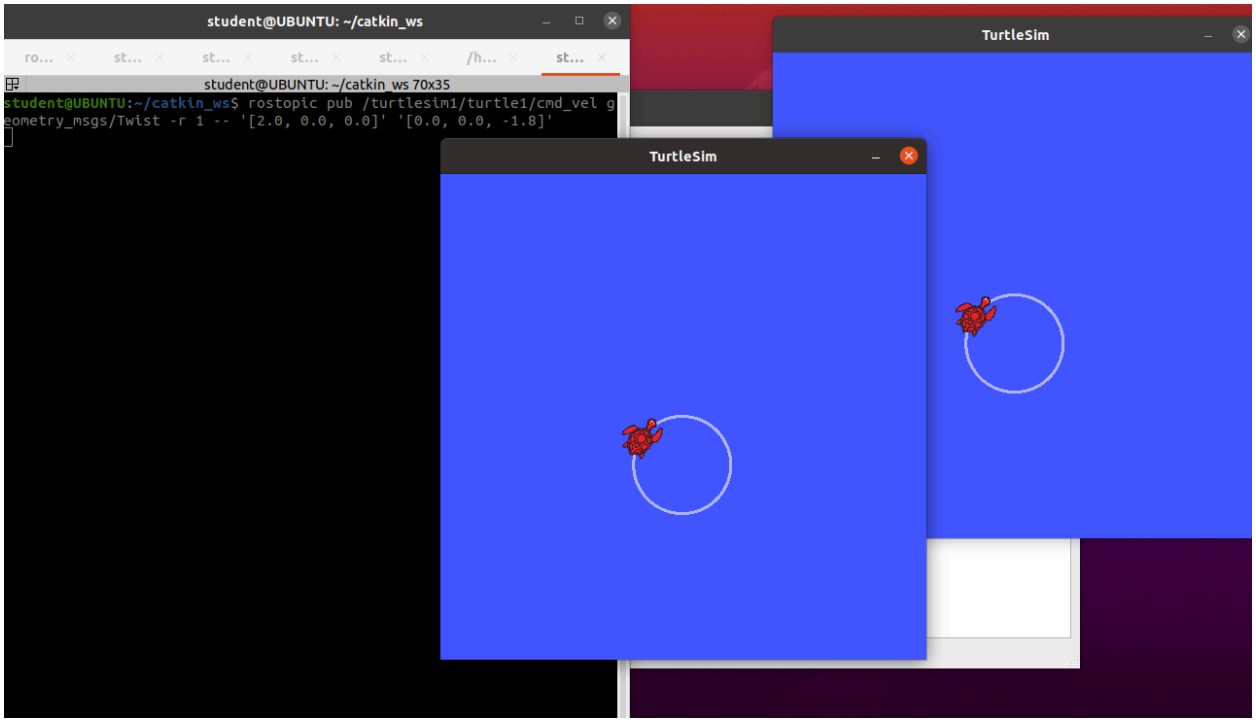
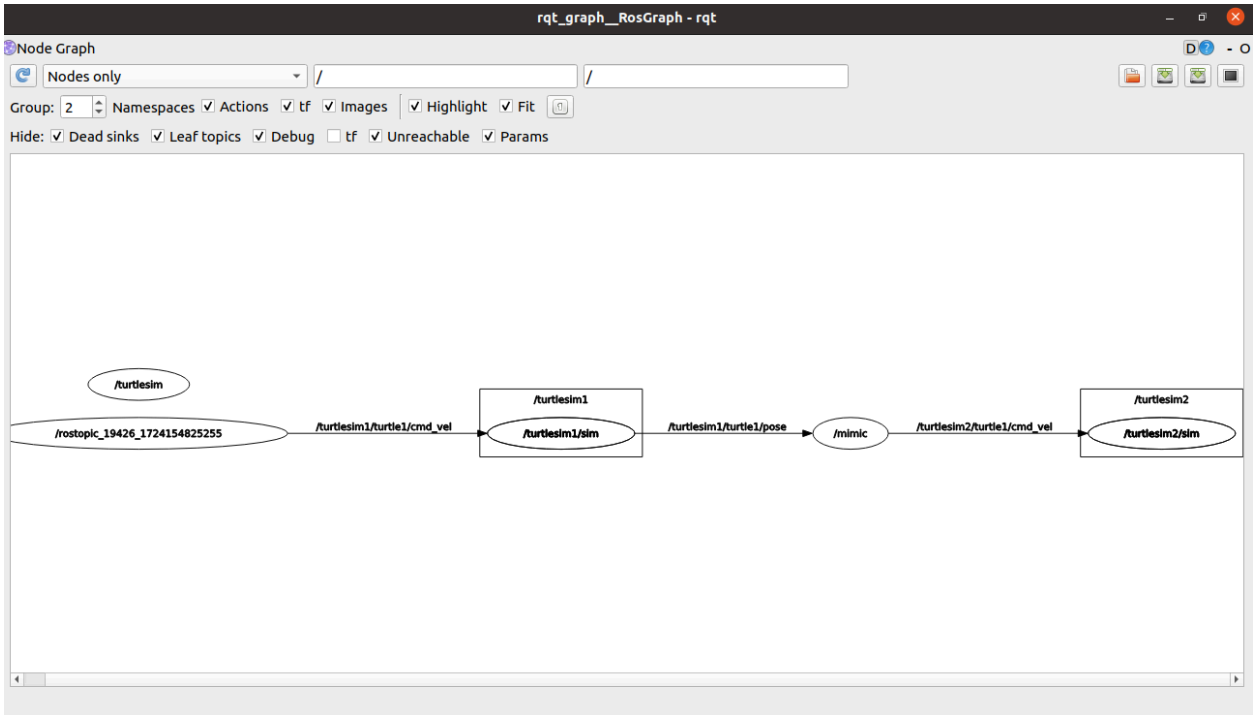


Figure 10. Understanding ROS Services and Parameters (Tutorial 7)




student@UBUNTU: ~/catkin_ws
student@UBUNTU: ~/catkin_ws 70x35
student@UBUNTU:~/catkin_ws$ rostopic pub /turtlesim1/turtle1/cmd_vel g
eometry_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, -1.8]'
TurtleSim
TurtleSim


rqt_graph__RosGraph - rqt
Node Graph
Nodes only
Group: 2
Namespaces
Actions
tf
Images
Highlight
Fit
Hide:
Dead sinks
Leaf topics
Debug
tf
Unreachable
Params
/turtlesim
/rostopic_19426_1724154825255
/turtlesim1/turtle1/cmd_vel
/turtlesim1
/turtlesim1/sim
/turtlesim1/turtle1/pose
/mimic
/turtlesim2/turtle1/cmd_vel
/turtlesim2
/turtlesim2/sim

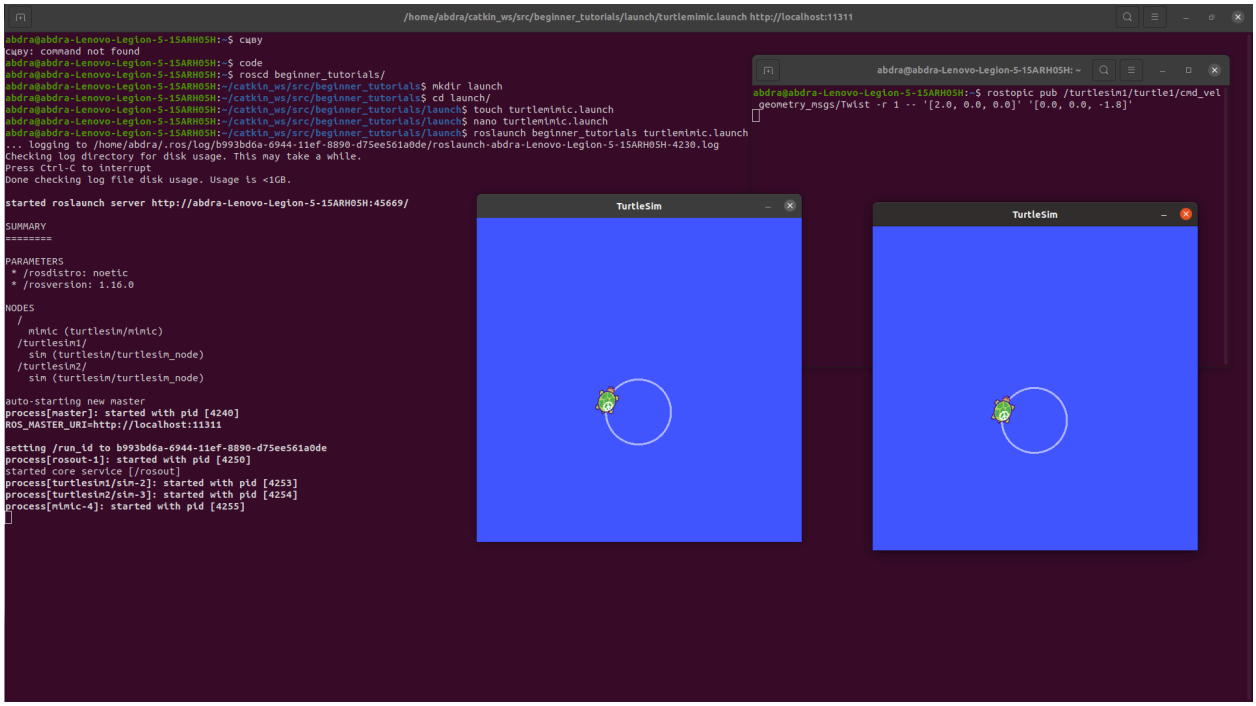


Figure 11. Using rqt_console and roslaunch (Tutorial 8)

```
student@UBUNTU:~/catkin_ws/beginner_tutorials/srv$ rossrv show AddTw
nts.srv
[rospy_tutorials/AddTwoInts]:
int64 a
int64 b
---
int64 sum

student@UBUNTU:~/catkin_ws/beginner_tutorials/srv$
```

```
student@UBUNTU: ~/catkin_ws/beginner_tutorials 70x35
eginner_tutorials/msg']]
student@UBUNTU:~/catkin_ws$ ls
beginner_tutorials  build  devel  params.yaml  src
student@UBUNTU:~/catkin_ws$ roscd beginner_tutorials/
student@UBUNTU:~/catkin_ws/src/beginner_tutorials$ ls
CMakeLists.txt  include  launch  package.xml  src
student@UBUNTU:~/catkin_ws/src/beginner_tutorials$ cd ..
student@UBUNTU:~/catkin_ws/src$ ls
beginner_tutorials  CMakeLists.txt  coolpack
student@UBUNTU:~/catkin_ws/src$ cd beginner_tutorials/
student@UBUNTU:~/catkin_ws/src/beginner_tutorials$ ls
CMakeLists.txt  include  launch  package.xml  src
student@UBUNTU:~/catkin_ws/src/beginner_tutorials$ mkdir msg
student@UBUNTU:~/catkin_ws/src/beginner_tutorials$ cd msg
student@UBUNTU:~/catkin_ws/src/beginner_tutorials/msg$ echo "int64 num
" > Num.msg
student@UBUNTU:~/catkin_ws/src/beginner_tutorials/msg$ cat Num.msg
int64 num
student@UBUNTU:~/catkin_ws/src/beginner_tutorials/msg$ ls
Num.msg
student@UBUNTU:~/catkin_ws/src/beginner_tutorials/msg$ cd ..
student@UBUNTU:~/catkin_ws/src/beginner_tutorials$ cd ..
student@UBUNTU:~/catkin_ws/src$ cd ..
student@UBUNTU:~/catkin_ws$ ls
beginner_tutorials  build  devel  params.yaml  src
student@UBUNTU:~/catkin_ws$ cd beg
bash: cd: beg: No such file or directory
student@UBUNTU:~/catkin_ws$ cd beginner_tutorials/
student@UBUNTU:~/catkin_ws/beginner_tutorials$ ls
CMakeLists.txt  include  msg  package.xml  src
student@UBUNTU:~/catkin_ws/beginner_tutorials$ rosmsg show beginner_tu
torials/Num
int64 num

student@UBUNTU:~/catkin_ws/beginner_tutorials$
```

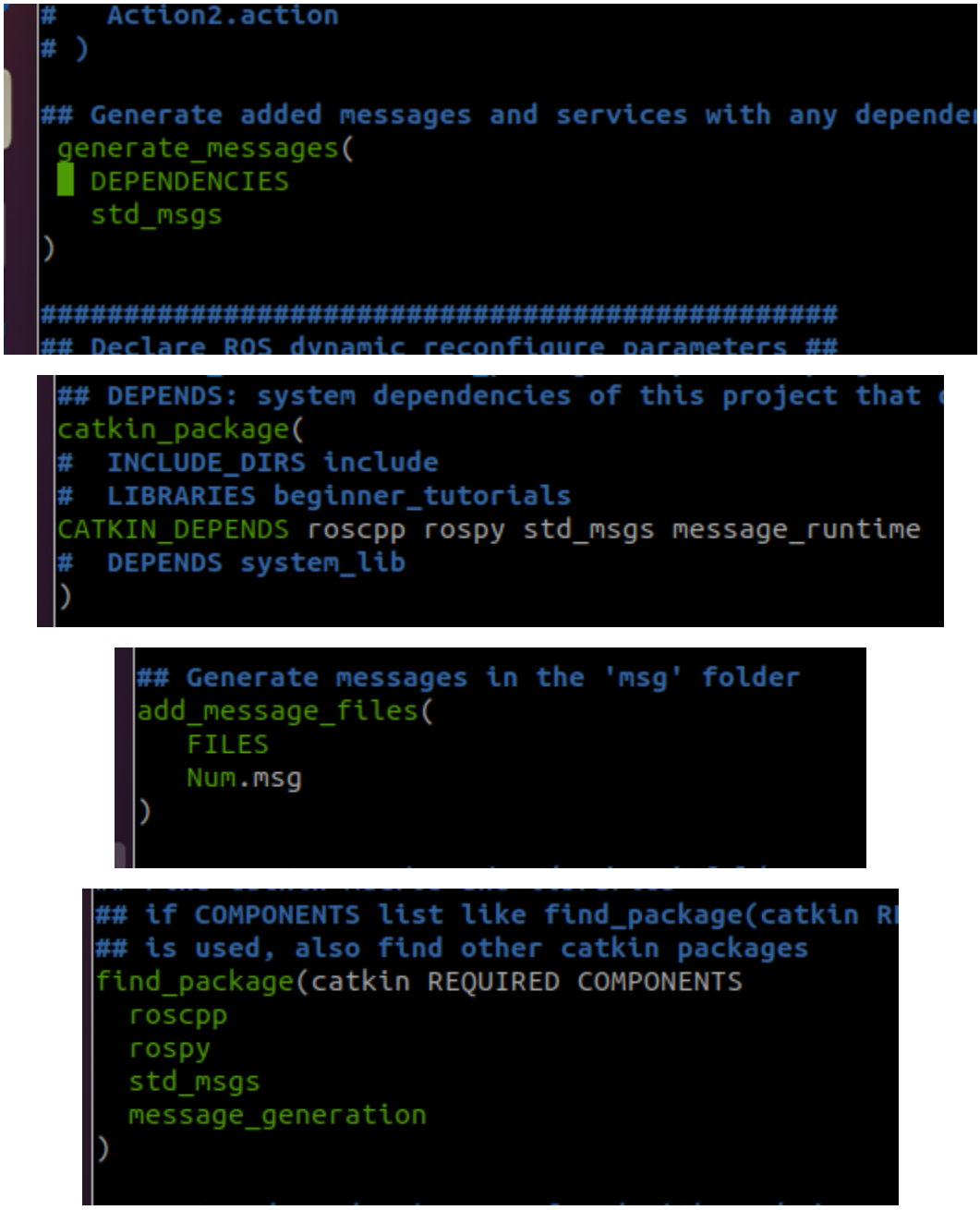


Figure 12. Creating a ROS msg and srv (Tutorial 10)

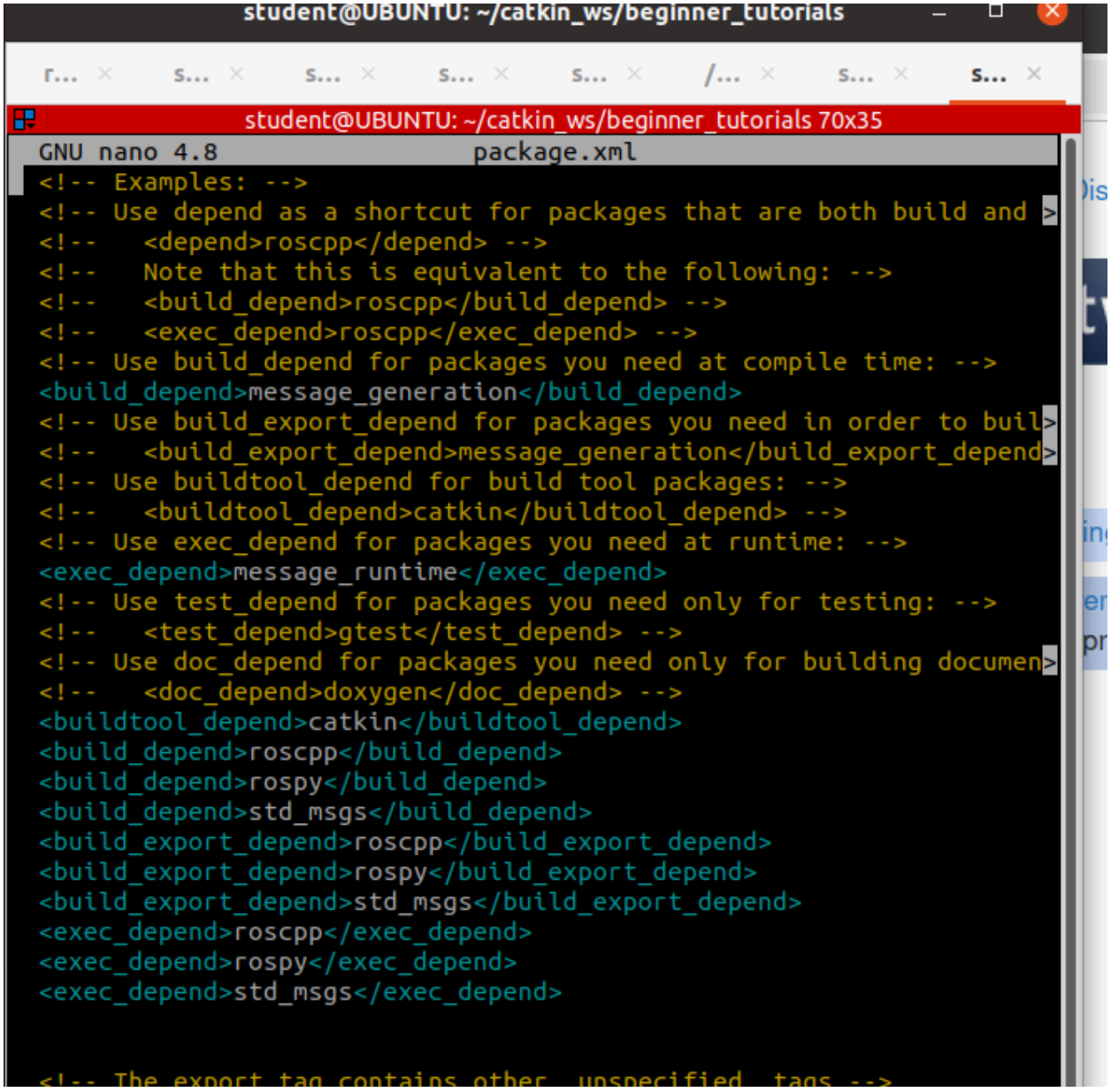


Figure 13. Writing a Simple Publisher and Subscriber (C++) (Tutorial 11)

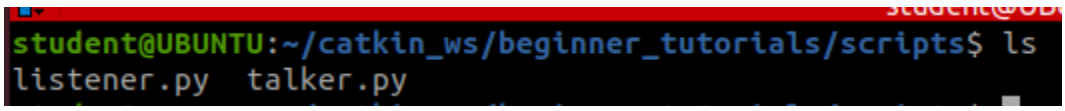


Figure 14. Writing a Simple Publisher and Subscriber (Python) (Tutorial 12)

```
-- BUILD_SHARED_LIBS is on
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- ~~  traversing 1 packages in topological order:
-- ~~  - beginner_tutorials
-- ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
-- +++ processing catkin package: 'beginner_tutorials'
-- ==> add_subdirectory(beginner_tutorials)
-- Using these message generators: gencpp;geneus;genlisp;gennodejs;genpy
-- beginner_tutorials: 1 messages, 0 services
-- Configuring done
-- Generating done
-- Build files have been written to: /home/abdra/catkin_ws/build
####
#### Running command: "make -j12 -l12" in "/home/abdra/catkin_ws/build"
####
Scanning dependencies of target _beginner_tutorials_generate_messages_check_deps_Num
Scanning dependencies of target std_msgs_generate_messages_cpp
Scanning dependencies of target std_msgs_generate_messages_nodejs
Scanning dependencies of target std_msgs_generate_messages_eus
Scanning dependencies of target std_msgs_generate_messages_py
Scanning dependencies of target std_msgs_generate_messages_lisp
[  0%] Built target std_msgs_generate_messages_eus
[  0%] Built target std_msgs_generate_messages_nodejs
[  0%] Built target std_msgs_generate_messages_lisp
[  0%] Built target std_msgs_generate_messages_py
[  0%] Built target std_msgs_generate_messages_cpp
[  0%] Built target _beginner_tutorials_generate_messages_check_deps_Num
Scanning dependencies of target beginner_tutorials_generate_messages_cpp
Scanning dependencies of target beginner_tutorials_generate_messages_lisp
Scanning dependencies of target beginner_tutorials_generate_messages_py
Scanning dependencies of target beginner_tutorials_generate_messages_eus
Scanning dependencies of target beginner_tutorials_generate_messages_nodejs
[ 27%] Generating Lisp code from beginner_tutorials/Num.msg
[ 27%] Generating C++ code from beginner_tutorials/Num.msg
[ 27%] Generating Python from MSG beginner_tutorials/Num
[ 36%] Generating Javascript code from beginner_tutorials/Num.msg
[ 45%] Generating EusLisp manifest code for beginner_tutorials
[ 54%] Generating EusLisp code from beginner_tutorials/Num.msg
[ 54%] Built target beginner_tutorials_generate_messages_lisp
[ 54%] Built target beginner_tutorials_generate_messages_nodejs
[ 54%] Built target beginner_tutorials_generate_messages_cpp
Scanning dependencies of target talker
Scanning dependencies of target listener
[ 63%] Building CXX object beginner_tutorials/CMakeFiles/listener.dir/src/listener.cpp.o
[ 72%] Building CXX object beginner_tutorials/CMakeFiles/talker.dir/src/talker.cpp.o
[ 81%] Generating Python msg __init__.py for beginner_tutorials
[ 81%] Built target beginner_tutorials_generate_messages_eus
[ 81%] Built target beginner_tutorials_generate_messages_py
Scanning dependencies of target beginner_tutorials_generate_messages
[ 81%] Built target beginner_tutorials_generate_messages
[ 90%] Linking CXX executable /home/abdra/catkin_ws/devel/lib/beginner_tutorials/listener
[100%] Linking CXX executable /home/abdra/catkin_ws/devel/lib/beginner_tutorials/talker
[100%] Built target listener
[100%] Built target talker
abdra@abdra-Lenovo-Legion-5-15ARH05H:~/catkin_ws$
```

```
abdra@abdra-Lenovo-Legion-5-15ARH05H:~/catkin_ws$ rosrun beginner_tutorials listener
[ INFO] [1724749673.772840889]: I heard: [hello world 3]
[ INFO] [1724749673.872812628]: I heard: [hello world 4]
[ INFO] [1724749673.972785973]: I heard: [hello world 5]
[ INFO] [1724749674.073133815]: I heard: [hello world 6]
[ INFO] [1724749674.172642059]: I heard: [hello world 7]
[ INFO] [1724749674.272461807]: I heard: [hello world 8]
[ INFO] [1724749674.372821171]: I heard: [hello world 9]
[ INFO] [1724749674.472756930]: I heard: [hello world 10]
[ INFO] [1724749674.572568227]: I heard: [hello world 11]
[ INFO] [1724749674.672555951]: I heard: [hello world 12]
[ INFO] [1724749674.772737351]: I heard: [hello world 13]
[ INFO] [1724749674.872756295]: I heard: [hello world 14]
[ INFO] [1724749674.972540423]: I heard: [hello world 15]
[ INFO] [1724749675.072641496]: I heard: [hello world 16]
[ INFO] [1724749675.172576477]: I heard: [hello world 17]
[ INFO] [1724749675.272785038]: I heard: [hello world 18]
[ INFO] [1724749675.372776383]: I heard: [hello world 19]
[ INFO] [1724749675.472829750]: I heard: [hello world 20]
[ INFO] [1724749675.572852455]: I heard: [hello world 21]
[ INFO] [1724749675.672637342]: I heard: [hello world 22]
[ INFO] [1724749675.772833400]: I heard: [hello world 23]
[ INFO] [1724749675.872821113]: I heard: [hello world 24]
[ INFO] [1724749675.972829361]: I heard: [hello world 25]
[ INFO] [1724749676.072825169]: I heard: [hello world 26]
[ INFO] [1724749676.172533357]: I heard: [hello world 27]
[ INFO] [1724749676.272623033]: I heard: [hello world 28]
[ INFO] [1724749676.372610178]: I heard: [hello world 29]
```

```
abdra@abdra-Lenovo-Legion-5-15ARH05H:~/catkin_ws$ rosrun beginner_tutorials talker
[ INFO] [1724749673.472046197]: hello world 0
[ INFO] [1724749673.572640525]: hello world 1
[ INFO] [1724749673.672140328]: hello world 2
[ INFO] [1724749673.772229125]: hello world 3
[ INFO] [1724749673.872338317]: hello world 4
[ INFO] [1724749673.972320812]: hello world 5
[ INFO] [1724749674.072528337]: hello world 6
[ INFO] [1724749674.172122140]: hello world 7
[ INFO] [1724749674.272078015]: hello world 8
[ INFO] [1724749674.372320738]: hello world 9
[ INFO] [1724749674.472310347]: hello world 10
[ INFO] [1724749674.572131493]: hello world 11
[ INFO] [1724749674.672124524]: hello world 12
[ INFO] [1724749674.772324014]: hello world 13
[ INFO] [1724749674.872326894]: hello world 14
[ INFO] [1724749674.972129950]: hello world 15
[ INFO] [1724749675.072162925]: hello world 16
[ INFO] [1724749675.172134365]: hello world 17
[ INFO] [1724749675.272356126]: hello world 18
[ INFO] [1724749675.372337763]: hello world 19
[ INFO] [1724749675.472342518]: hello world 20
[ INFO] [1724749675.572341476]: hello world 21
[ INFO] [1724749675.672146129]: hello world 22
[ INFO] [1724749675.772353711]: hello world 23
[ INFO] [1724749675.872343799]: hello world 24
[ INFO] [1724749675.972346879]: hello world 25
[ INFO] [1724749676.072352954]: hello world 26
[ INFO] [1724749676.172082445]: hello world 27
[ INFO] [1724749676.272139014]: hello world 28
[ INFO] [1724749676.372170022]: hello world 29
^C[ INFO] [1724749676.472151439]: hello world 30
```

Figure 15. Examining the Simple Publisher and Subscriber (Tutorial 13)

```
abdra@abdra-Lenovo-Legion-5-15ARH05H:~/catkin_ws/src/beginner_tutorials$ cd srv
abdra@abdra-Lenovo-Legion-5-15ARH05H:~/catkin_ws/src/beginner_tutorials/srv$ ls
AddTwoInts.srv
abdra@abdra-Lenovo-Legion-5-15ARH05H:~/catkin_ws/src/beginner_tutorials/srv$ cd ..
abdra@abdra-Lenovo-Legion-5-15ARH05H:~/catkin_ws/src/beginner_tutorials$ cd src
abdra@abdra-Lenovo-Legion-5-15ARH05H:~/catkin_ws/src/beginner_tutorials/src$ ls
add_two_ints_client.cpp  add_two_ints_server.cpp  listener.cpp  talker.cpp
abdra@abdra-Lenovo-Legion-5-15ARH05H:~/catkin_ws/src/beginner_tutorials/src$
```

Figure 16. Writing a Simple Service and Client (C++) (Tutorial 14)

```
abdra@abdra-Lenovo-Legion-5-15ARH05H:~/catkin_ws$ rosrun beginner_tutorials add_two_ints_client 1 3
[ INFO] [1724752628.711693895]: Sum: 4
abdra@abdra-Lenovo-Legion-5-15ARH05H:~/catkin_ws$
```

```
abdra@abdra-Lenovo-Legion-5-15ARH05H:~/catkin_ws$ rosrun beginner_tutorials add_two_ints_server
[ INFO] [1724752612.493347458]: Ready to add two ints.
[ INFO] [1724752628.711302299]: request: x=1, y=3
[ INFO] [1724752628.711384361]: sending back response: [4]
```

Figure 17. Examining the Simple Service and Client (Tutorial 16)

Finally, we have implemented a C++ code that sends the ID of Abdirakhman to the listener in 1 Hz and 100 Hz.

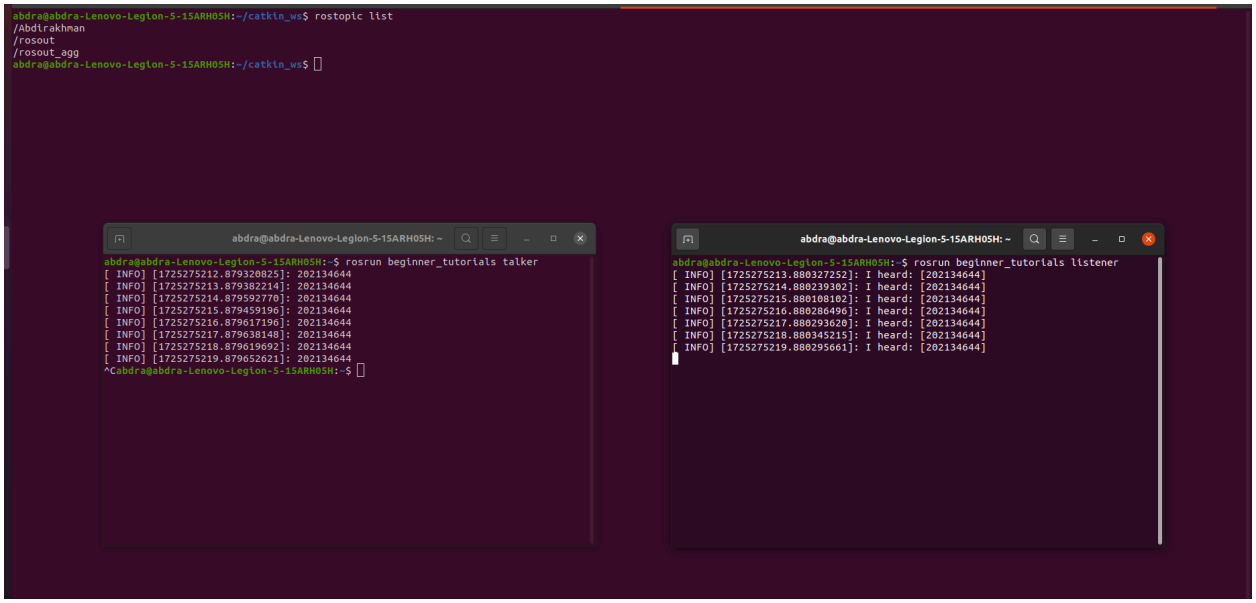


Figure 18. Sending ID in 1 Hz (Task 0)

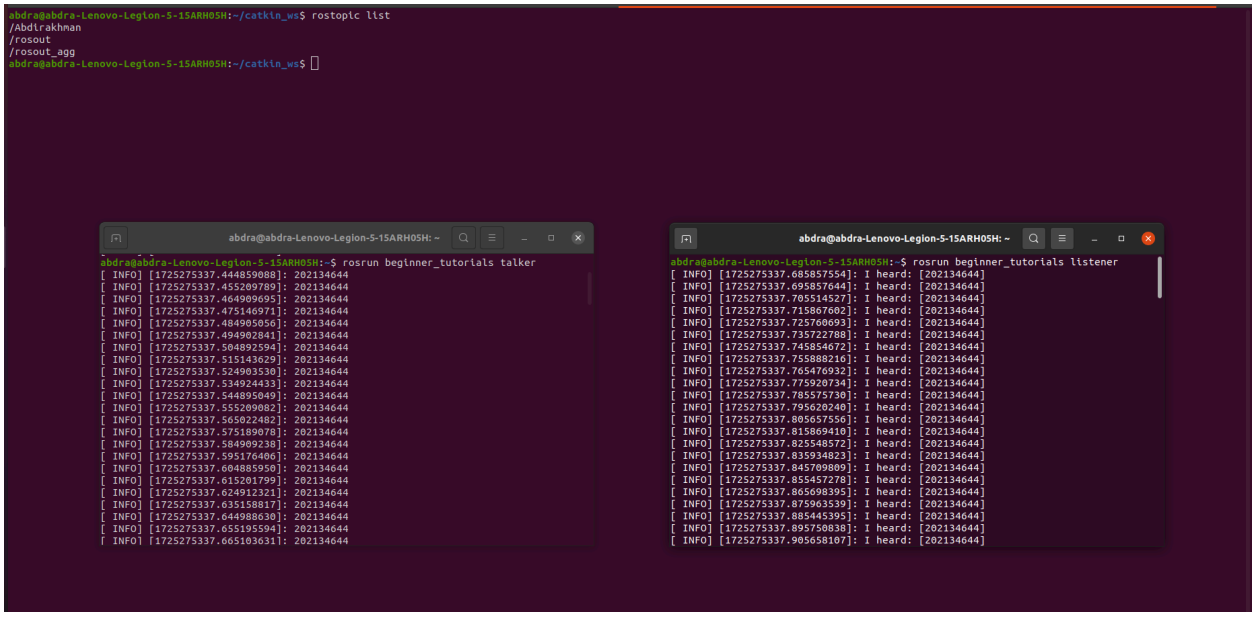


Figure 19. Sending ID in 100 Hz (Task 0)

Following the first 16 ROS beginner tutorials, we have learned several foundational skills. We start by installing and setting up the ROS environment, navigating the ROS filesystem, and creating and building ROS packages. We then delve into the core concepts of ROS, such as nodes, topics, services, and parameters, by writing simple publisher and subscriber nodes in both C++ and Python. Additionally, we create custom message types, work with ROS services and clients, and explore essential debugging tools to troubleshoot and optimize our ROS applications.

listener.cpp | README.md

C: > Users > filya > Downloads > Telegram Desktop > listener.cpp

```cpp
#include "ros/ros.h"
#include "std_msgs/String.h"

/**
 * This tutorial demonstrates simple receipt of messages over the ROS system.
 */
void chatterCallback(const std_msgs::String::ConstPtr& msg)
{
  ROS_INFO("I heard: [%s]", msg->data.c_str());
}

int main(int argc, char **argv)
{
  /**
   * The ros::init() function needs to see argc and argv so that it can perform
   * any ROS arguments and name remapping that were provided at the command line.
   * For programmatic remappings you can use a different version of init() which takes
   * remappings directly, but for most command-line programs, passing argc and argv is
   * the easiest way to do it.  The third argument to init() is the name of the node.
   *
   * You must call one of the versions of ros::init() before using any other
   * part of the ROS system.
   */
  ros::init(argc, argv, "listener");

  /**
   * NodeHandle is the main access point to communications with the ROS system.
   * The first NodeHandle constructed will fully initialize this node, and the last
   * NodeHandle destructed will close down the node.
   */
  ros::NodeHandle n;

  /**
   * The subscribe() call is how you tell ROS that you want to receive messages
   * on a given topic.  This invokes a call to the ROS
   * master node, which keeps a registry of who is publishing and who
   * is subscribing.  Messages are passed to a callback function, here
   * called chatterCallback.  subscribe() returns a Subscriber object that you
   * must hold on to until you want to unsubscribe.  When all copies of the Subscriber
   * object go out of scope, this callback will automatically be unsubscribed from
   * this topic.
   *
   * The second parameter to the subscribe() function is the size of the message
   * queue.  If messages are arriving faster than they are being processed, this
   * is the number of messages that will be buffered up before beginning to throw
   * away the oldest ones.
   */
  ros::Subscriber sub = n.subscribe("Abdirakhman", 1000, chatterCallback);

  /**
   * ros::spin() will enter a loop, pumping callbacks.  With this version, all
   * callbacks will be called from within this thread (the main one).  ros::spin()
   * will exit when Ctrl-C is pressed, or the node is shutdown by the master.
   */
  ros::spin();

  return 0;
}
```

Figure 20. Code for sending ID in specified frequency (Task 0)

4. Task 1

In this task, we had to use two cpp files – publisher and subscriber and add them to the CMake list in order to be able to execute it. Figure below shows the result of execution of both files.

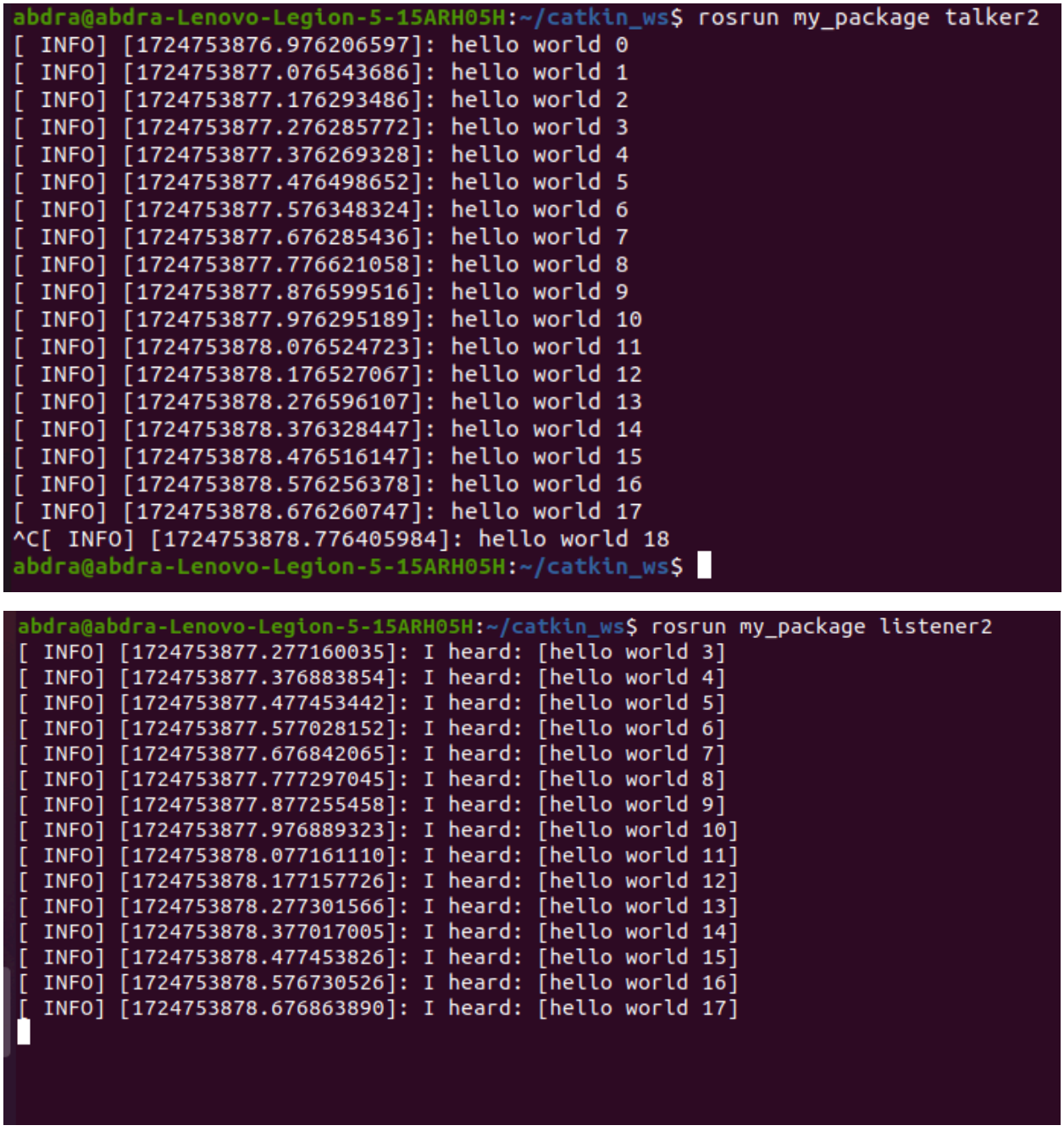


Figure 21. Running publisher and subscriber (Task 1)

5. Task 2

After the success with task 1, it was time to use the listener from the turtlesim and get the position (x, y) of the turtle in the virtual environment. To make it possible, we had to run the roscore, turtlesim node, and listener. As could be seen below, we were successfully receiving turtlesim's positions.

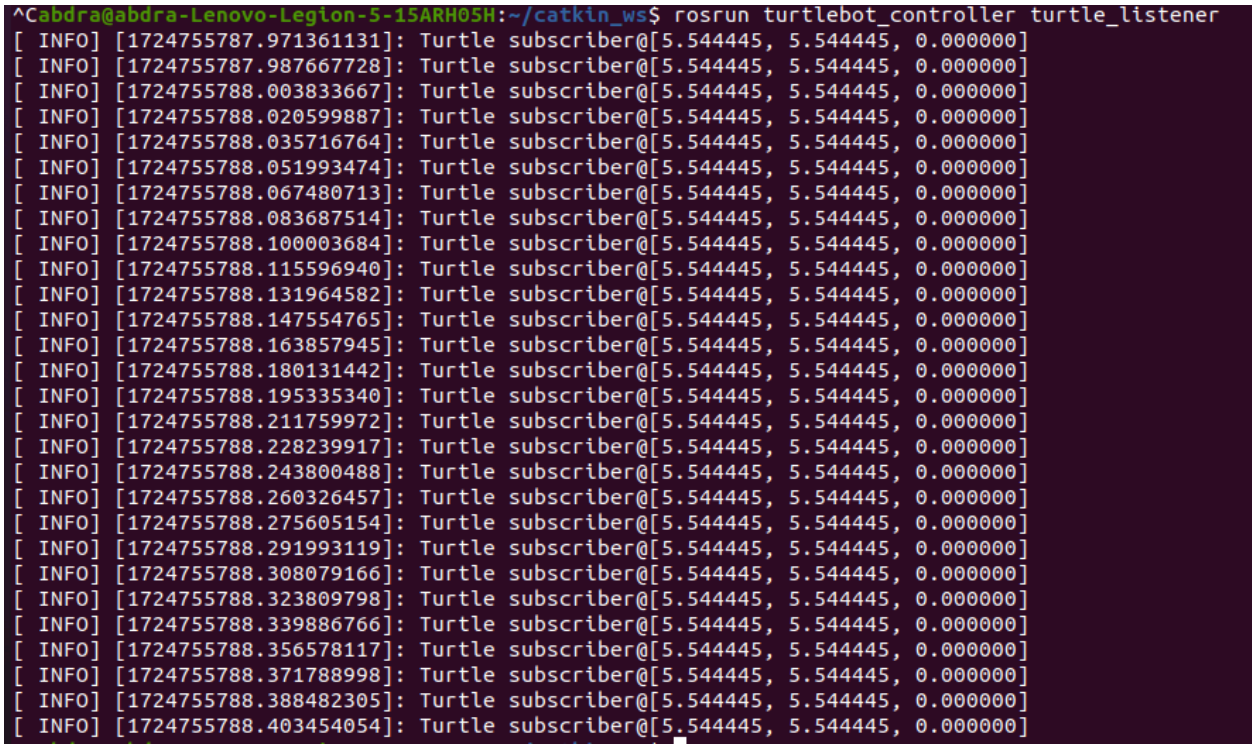


Figure 22. Turtle Listener Output (Task 2)

6. Task 3

After the success of task 2, we felt a need to not just get the position of the turtle, but also to be able to control it from our own code. For this reason, we have written a turtle subscriber to send the desired velocities to the turtle. Result was good – the turtle was moving in a circle and the listener reported back its position.

```cpp
#include "ros/ros.h"
#include "turtlesim/Pose.h"
#include "geometry_msgs/Twist.h"
ros::Publisher pub;
//pub = nh.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1);

void turtleCallback(const turtlesim::Pose::ConstPtr& msg)
{
        ROS_INFO("Turtle subscriber@[%f, %f, %f]",
        msg->x, msg->y, msg->theta);
        geometry_msgs::Twist my_vel;
        my_vel.linear.x = 1.0;
        my_vel.angular.z = 1.0;
        pub.publish(my_vel);
}



int main (int argc, char **argv)
{
        // Initialize the node, setup the NodeHandle for handling the communication with the ROS
        //system
        ros::init(argc, argv, "turtlebot_subscriber");
        ros::NodeHandle nh;
        // Define the subscriber to turtle's position
        ros::Subscriber sub = nh.subscribe("turtle1/pose", 1,turtleCallback);
        pub = nh.advertise<geometry_msgs::Twist>("turtle1/cmd_vel", 1);
        ros::spin();
        return 0;
}
```

Figure 23. Turtle Subscriber modified code (Task 3)

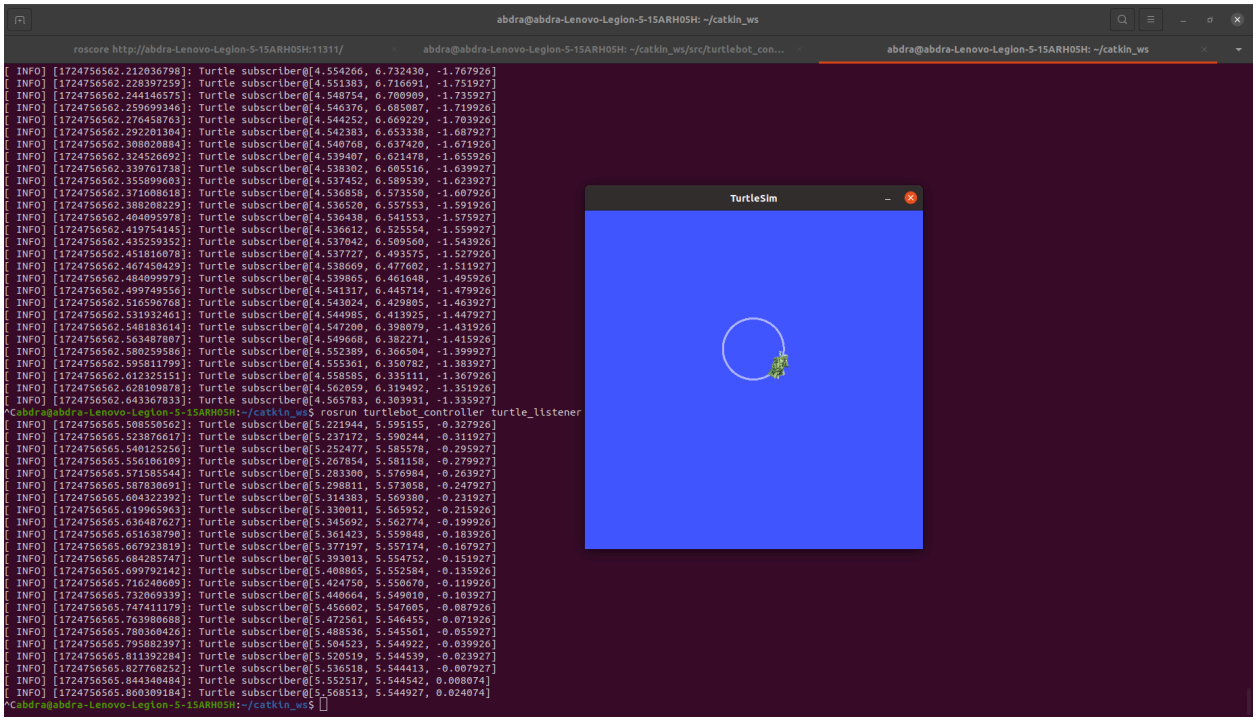


Figure 24. Turtle Listener Output & Turtle trajectory (Task 3)

7. Task 4

While the Publish/Subscribe architecture is highly effective for decoupling components and managing data flow in distributed systems, it is not ideal for scenarios requiring Remote Procedure Call (RPC) or Request/Reply interactions. These interactions, which are common in many distributed systems, require a different communication model where a client sends a request and waits for a specific reply from a server. The nature of Publish/Subscribe, with its one-to-many communication pattern, doesn't align well with the direct, synchronous exchanges required by RPC. For this reason, we needed to add a service to the cpp code.

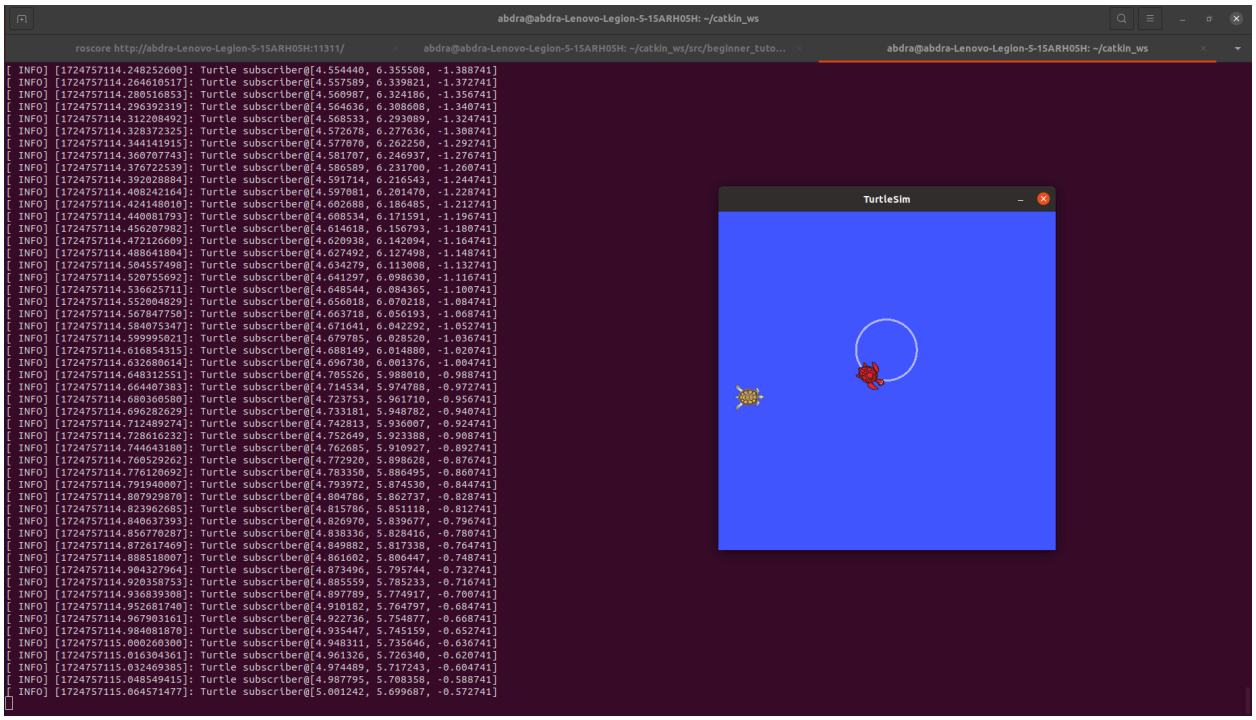


Figure 25. Turtle Listener Output, Turtle trajectory, Spawn of new turtle (Task 4)

FailTeamLab1 / src / turtlebot_controller / src / spawn_client.cpp

Meneralllka This is the code that we produced for the Lab 1 tasks.

Code Blame 28 lines (24 loc) · 590 Bytes Code 55% faster with GitHub

```cpp
#include "ros/ros.h"
#include "turtlesim/Spawn.h"
#include <cstdlib>

int main(int argc, char **argv)
{
  ros::init(argc, argv, "spawn_turtle_client");
  ros::NodeHandle nh;
  ros::ServiceClient client1 = nh.serviceClient<turtlesim::Spawn>("/

  turtlesim::Spawn srv1; // define a /spawn service message
  srv1.request.x = 1.0;
  srv1.request.y = 5.0;
  srv1.request.theta = 0.0;
  srv1.request.name = "Turtle_FAILED_TEAM";

  if (client1.call(srv1))
  {
    ROS_INFO("Spawned");
  }
  else
  {
    ROS_ERROR("Failed to call service add_two_ints");
    return 1;
  }

  return 0;
}
```

Figure 26. C++ code with added service to spawn a new turtle (Task 4)

8. Task 5

For the final task, we had to make the turtle move along a desired trajectory. For this task, we have used terminal commands to specify turtle's speed in each axis.

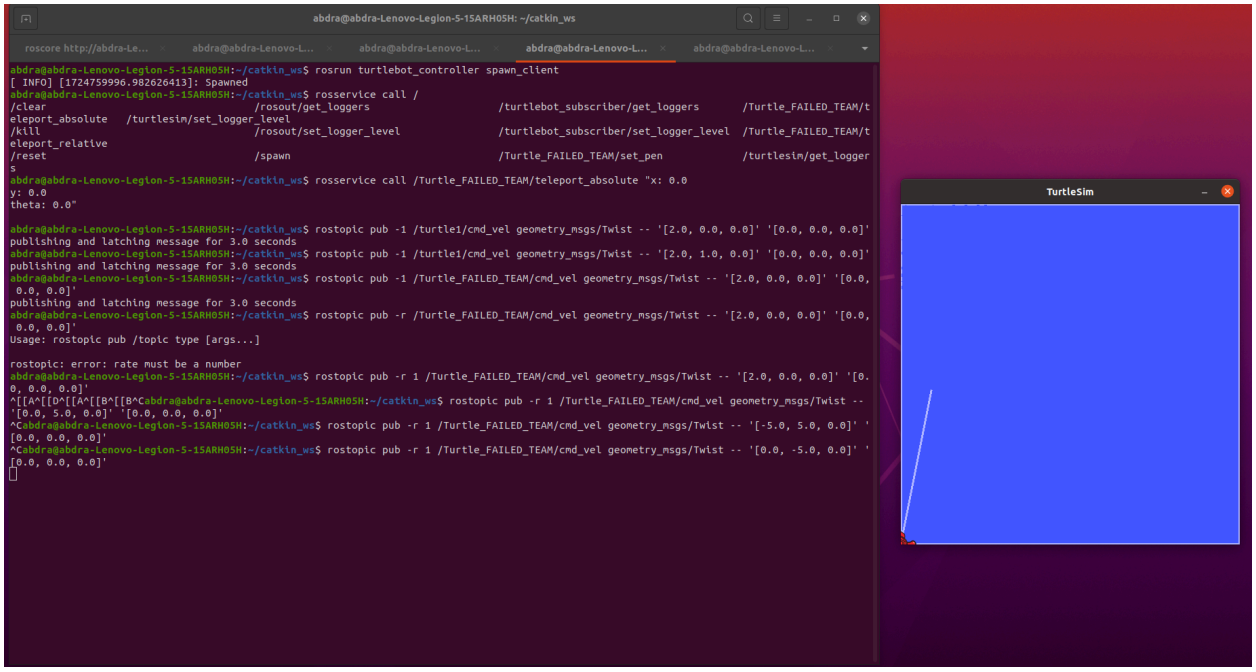


Figure 27. Turtle moving in rectangle trajectory (Task 5)

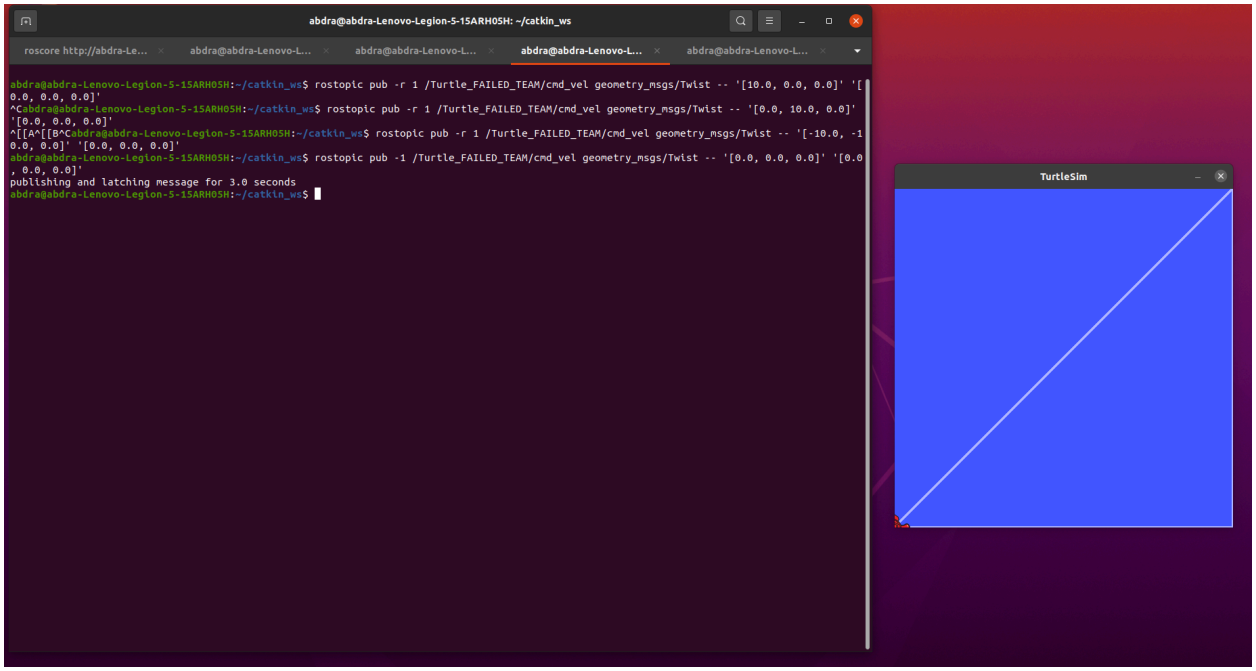


Figure 27. Turtle moving in triangle trajectory (Task 5)

9. Conclusion

In this laboratory report, we explored the practical applications of the Robot Operating System (ROS) through a series of structured tasks that progressively introduced us to the foundational concepts and tools within ROS. The exercises undertaken demonstrated the flexibility and power of ROS in managing robotic systems, from basic communication between nodes to more complex tasks involving real-time control and feedback.

Throughout the tasks, we successfully implemented and tested various components, including the creation of ROS packages, the implementation of publisher and subscriber nodes, and the manipulation of a virtual robot within the Turtlesim environment. By working through these tasks, we gained hands-on experience in setting up and configuring a ROS environment, developing custom message types, and debugging ROS applications.

One of the key takeaways from this lab is the importance of modularity and reusability in robotic software design, as exemplified by ROS's architecture. The ability to easily integrate different nodes and leverage a vast library of pre-existing packages significantly accelerates the development process, making ROS an indispensable tool in both academic research and industrial robotics.

The exercises also highlighted the strengths and limitations of different communication models within ROS, such as the Publish/Subscribe architecture and the use of services for request-reply interactions. Understanding when and how to use these models is crucial for developing efficient and effective robotic systems.

Overall, this lab has provided a solid foundation in ROS, equipping us with the knowledge and skills necessary to tackle more complex robotic projects in the future. The experience gained here will be invaluable as we continue to explore the capabilities of ROS in various applications and contribute to the ongoing development of robotic technologies.