Robot Operating System (ROS) Fundamentals

Foundations of Robotics

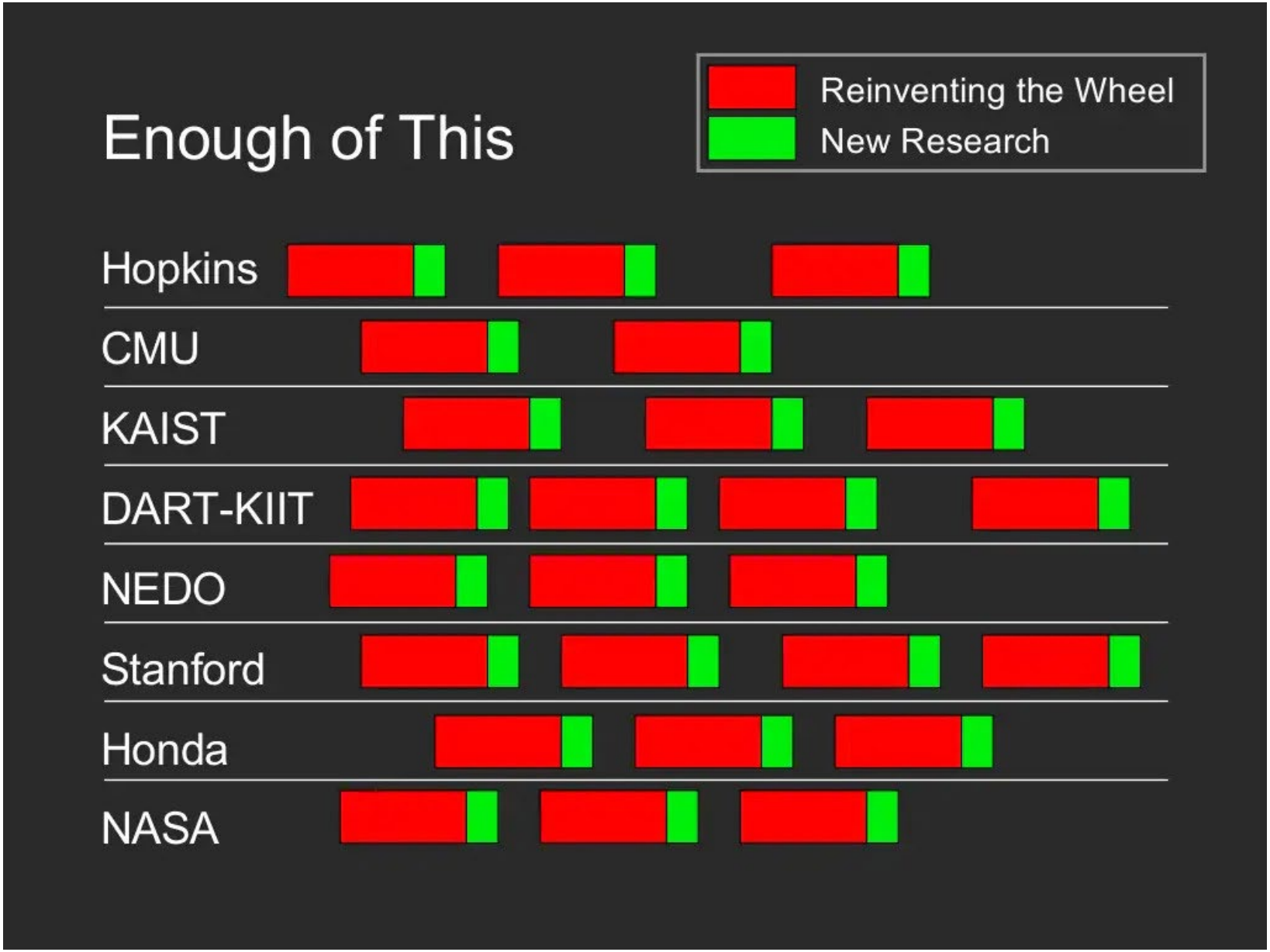
Cornell University

Cornell Bowers CIS
Computer Science

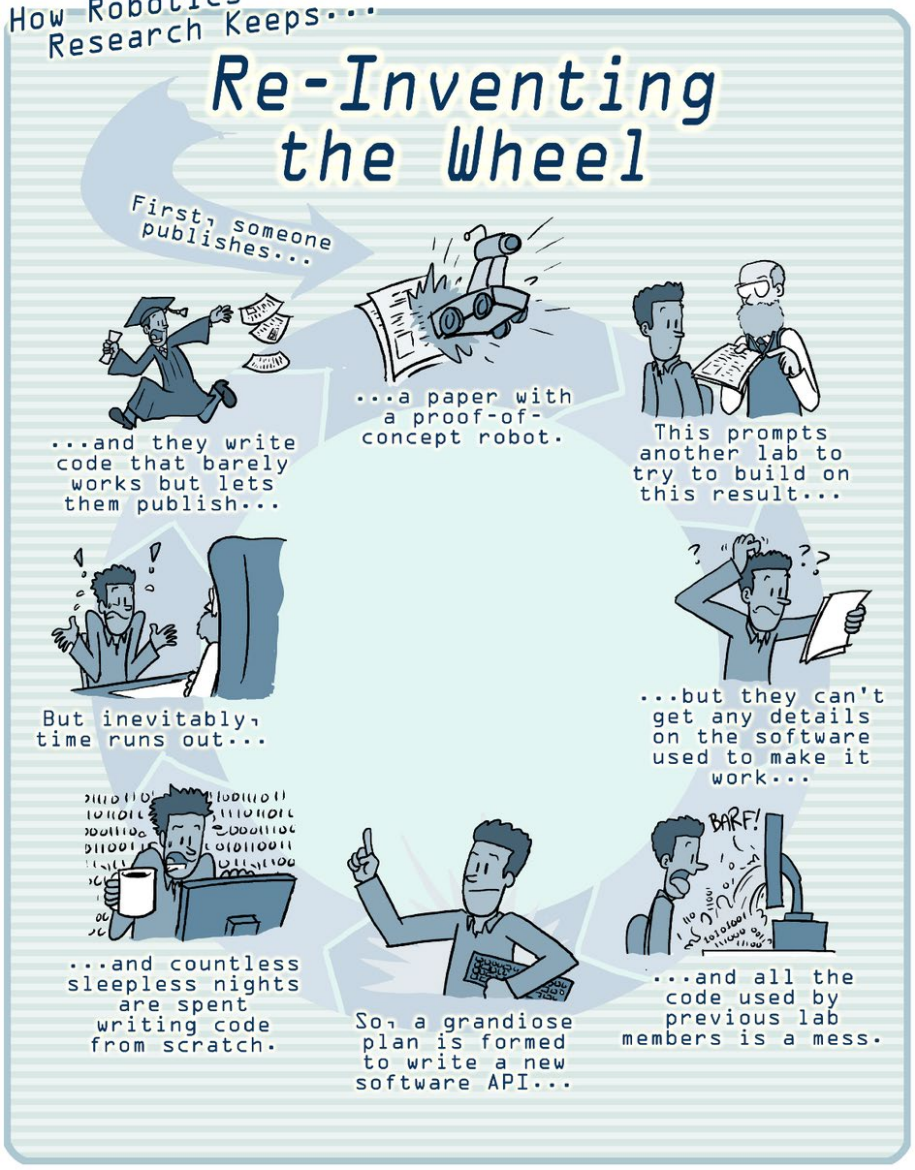
Lab A has built their own software stack for their robot.
Lab B just got a new robot for their research. What should they do?

A. *Build their own stack*

B. *Pick the stack from Lab A*



From the [original pitch deck] from 2006 used to raise donor money for the PR1 and ROS project at Stanford.

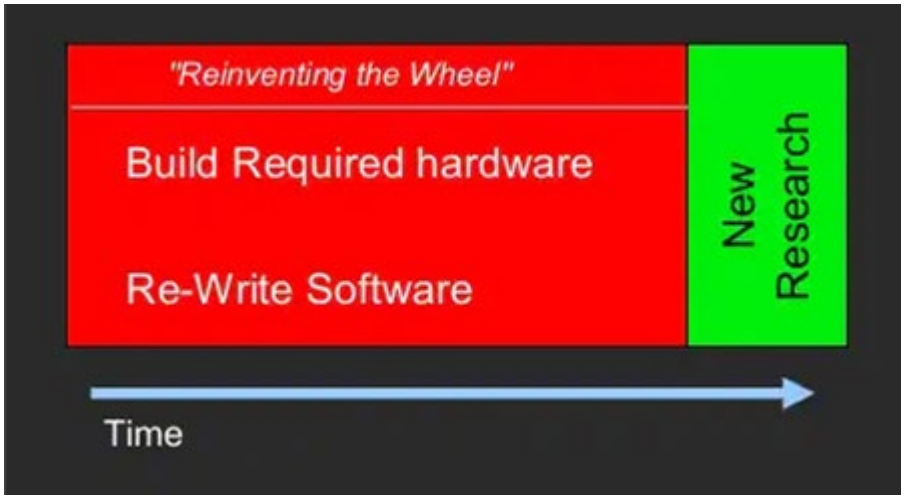


From IEEE Spectrum article, The Origin Story of ROS, the Linux of Robotics

Why reinventing the wheel is time-consuming?

Robot systems are complicated.

Maintaining multiple robot systems is difficult.



Before ROS

Many labs had their own robot drivers, communication protocols, other software modules compatible with those drivers and communication protocols.

Comes 

History
Stanford Personal Robotics Program

Philosophy
Enough functionality that it saves people's time. Enough flexibility that they can do whatever they want.

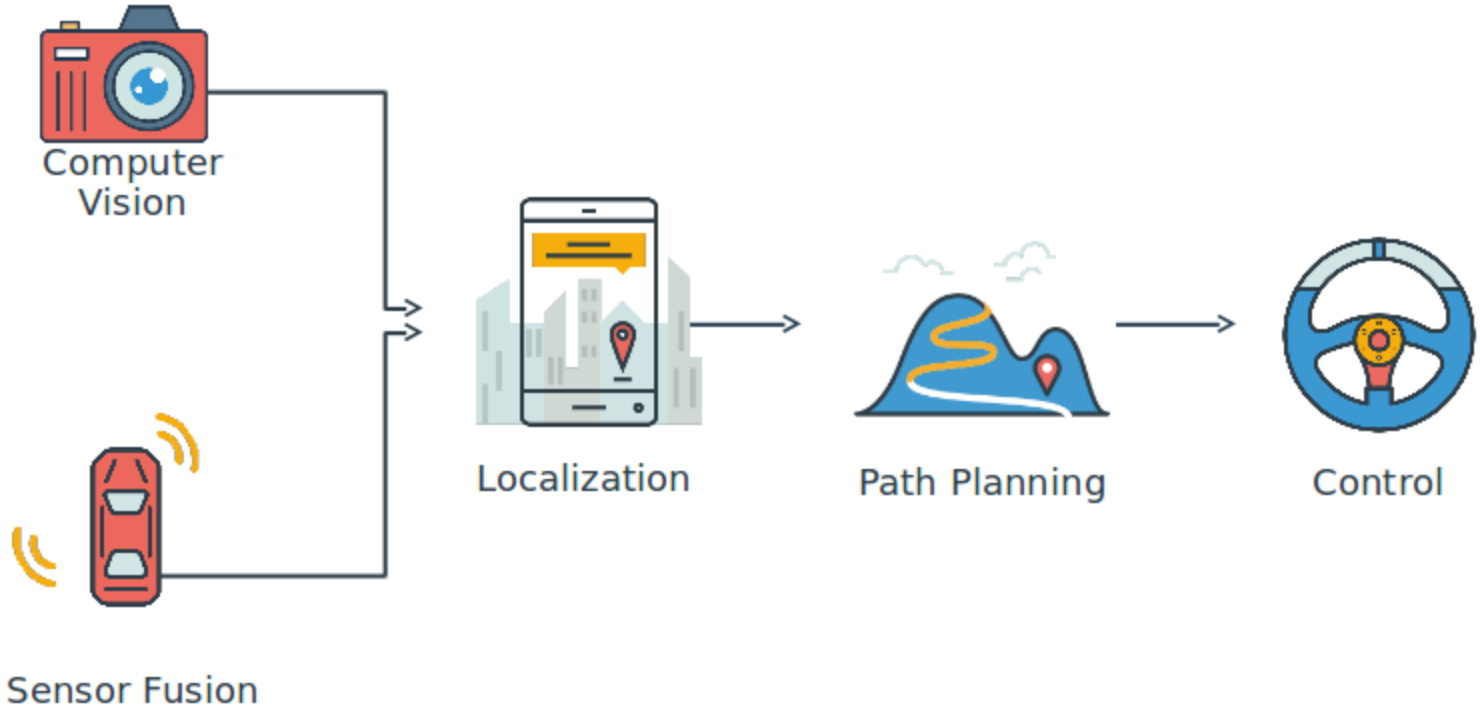
A standard framework for developing robot systems.

What is ROS?

A “meta” operating system. Provides most services that you’d expect from an operating system.

“The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms. ”

ROS was built from the ground up to encourage collaborative robotics software development.



Advantages

- Modularity: Each functionality can be run as an individual program
- Distributed: Programs can run on multiple computers across different networks
- Reproducible software for robots
- Open source packages for several robot functionalities
- Provides interfaces/API for both Python and C++.

Installation

First, you need to install ROS on your system [installation guide]

We will be using ROS noetic on Ubuntu 20.04 for this course.

Any given robot system has multiple components.

- Hardware communication
- Perception
- Localization
- Planning
- Controls
- Simulation

Husky

- husky_base
- husky_bringup
- husky_control
- husky_description
- husky_desktop
- husky_gazebo
- husky_msgs
- husky_navigation
- husky_robot
- husky_simulator
- husky_viz

ROS workspace

Folder to host, modify and update all the components of a robotic system in one place.

```
workspace_folder/          -- WORKSPACE
  src/                     -- SOURCE SPACE
    CMakeLists.txt         -- The 'toplevel' CMake file
    package_1/
      CMakeLists.txt
      package.xml
      ...
    package_n/
      CATKIN_IGNORE        -- Optional empty file to exclude package_n from being processed
      CMakeLists.txt
      package.xml
      ...
  build/                   -- BUILD SPACE
  devel/                   -- DEVELOPMENT SPACE
  install/                 -- INSTALL SPACE
```

Create a ROS Workspace

http://wiki.ros.org/catkin/Tutorials/create_a_workspace

ROS Packages

and how to *create* one

Software in ROS is organized in packages. A package might contain ROS nodes, a ROS-independent library, a dataset, configuration files, or anything else that logically constitutes a useful module.

ROS Packages

```
workspace_folder/          -- WORKSPACE
  src/                     -- SOURCE SPACE
    husky_object_detection/
      CMakeLists.txt
      package.xml
      src/
      launch/
      include/
      config/
```

ROS Packages

“The goal of ROS packages it to provide this useful functionality in an easy-to-consume manner so that software can be easily reused.

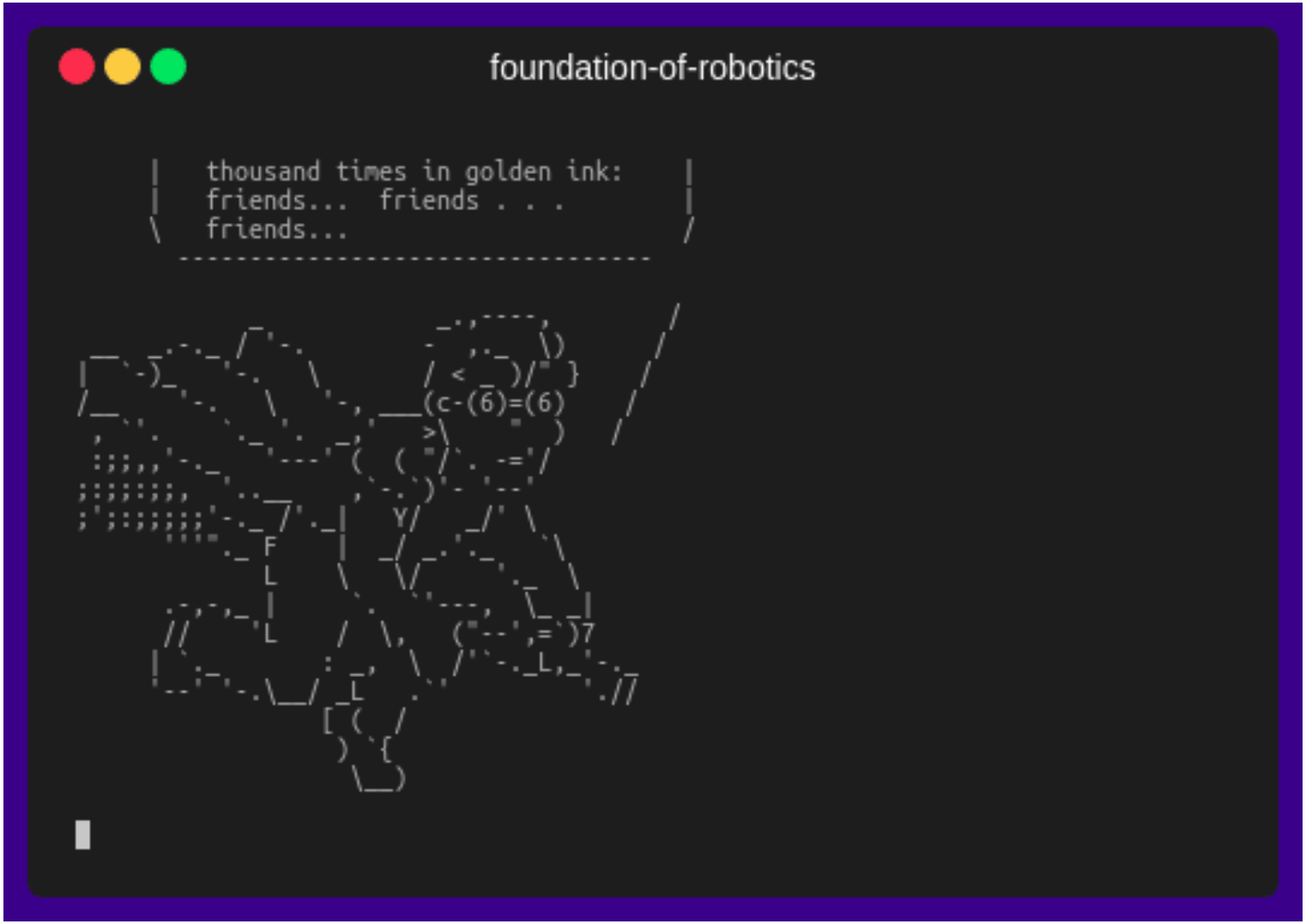
ROS packages follow a "Goldilocks" principle: enough functionality to be useful, but not too much that the package is heavyweight and difficult to use from other software.”

Create a ROS package

```
$ catkin_create_pkg <package_name> [depend1] [depend2]
```

`catkin_create_pkg` requires that you give it a package_name and optionally a list of dependencies on which that package depends.

Create a ROS package



http://wiki.ros.org/ROS/Tutorials/CreatingPackage

You have everything you need in a ROS package. What next?

Building a ROS Package

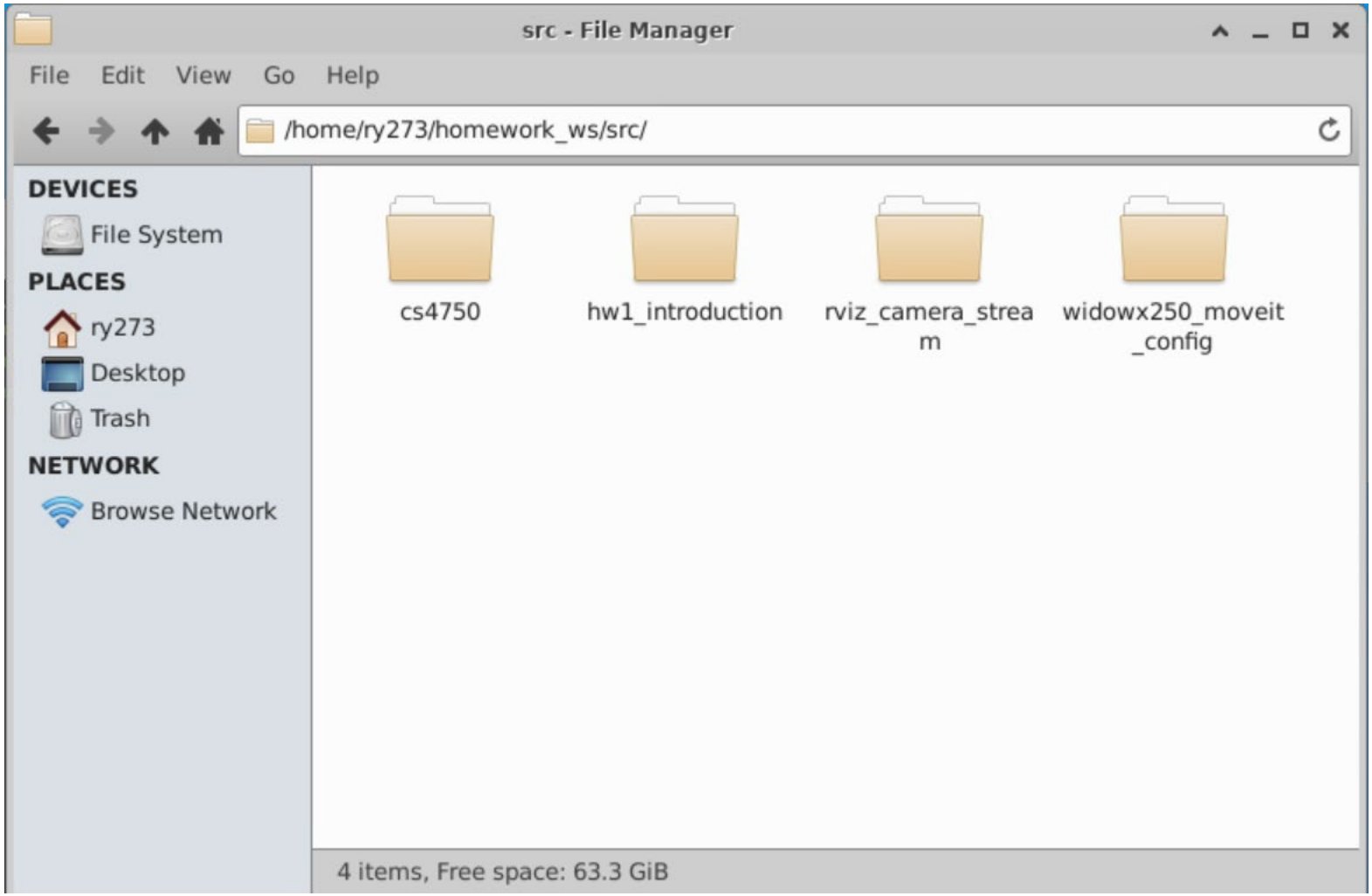
A build system is responsible for generating 'targets' from raw source code that can be used by an end user.

Forms of targets: libraries, executable programs, generated scripts, exported interfaces (e.g. C++ header files) or anything else that is not static code.

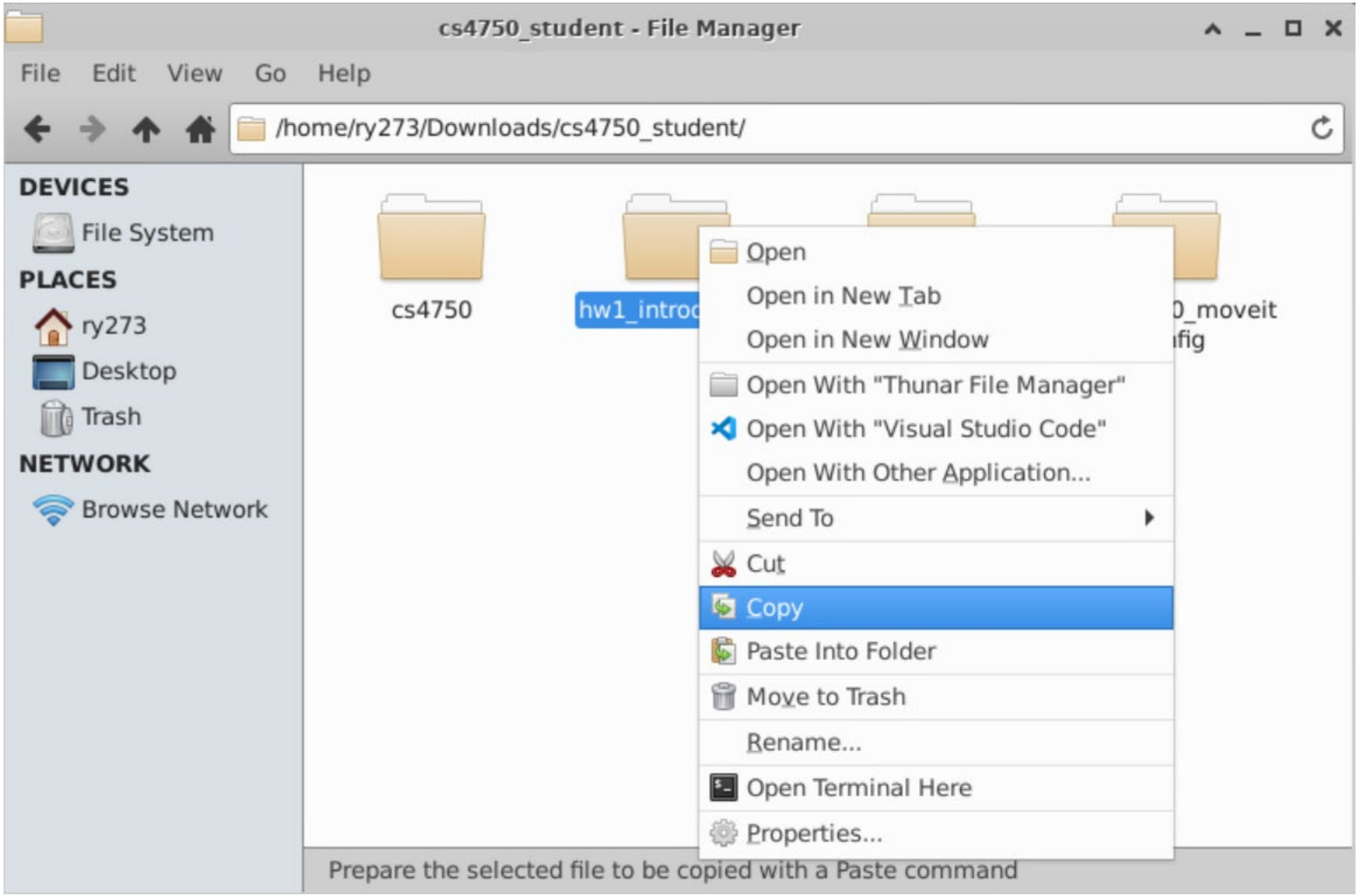
In ROS terminology, source code is organized into 'packages' where each package typically consists of one or more targets when built.

`catkin build` in `/src` directory inside the package

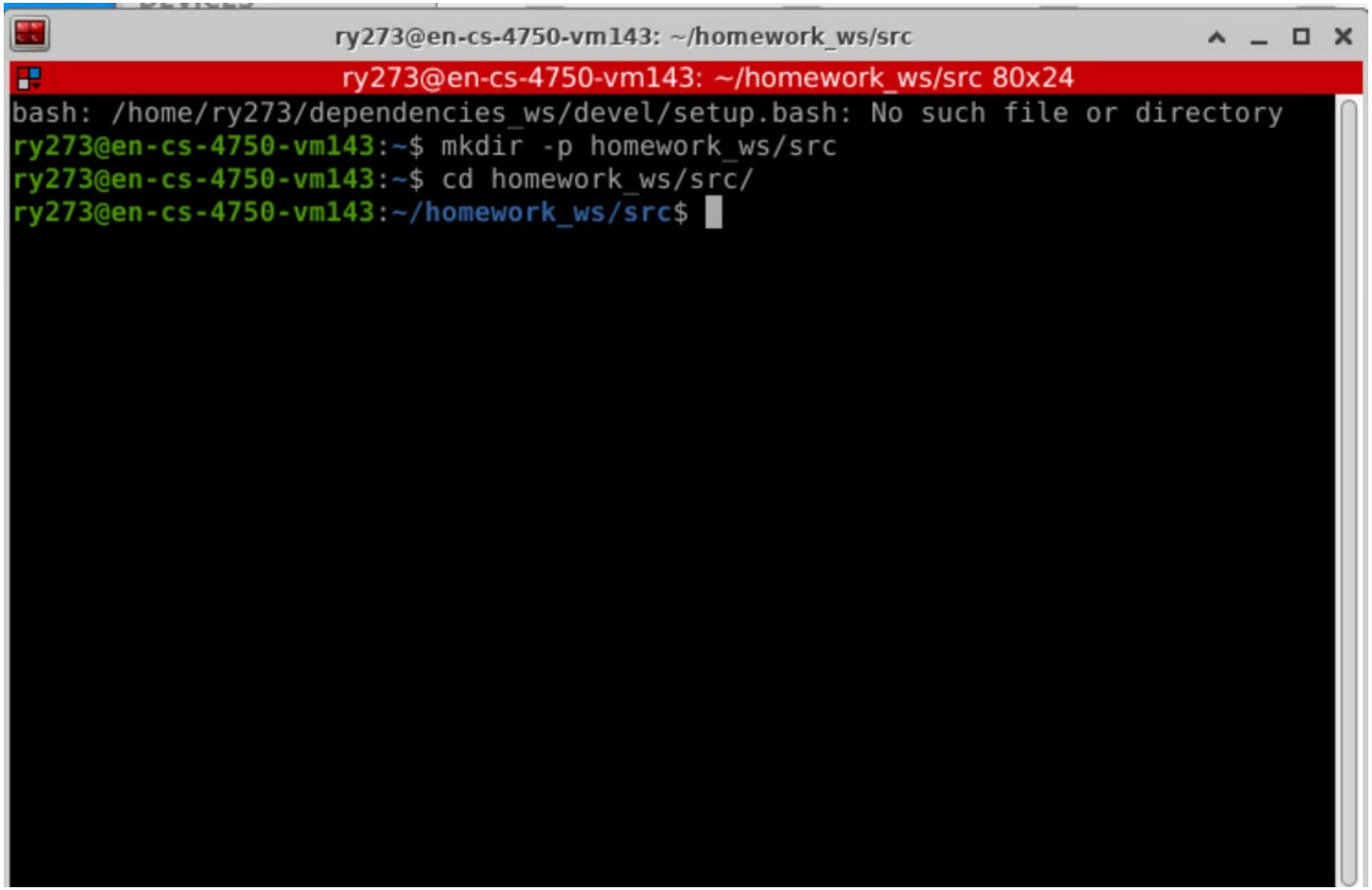
Building a ROS Package: HW



Building a ROS Package: HW



Building a ROS Package: HW



Navigating ROS filesystem

rospack allows you to get information about packages.

```
$ rospack find [package_name]
$ rospack find rospy
```

roscd allows you to change directory (cd) directly to a package.

```
$ roscd <package>[/subdir]
$ roscd rospy
```

rosls allows you to ls directly in a package by name rather than by absolute path.

```
$ rosls rospy
cmake  package.xml rosbuild
```

Navigating ROS filesystem

tab complete: It can get tedious to type out an entire package name.

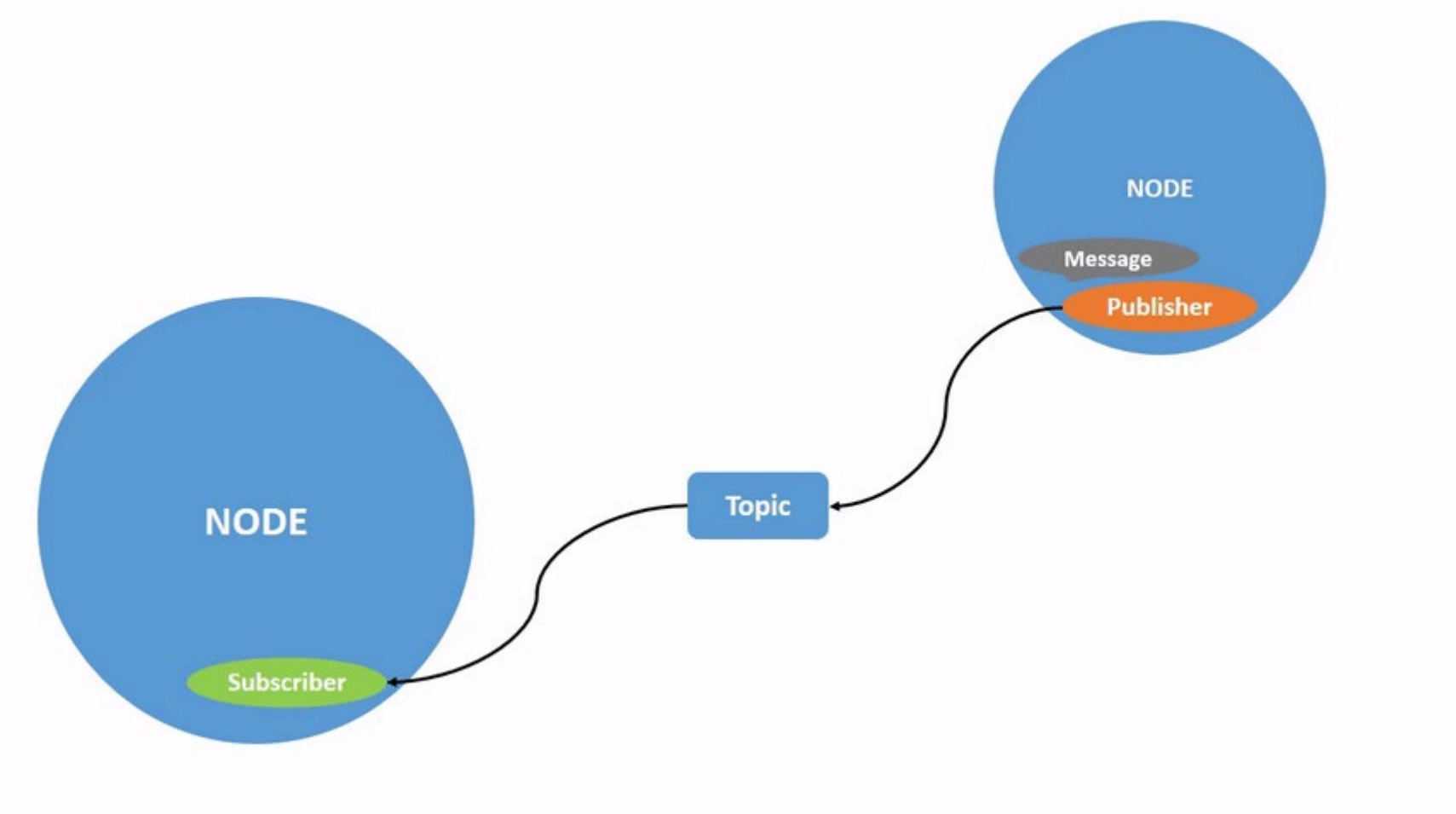
```
$ roscd joint_tra<<< now push the
TAB key >>>
```

```
$ roscd joint_trajectory_controller/
```

You need to command Husky to go near the white gate in front of it.

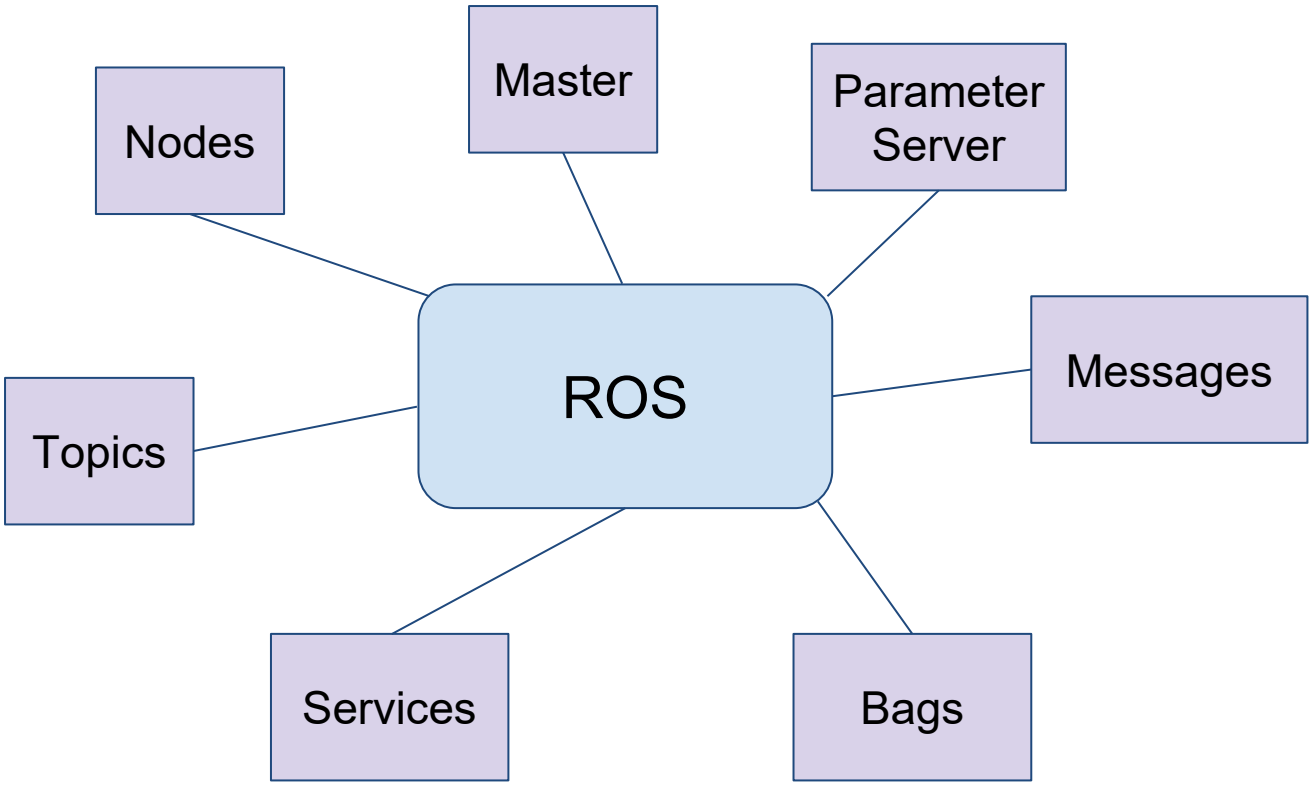
How would you share the camera data with robot's planner?

Use Publishers and Subscribers

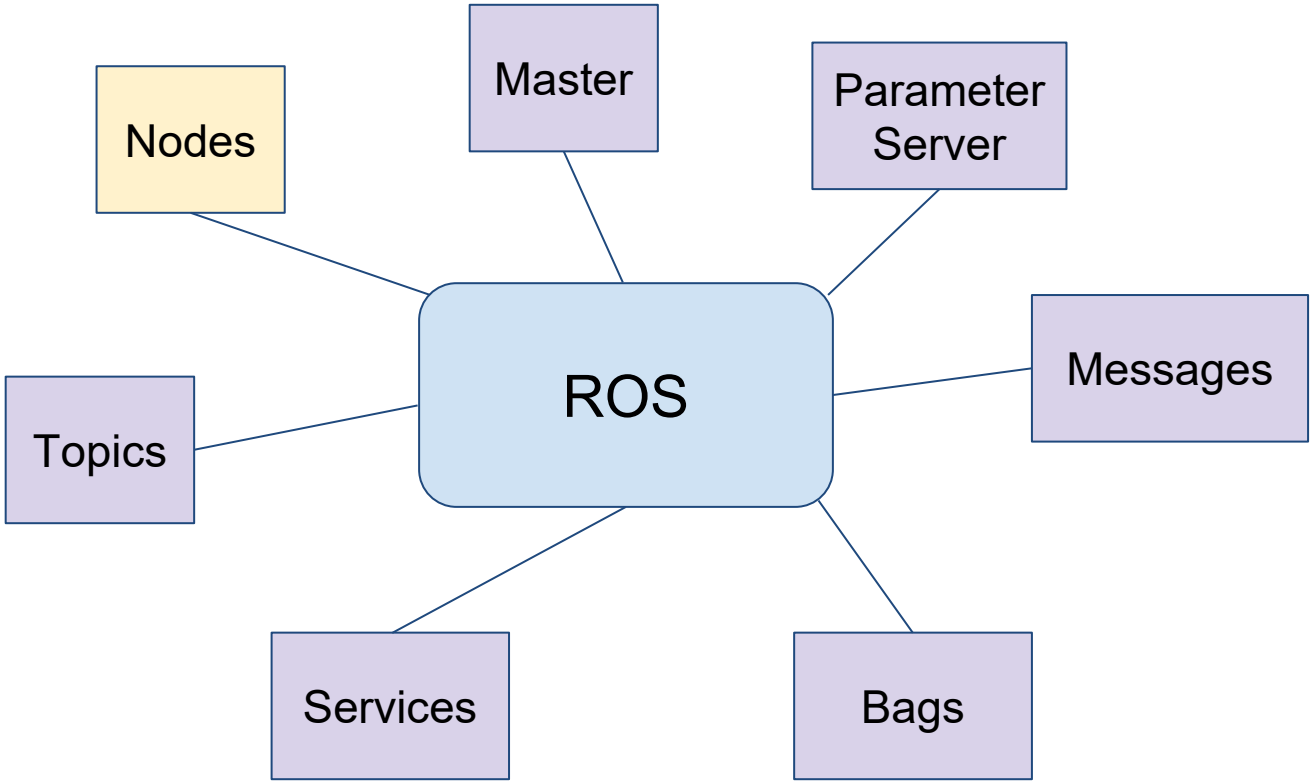


ROS Publisher Subscriber Model (Courtesy:ROS2)

Basic ROS components



Basic ROS components



Nodes

Nodes are the executables that take care of computation.

Different nodes can interact with each other.

Nodes are implemented using ROS client libraries like `roscpp` or `rospy`

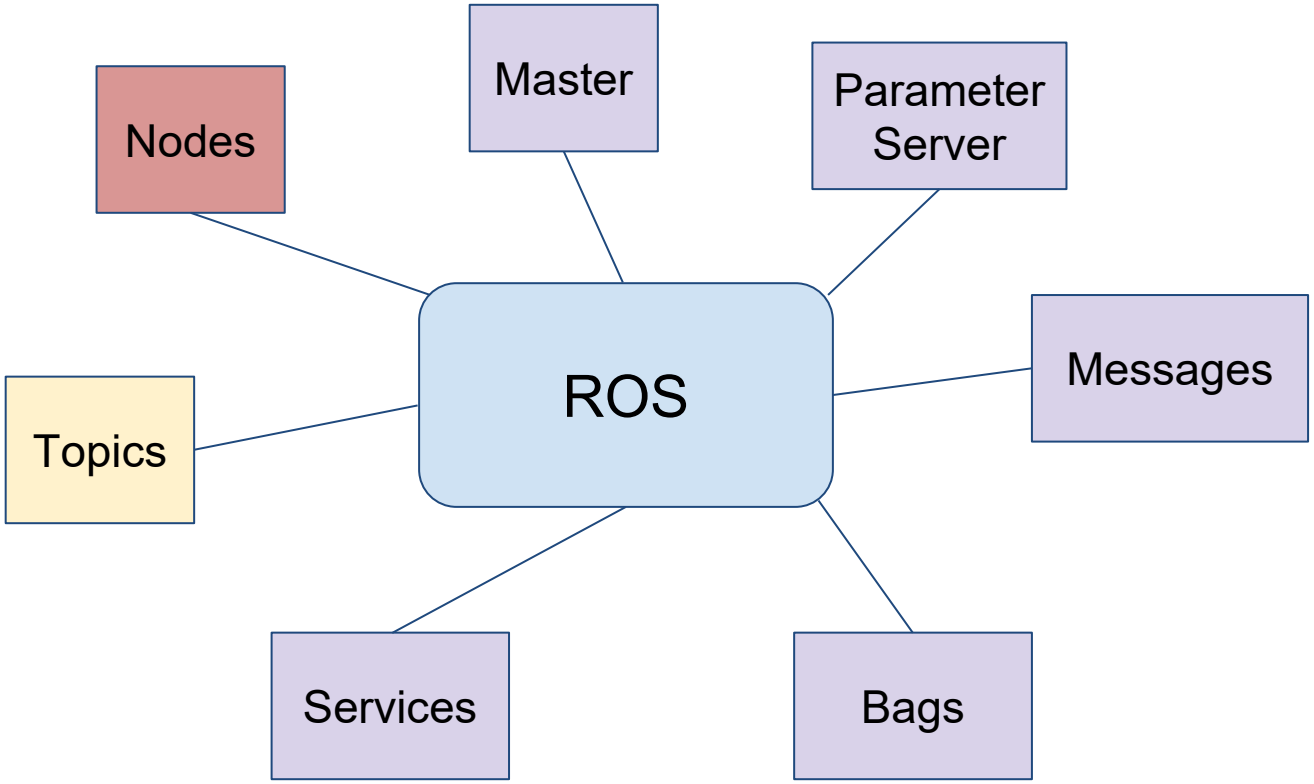
rosrun

Allows you to run executables from any given package without knowing the full path to the package.

```
$ rosrun <package_name> <executable_file_name>
```

```
$ rosrun husky_camera camera_publisher
$ rosrun husky_planner visual_navigation
```

Basic ROS components



Topics

Topics are named routes over which nodes exchange messages.

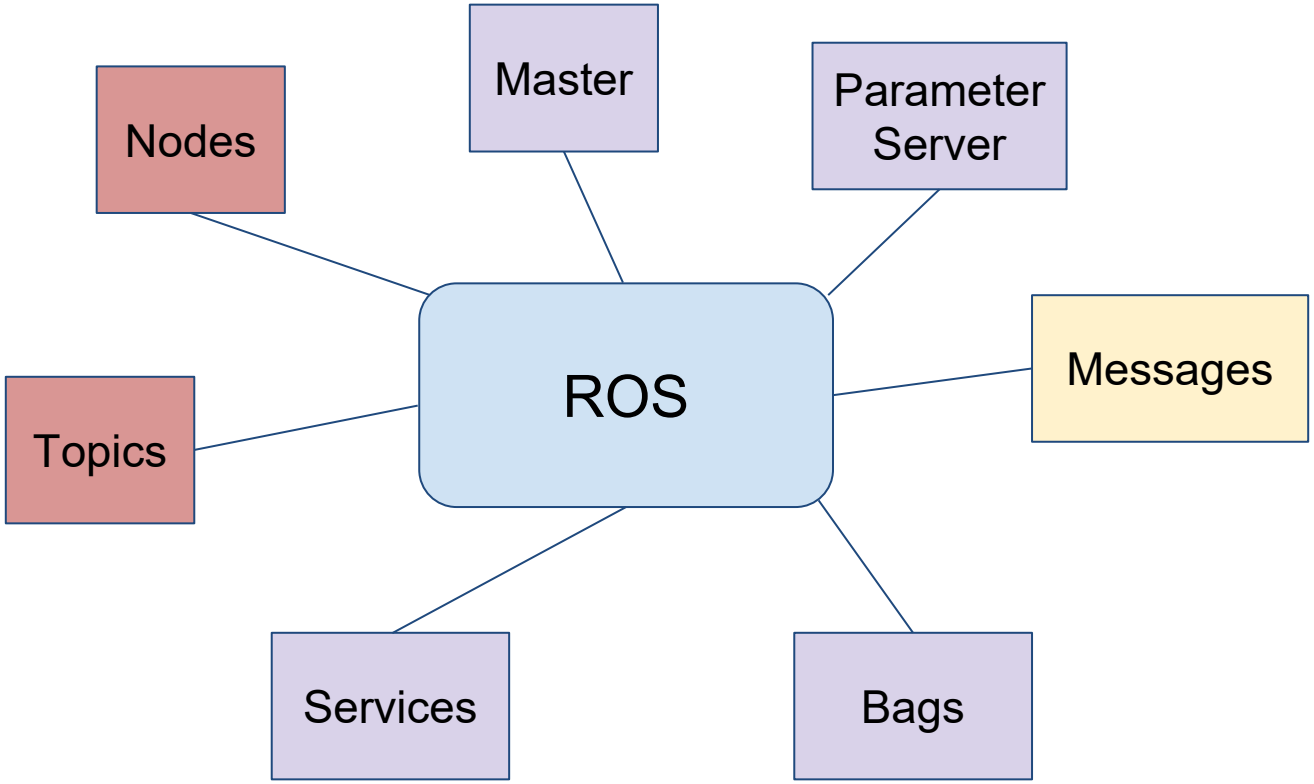
rostopic command line tool

The rostopic command-line tool displays information about ROS topics.

`rostopic echo` print messages to screen
`rostopic info` print information about active topic
`rostopic list` list active topics
`rostopic type` print topic type

See wiki for more details.

Basic ROS components



Msg

Nodes communicate with each other by publishing **messages** to topics.

A **message (msg)** is a simple data structure, comprising typed fields.

Msg files

msg files are simple text files for specifying the data structure of a message.

geometry_msgs/Point Message

File: geometry_msgs/Point.msg

Raw Message Definition

```
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

Compact Message Definition

```
float64 x
float64 y
float64 z
```

geometry_msgs/Pose Message

File: geometry_msgs/Pose.msg

Raw Message Definition

```
# A representation of pose in free space, composed of position and orientation.
Point position
Quaternion orientation
```

Compact Message Definition

```
geometry_msgs/Point position
geometry_msgs/Quaternion orientation
```

geometry_msgs/PoseArray Message

File: geometry_msgs/PoseArray.msg

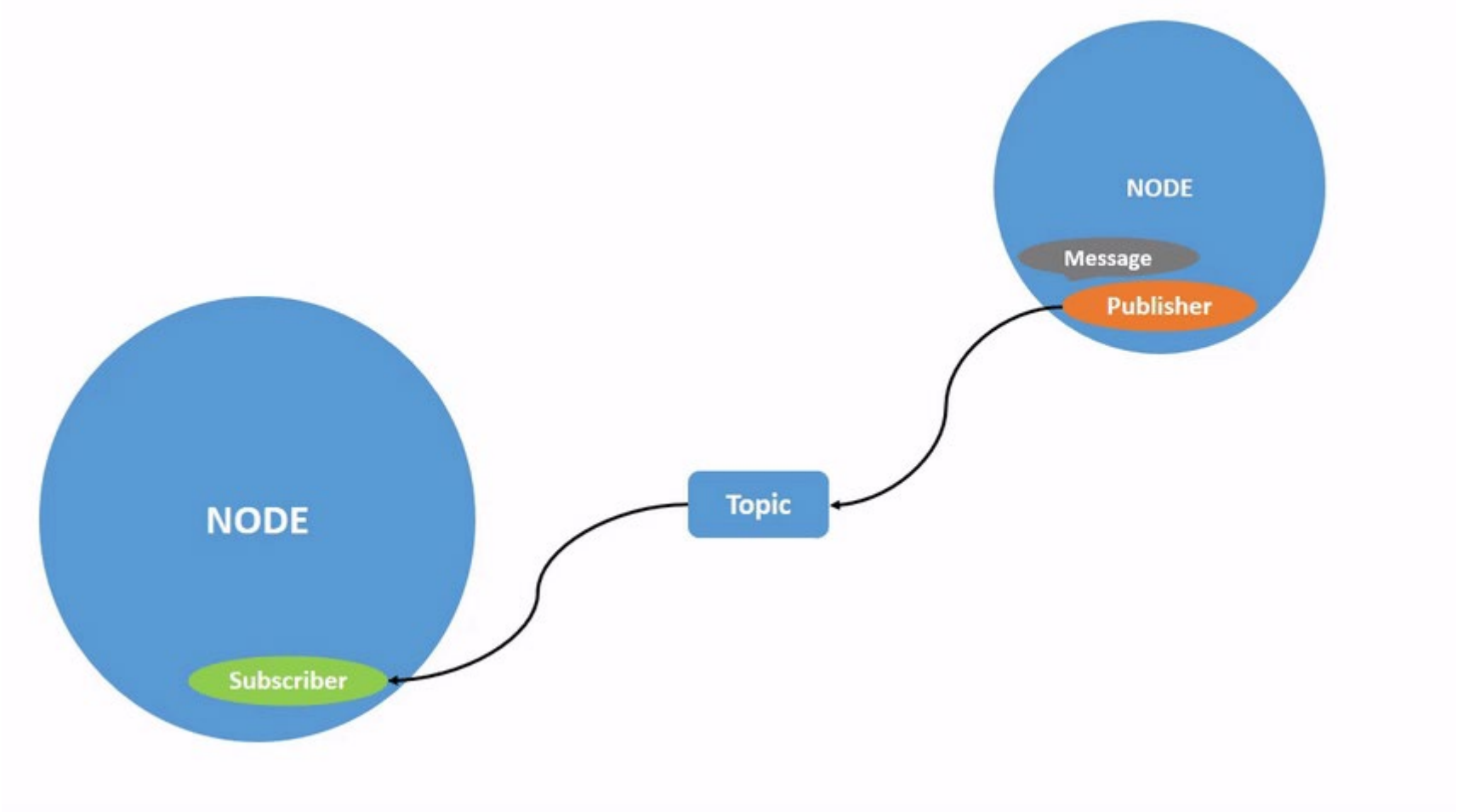
Raw Message Definition

```
# An array of poses with a header for global reference.

Header header

Pose[] poses
```

Compact Message Definition

```
std_msgs/Header header
geometry_msgs/Pose[] poses
```

Revisiting Publishers and Subscribers



ROS Publisher Subscriber Model (Courtesy:ROS2)

Publishers and Subscribers

Publisher (Pub): a **node** that publishes a specific type of **message** over a given **topic**

Subscriber (Sub): a **node** that subscribes a specific type of **message** over a given **topic** and invokes a **callback** function to process received message.

Pub/Sub is a form of asynchronous communication.

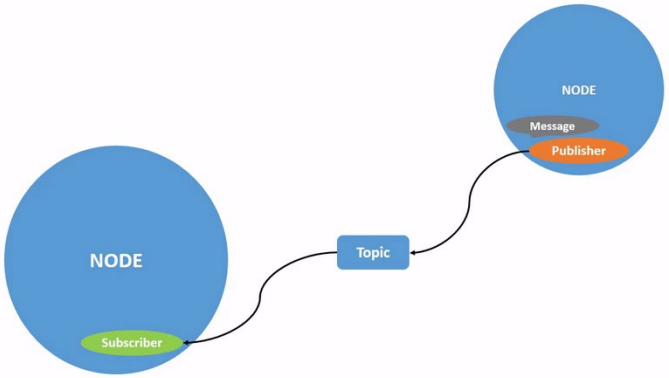
Implementing a simple publisher/subscriber in Python

rospy.Publisher(TOPIC_NAME, MSG_TYPE, QUEUE_SIZE)

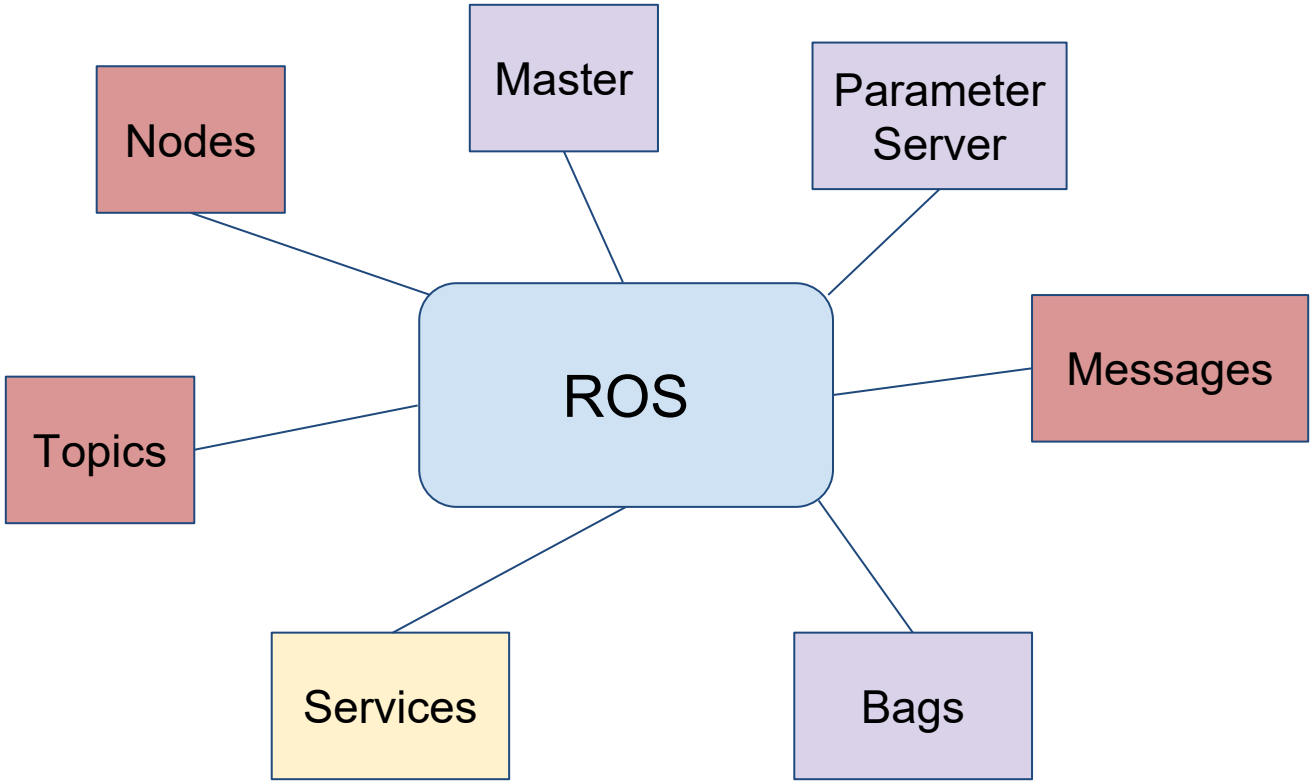
```
pub = rospy.Publisher('camera/image_raw', Image,
queue_size=10)
pub.publish(rgb_image)
```

rospy.Subscriber(TOPIC_NAME, MSG_TYPE, CALLBACK_FUNCTION)

```
rospy.Subscriber("camera/image_raw", Image,
planner_callback)

def planner_callback(msg):
       # This function will be called
message is received.
       # Plan a path given the Image r
```



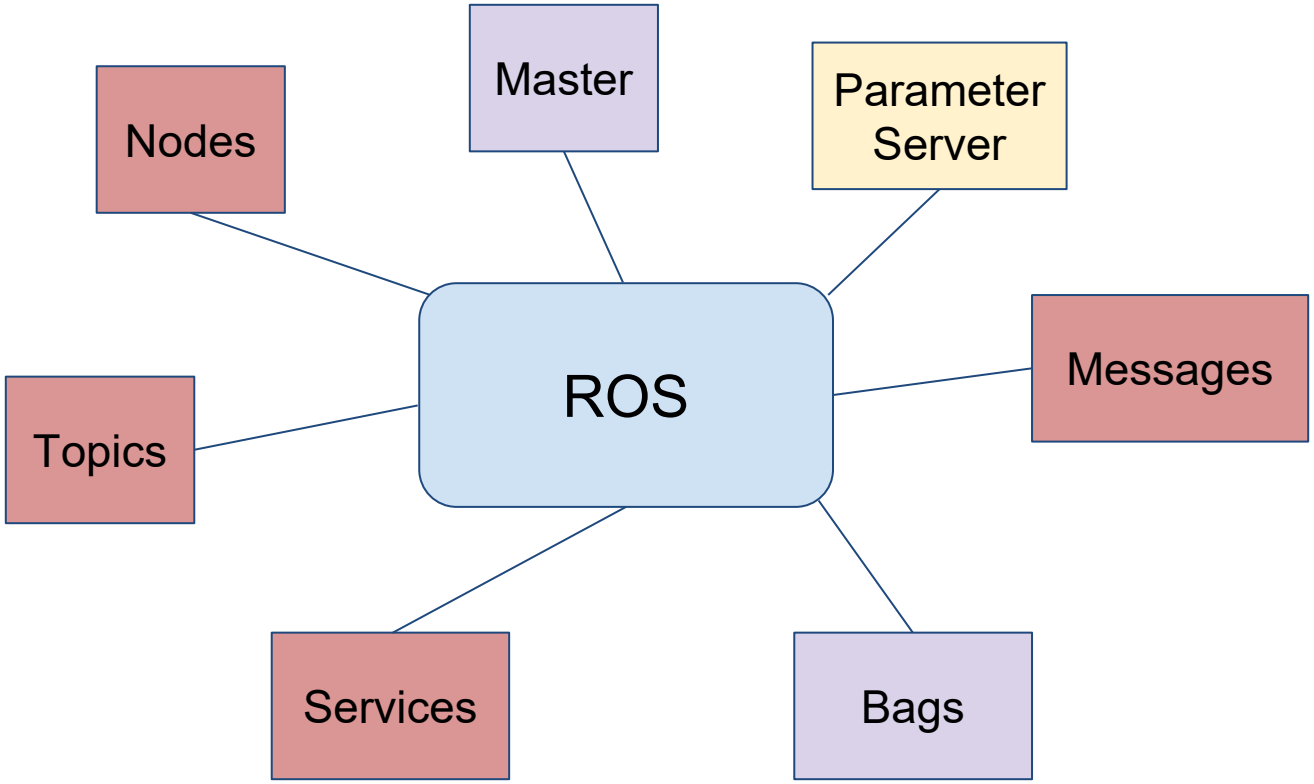
Basic ROS components



A robot wants to ask a voice assistant about the weather today. What communication protocol should it use?

A. *Publisher / Subscriber*

B. *Service / Client*


Master
Nodes
Parameter Server
ROS
Messages
Topics
Services
Bags

Parameter Server

Parameter Server is simply a shared dictionary used by nodes to store and retrieve parameters at runtime.

Parameters

Parameters are named following the ROS naming convention.

For example:

```
floor_map: “$(find husky_navigation)/maps/gates_101.yaml”
```

Parameters can be 32-bit integers, booleans, strings, doubles, lists, etc.

rosparam command-line tool

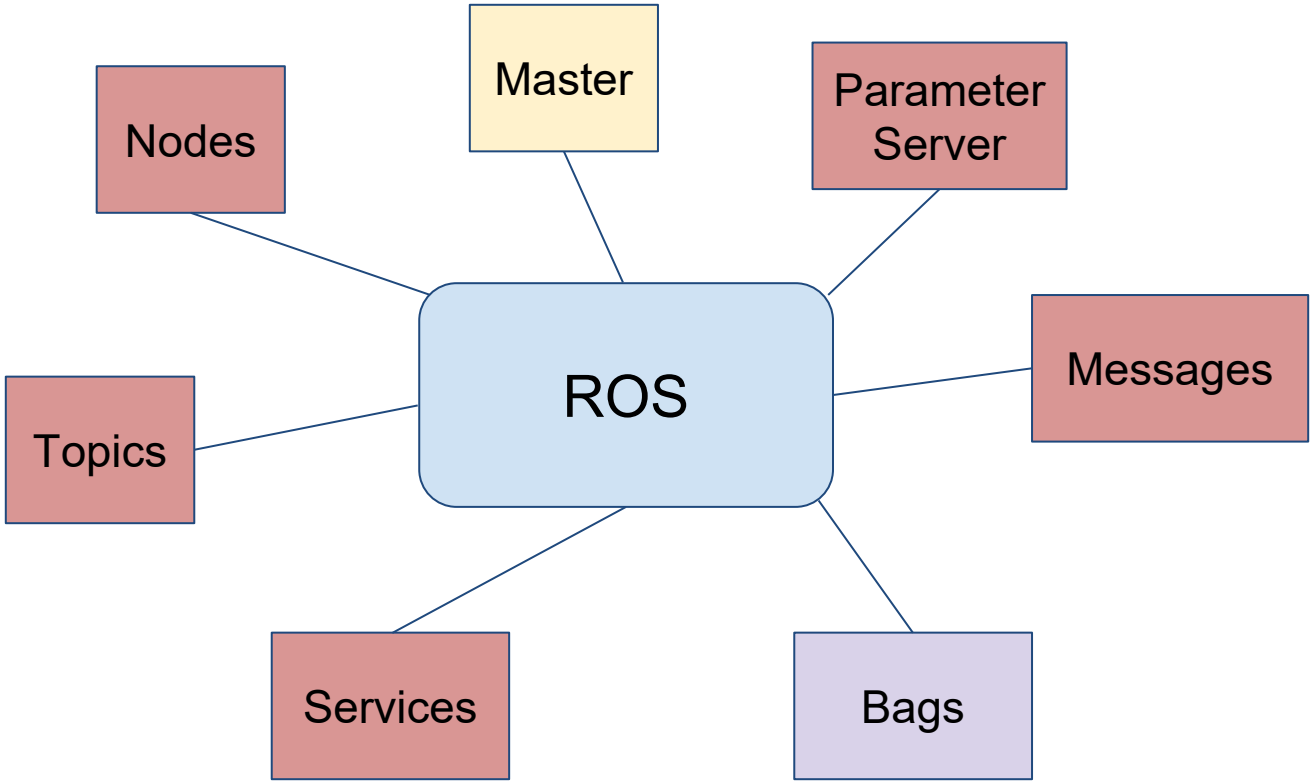
`rosparam list`
list all parameters' names

`rosparam get <param-name>`
get parameter's value

`rosparam set` <param-name>
set parameter to a value

`rosparam delete <param-name>` delete a parameter

See wiki for more details.


Master
Nodes
Parameter Server
ROS
Messages
Topics
Services
Bags

ROS Master

When we run the program, there maybe several independent nodes running at the same time.

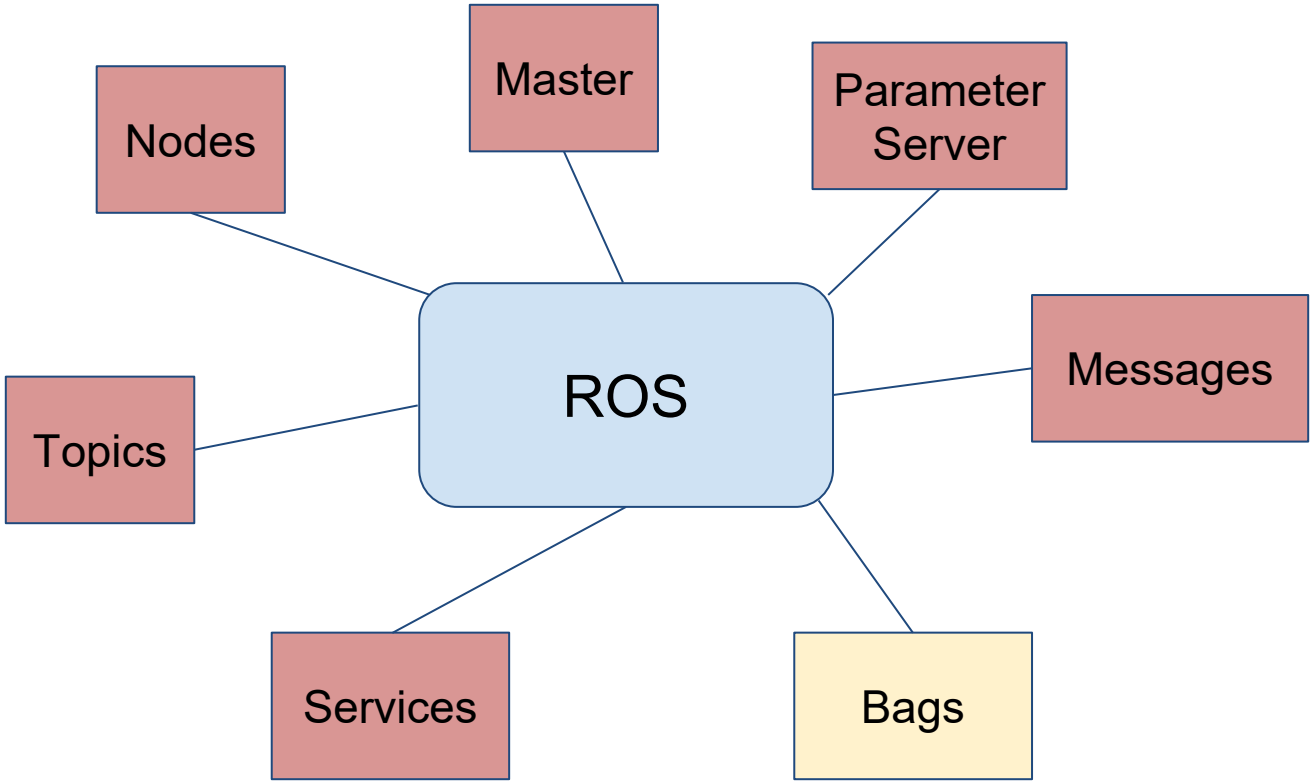
We want them to share data with each other. Responsible for communication **between** the nodes.

Maintains registries for nodes and helps them find each other.

roscore

Use roscore command to start the master and create the communication for ROS nodes.

Remember to run roscore first, the master should always be running when you use ROS.


Nodes
Master
Parameter Server
ROS
Topics
Messages
Services
Bags

You are going on a field trip and want to record some sensor data using your Husky. How do you do that?

You would like to replay your experiment in the future, how do you save the whole experiment?

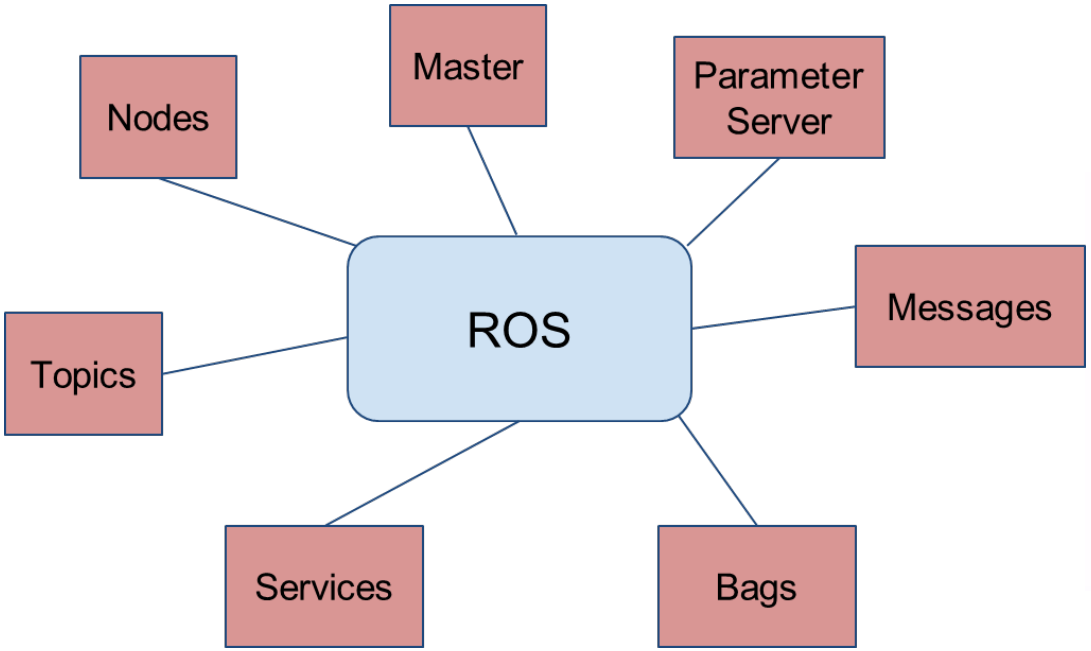
rosbag

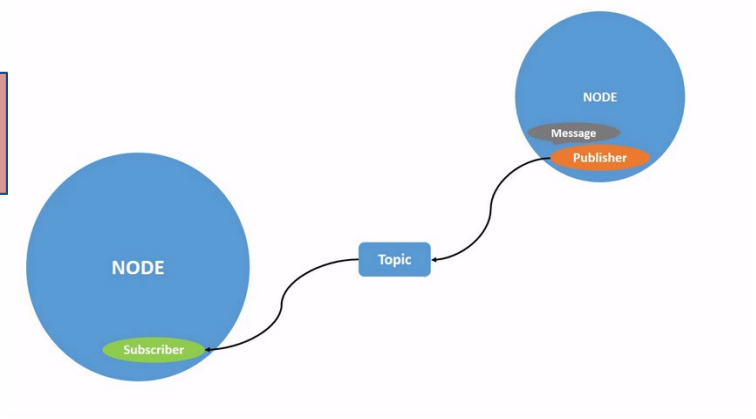
This is a set of tools for recording from and playing back to ROS topics.

```
rosbag record [TOPIC NAME]
rosbag play [bagfile]
rosbag play --clock [bagfile]
```

(publish simulated time synchronized to the messages in the bag file)

So far, we have looked at …





There may be multiple nodes you need to launch and parameters you want to load, to get the Husky up and running. How do you do that in a single command?

- husky_base
- husky_bringup
- husky_control
- husky_description
- husky_desktop
- husky_gazebo
- husky_msgs
- husky_navigation
- husky_robot
- husky_simulator
- husky_viz

roslaunch

roslaunch is a tool for easily launching multiple ROS nodes and setting parameters on the Parameter Server (a shared dictionary which nodes can access to store and retrieve parameters at runtime.).

roslaunch takes in one or more XML configuration files (with the .launch extension) that specify the parameters to set and nodes to launch.

Starting Husky localization and planner without roslaunch

```
$ rosparam set map_file '$(find
husky_navigation)/maps/playpen_map.yaml'

$ rosparam set base_global_planner
“navfn/NavfnROS”

$ rosrun map_server map_server

$ rosrun amcl amcl

$ rosrun move_base move_base
```

Example Launch File

```
<?xml version="1.0"?>

<launch>

  <arg name="map_file" default="$(find husky_navigation)/maps/playpen_map.yaml"/>
  <arg name="base_global_planner" default="navfn/NavfnROS"/>


  <node name="map_server" pkg="map_server" type="map_server" args="$(arg map_file)" />
  <node pkg="amcl" type="amcl" name="amcl">

  <node pkg="move_base" type="move_base" respawn="false" name="move_base" output="screen">
    <param name="base_global_planner" value="$(arg base_global_planner)"/>
  </node>

</launch>
```

$ roslaunch husky_navigation planner.launch

roslaunch

- .launch file:
  - XML format configuration file
  - Can include other launch files
- <param>:
  - Parameters to take in
  - Example:
    - `<param name="publish_frequency" type="double" value="10.0" />`
    - `<param name="configfile" textfile="$(find roslaunch)/example_xml.xml" />`

ROS Environment Variables

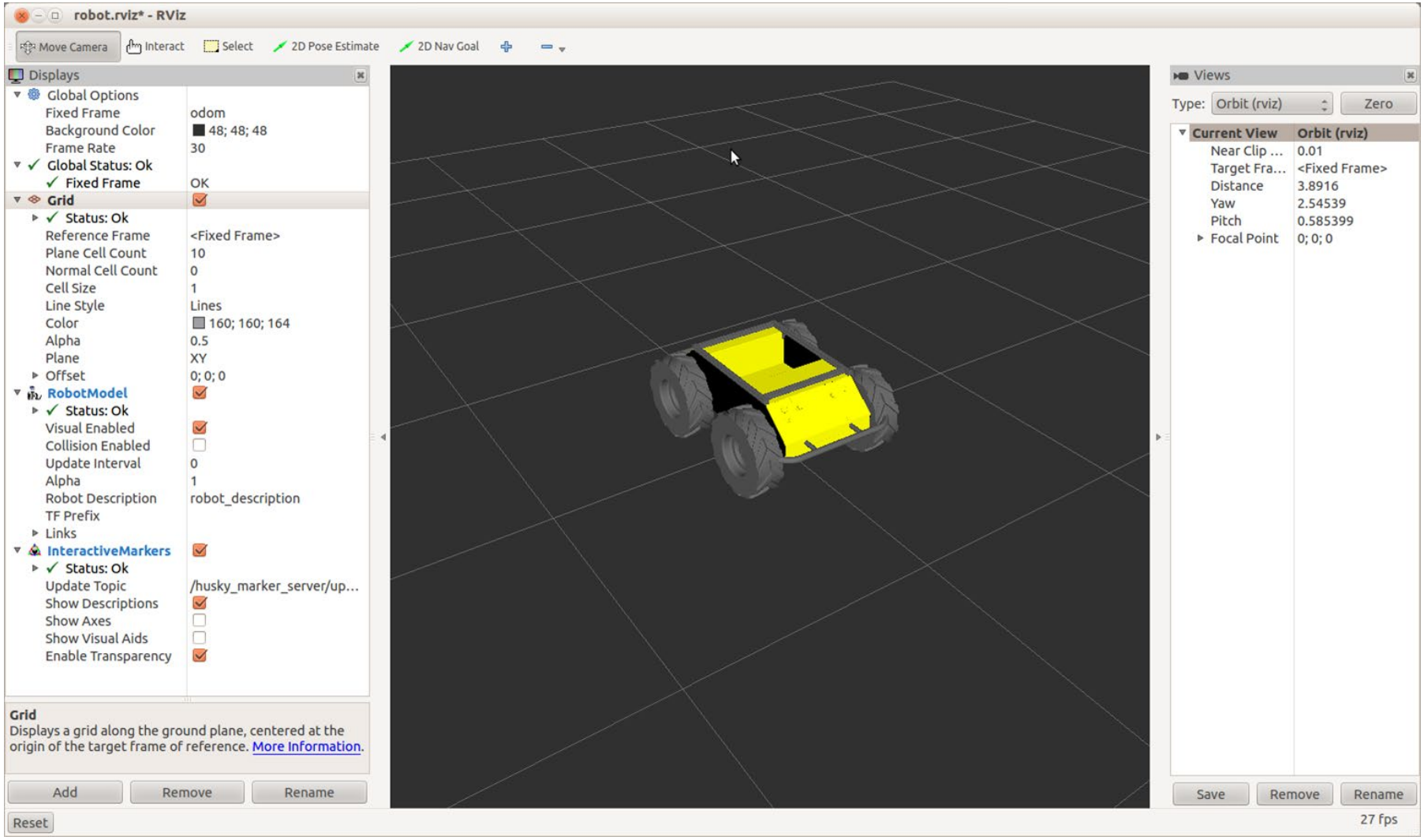
Environment variables are generally parameters used in the operating system to specify the operating environment of the operating system, such as the location of the system folder.

ROS Environment Variables

ROS_ROOT : records the location of ROS core packages
ROS_MASTER_URI : helps nodes to find the master
ROS_PACKAGE_PATH : records the paths of the packages
ROS_IP/ROS_HOSTNAME : the IP address / hostname of a Node or another equipment (maybe another robot)

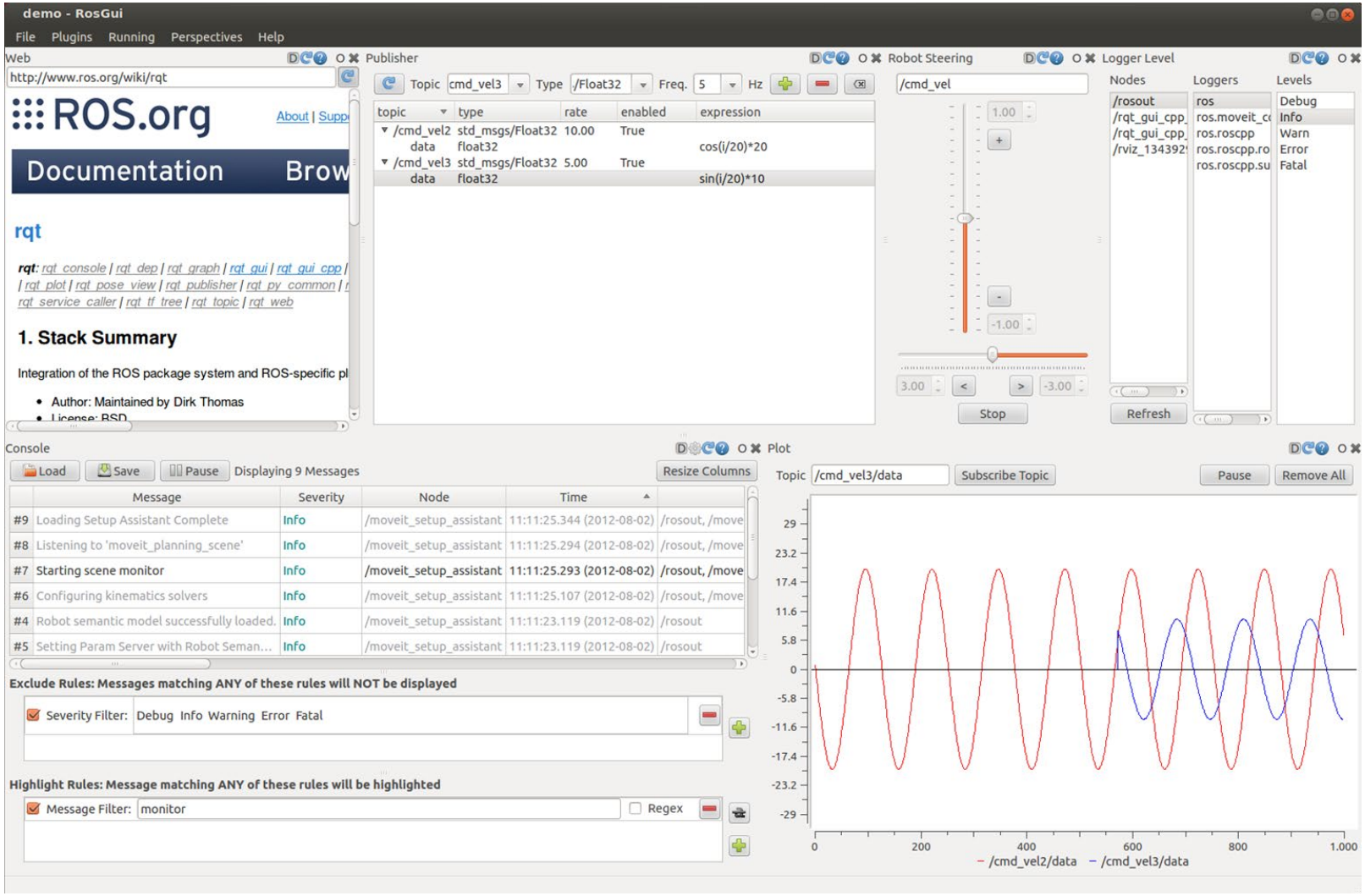
Visualization and GUI tools in ROS

RViz



https://github.com/ros-visualization/rviz

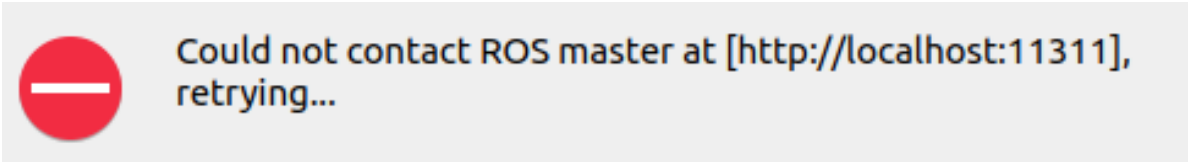
rqt tools



http://wiki.ros.org/rqt_common_plugins

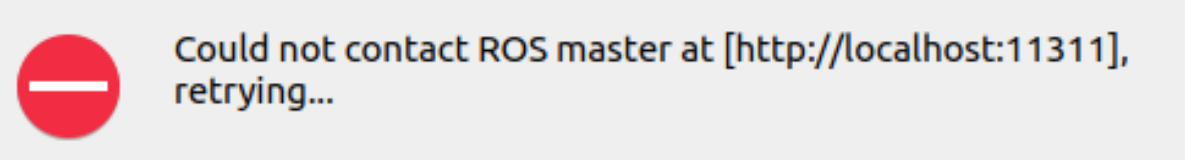
Most common ROS errors

If you see something like:



ROS master is not running

If you see something like:



You may have forgotten to start `roscore`!

Interesting fact: roslaunch will start roscore for you automatically.

Executable does not exist

```
$ rosrun dummy_package hello_world
[rosrun] Couldn't find executable named hello_world
below /home/hw_ws/src/dummy_package
[rosrun] Found the following, but they're either not
files,
[rosrun] or not executable:
[rosrun]
/home/hw_ws/src/dummy_package/scripts/hello_world
```

Executable does not exist

```
$ rosrun dummy_package hello_world
[rosrun] Couldn't find executable named hello_world
below /home/hw_ws/src/dummy_package
[rosrun] Found the following, but they're either not
files,
[rosrun] or not executable:
[rosrun]
/home/hw_ws/src/dummy_package/scripts/hello_world
```

Solution

use `$ chmod u+x hello_world`

Forgetting to source setup.bash!

After catkin build, remember to `source devel/setup.bash` or expect an error like this one

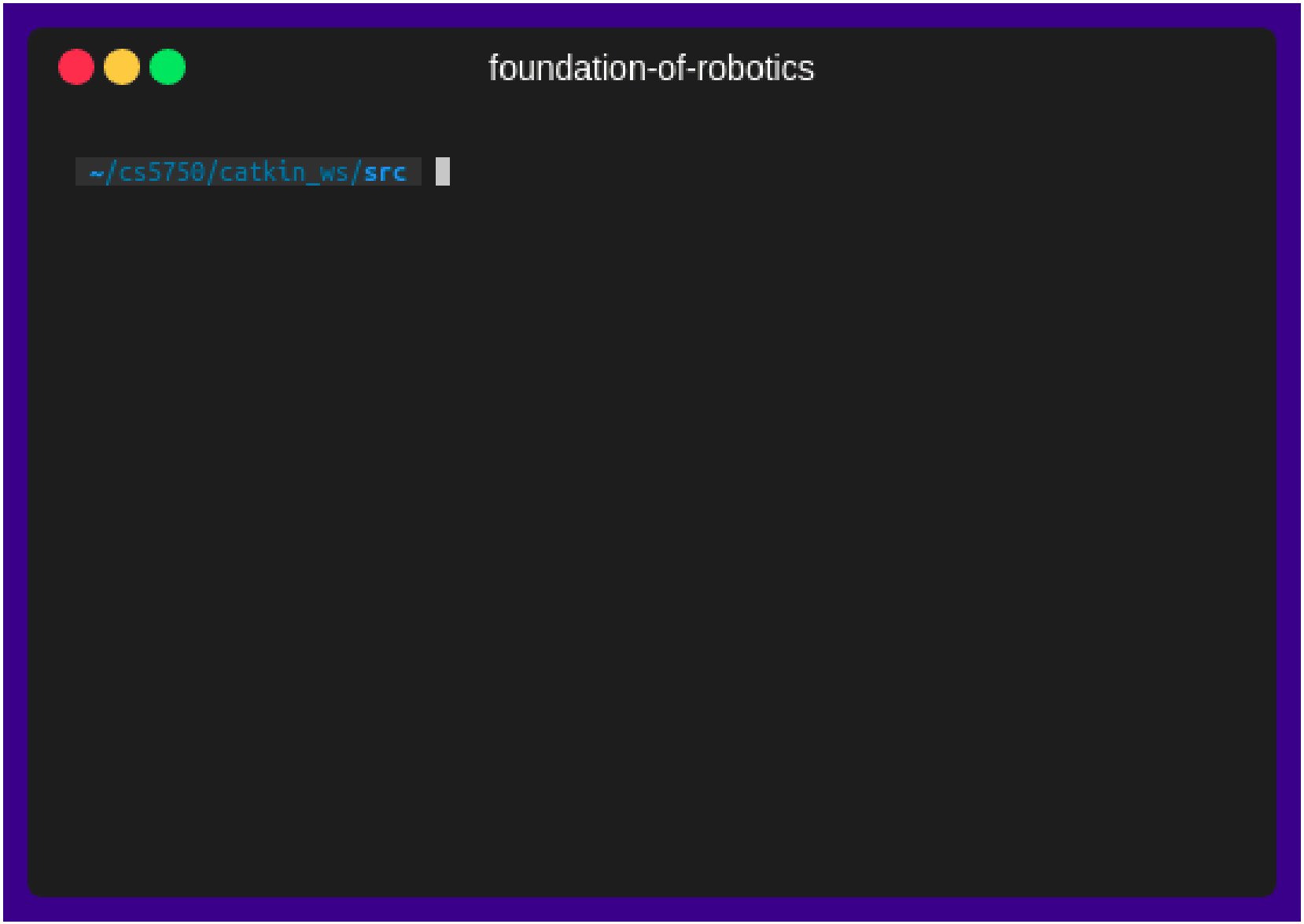
```
$ roslaunch dummy_package start.launch
RLException: [start.launch] is neither a launch file in package [dummy_package] nor is [dummy_package] a launch file name
The traceback for the exception was written to the log file
```

Tab completion will also not work since the package cannot be found.

catkin build in the root

Remember to `catkin build` in the correct path.

The first time you build a workspace, it is possible to build in package folder, but it won't work ☺


foundation-of-robotics
~/cs5750/catkin_ws/src

rosrun vs roslaunch

`roslaunch` works with launch files.
`rosrun` works with executables.

```
$ rosrun dummy_package start.launch
[rosrun] Couldn't find executable named start.launch
below /home/hw_ws/src/cs4750-hw-
solutions/dummy_package
[rosrun] Found the following, but they're either not
files,
[rosrun] or not executable:
[rosrun]
/home/hw_ws/src/dummy_package/launch/start.launch
```

dependencies missing? rosdep to help!

If you run `catkin build` with missing dependencies (e.g. librealsense)

```
Errors << introduction:check /home/for/hw_ws/logs/introduction/build.check.000.log
CMake Error at /opt/ros/noetic/share/catkin/cmake/catkinConfig.cmake:83 (find_package):
  Could not find a package configuration file provided by "librealsense" with
  any of the following names:

            librealsenseConfig.cmake
            librealsense-config.cmake

  Add the installation prefix of "librealsense" to CMAKE_PREFIX_PATH or set
  "librealsense_DIR" to a directory containing one of the above files.  If
  "librealsense" provides a separate development package or SDK, be sure it
  has been installed.
Call Stack (most recent call first):
  CMakeLists.txt:4 (find_package)
```

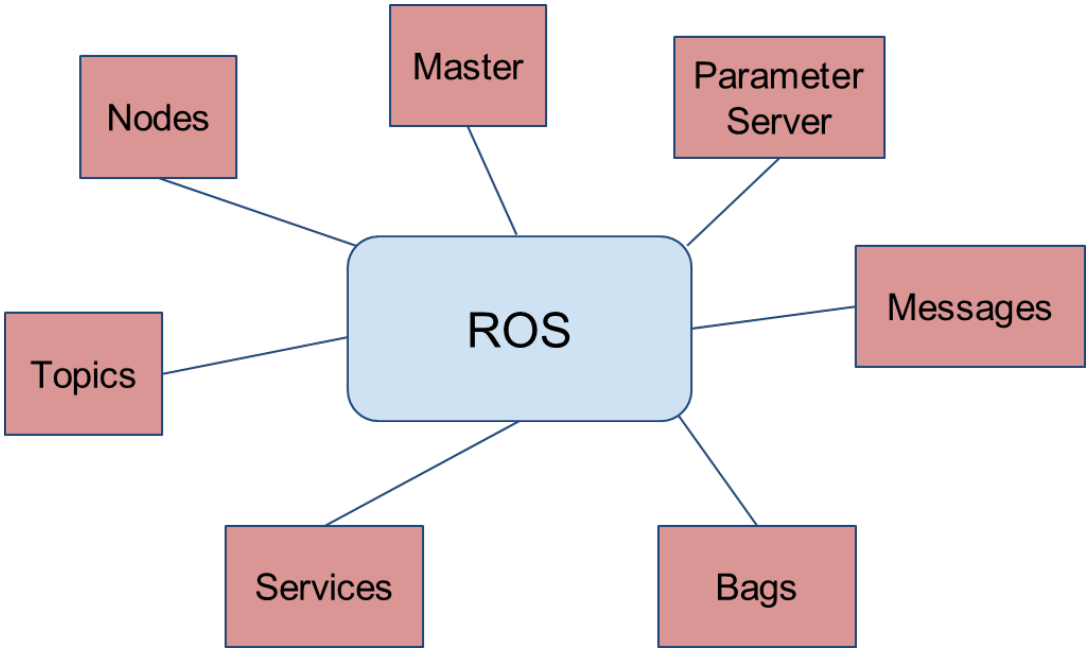
Use `rosdep install --from-paths src --ignore-src -r -y`

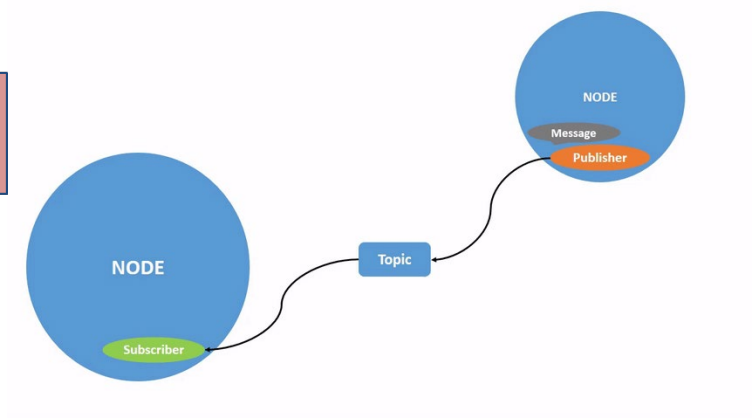
This should install all missing dependencies for all packages in the workspace.

Best Practices

https://github.com/leggedrobotics/ros_best_practices/wiki

Revisiting ROS





Quick Hands-On Practice

Follow https://tinyurl.com/ros-catkin-ws to create a workspace.

Note: The tutorial uses `catkin_make` for building the workspace. We recommend using `catkin build` instead.

Follow https://tinyurl.com/ros-package to create a package.

Quick Hands-On Practice

We will be doing a walkthrough of this ROS tutorial on writing a simple publisher and subscriber in Python.

Hopefully you have the catkin_ws and the beginner_tutorials package in it by now!

Now let's try to implement a simple publisher and subscriber.

```
$ roscd beginner_tutorials
$ mkdir scripts
$ cd scripts
```

Now let's go ahead and implement the publisher node.
In `scripts` folder, …

Fetch the file using the following command line on your terminal:

```
$ wget
https://raw.github.com/ros/ros_tutorials
/kinetic-
devel/rospy_tutorials/001_talker_listene
r/talker.py
```

Convert to executable

```
$ chmod +x talker.py
```

Now, let's create the subscriber node.

In `scripts` folder, create another file called listener.py

Fetch the file using the following command line on your terminal:

```
$ wget
https://raw.github.com/ros/ros_tutorials
/kinetic-
devel/rospy_tutorials/001_talker_listene
r/listener.py
```

Convert to executable

```
$ chmod +x listener.py
```

Running the publisher and subscriber

In terminal 1,

```
$ roscore
```

In terminal 2,

```
$ cd ~/catkin_ws
$ source devel/setup.bash OR
$ . devel/setup.bash
```

Running the publisher and subscriber

In terminal 2,

```
$ rosrun beginner_tutorials talker.py
```

should output:

```
[INFO] [WallTime: 1314931831.774057] hello world 1314931831.77
[INFO] [WallTime: 1314931832.775497] hello world 1314931832.77
[INFO] [WallTime: 1314931833.778937] hello world 1314931833.78
[INFO] [WallTime: 1314931834.782059] hello world 1314931834.78
```

Running the publisher and subscriber

In terminal 3 (after sourcing),

```
$ rosrun beginner_tutorials listener.py
```

should output:

```
[INFO] [WallTime: 1314931969.258941]
/listener_17657_1314931968795I heard hello world 1314931969.26
[INFO] [WallTime: 1314931970.262246]
/listener_17657_1314931968795I heard hello world 1314931970.26
[INFO] [WallTime: 1314931971.266348]
/listener_17657_1314931968795I heard hello world 1314931971.26
```

talker.py

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
         pass
```

talker.py

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
         pass
```

talker.py

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
         pass
```

talker.py

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
         pass
```

talker.py

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
         pass
```

talker.py

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
         pass
```

talker.py

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
         pass
```

talker.py

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
         pass
```

talker.py

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
         pass
```

talker.py

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
         pass
```

talker.py

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

listener.py

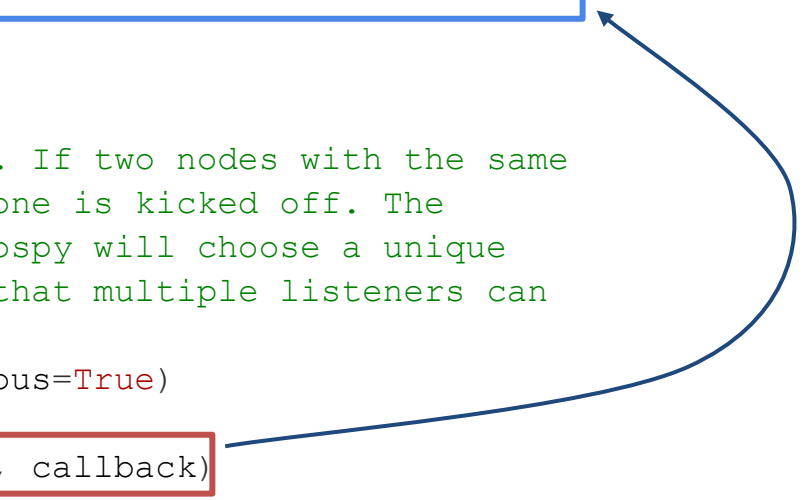
```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
     listener()
```

listener.py

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
      listener()
```

listener.py

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
      listener()
```



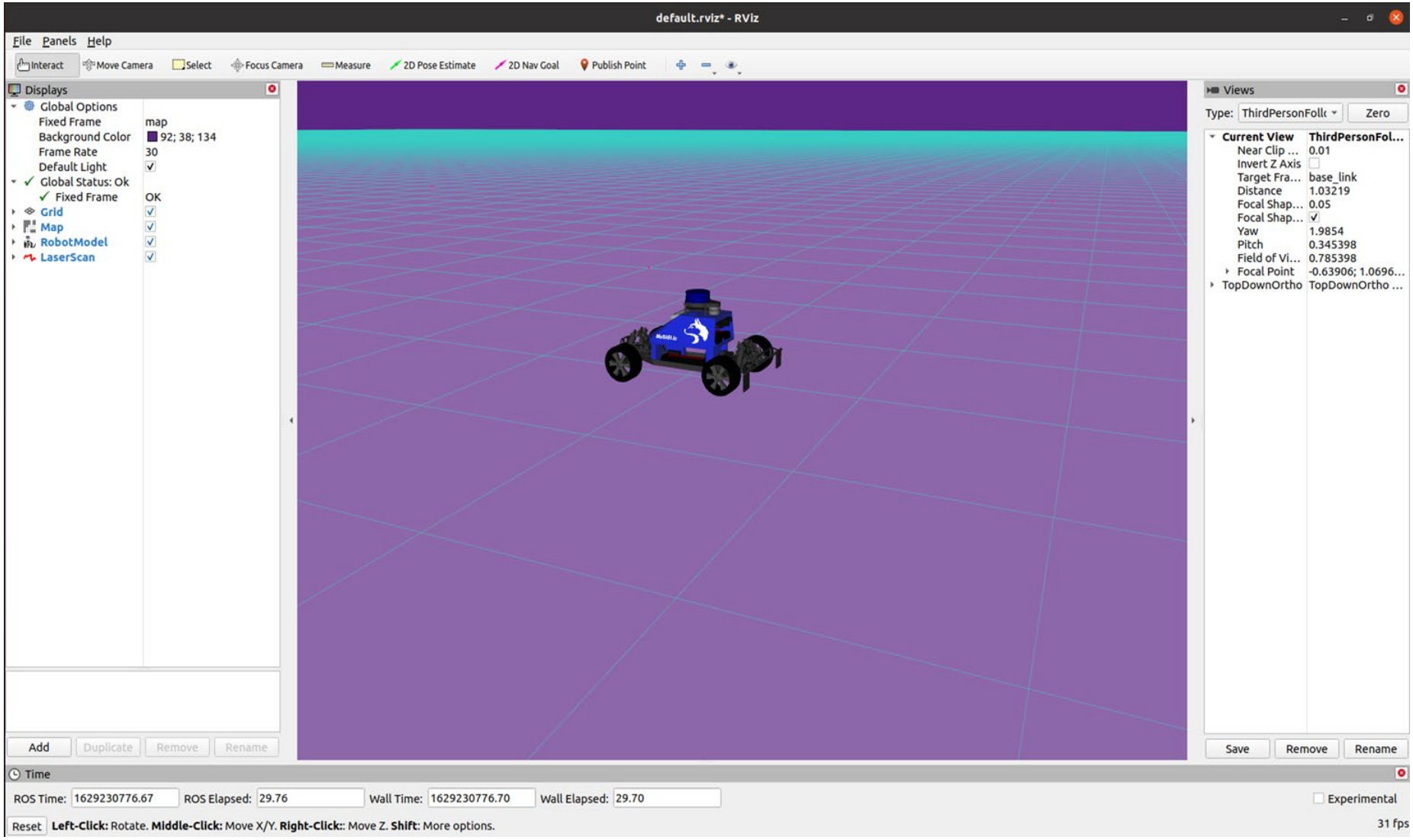
listener.py

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
      listener()
```

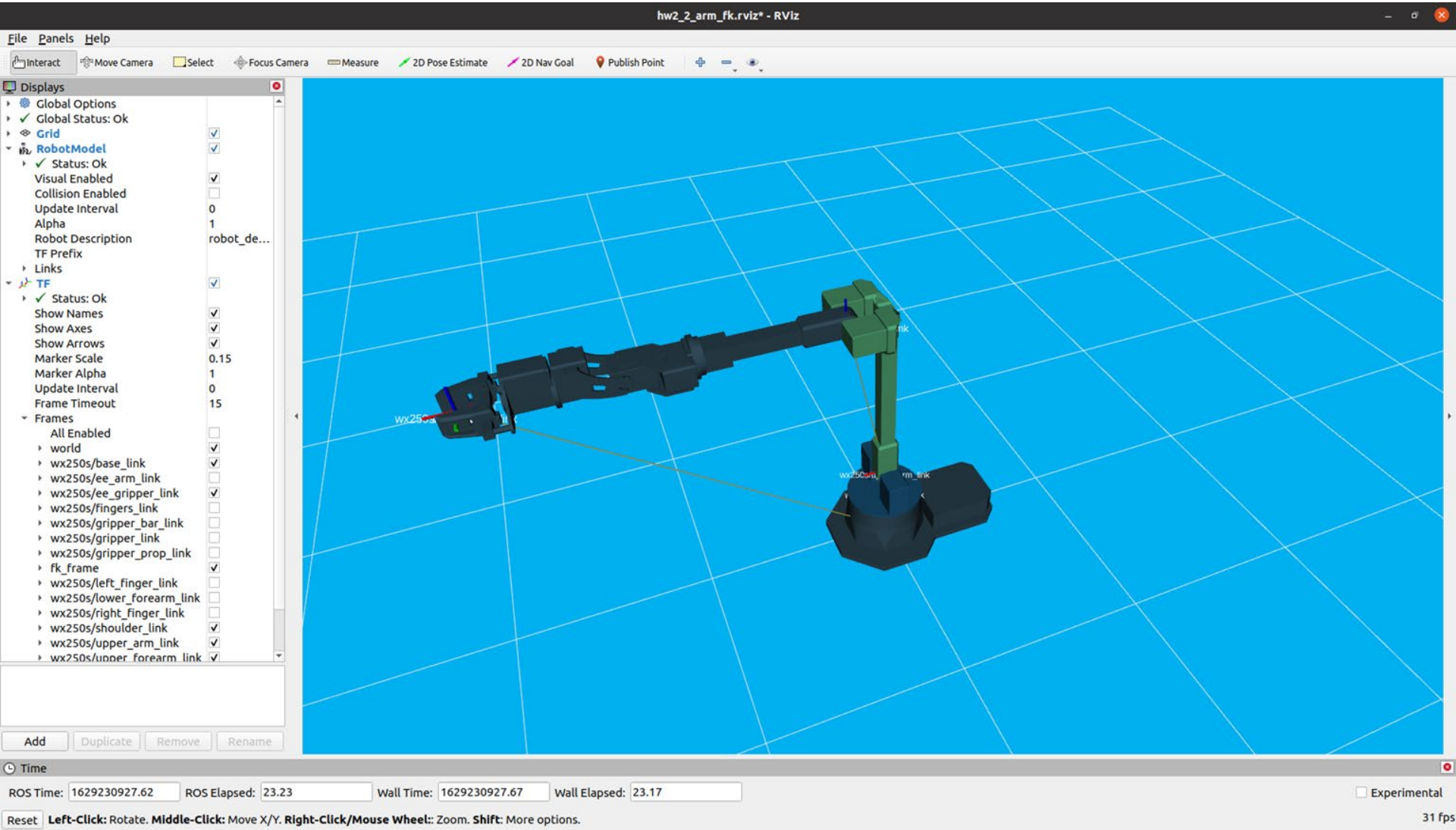
listener.py

```
#!/usr/bin/env python
import rospy
from std_msgs.msg import String

def callback(data):
    rospy.loginfo(rospy.get_caller_id() + "I heard %s", data.data)

def listener():

    # In ROS, nodes are uniquely named. If two nodes with the same
    # name are launched, the previous one is kicked off. The
    # anonymous=True flag means that rospy will choose a unique
    # name for our 'listener' node so that multiple listeners can
    # run simultaneously.
    rospy.init_node('listener', anonymous=True)

    rospy.Subscriber("chatter", String, callback)

    # spin() simply keeps python from exiting until this node is stopped
    rospy.spin()

if __name__ == '__main__':
    listener()
```

MuSHR car environment



WidowX 250 arm environment



Additional resources

- ROS cheatsheet
- ROS answers forum
- awesome-ros repo

Takeaways

- In this class, we learned...
  - ROS workspace and how to create one
  - ROS package
  - Basic ROS components
    - Node, Topics, Services, Bags, Messages, Parameter Server, Master
  - Using Publisher/Subscriber for communication
  - Visualization and GUI tools in ROS