# ECE 478-578
# Intelligent Robotics I

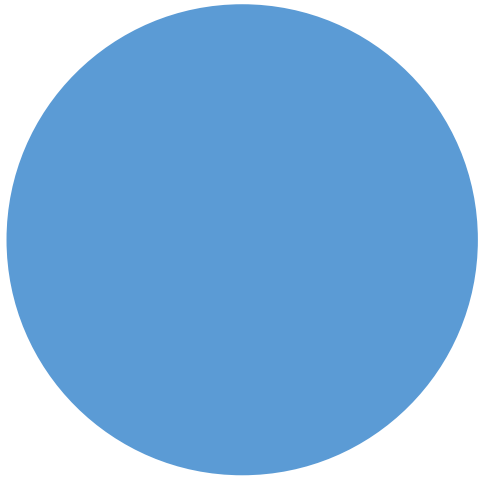
**PhD. Husnu Melih Erdogan – Electrical & Computer Engineering**

herdogan@pdx.edu **Teaching Assistant**

Portland State
UNIVERSITY

# Introduction to ROS Part - 1


Portland State
UNIVERSITY



# ROS
# (Robot Operating system)

Portland State
UNIVERSITY

# Course Structure

- **Part 1 - Overview**
  - What is ROS?
  - Introduction to ROS
  - ROS architecture, philosophy, history
  - How to install ROS?
  - Examples
  - Installation
  - ROS Master
  - ROS Nodes
  - ROS Topic
  - ROS Messages
  - Console Commands
  - ROS Packages
  - ROS Launch-files
  - Catkin Workspace and Build System
  - Turtlesim

  - Assignment 3 – Part1

- **Part 2 - Details**
  - ROS File System
  - ROS Package
  - How to create a package?
  - How to build a package?
  - Creating a Publisher Node
  - Creating a Subscriber Node
  - Creating a Launch File
  - How to use ROS .bagfiles?

  - Assignment 3- Part2

- **Part 3 - Details**
  - ROS Parameters
  - ROS Messages
  - ROS Services
  - ROS Actions

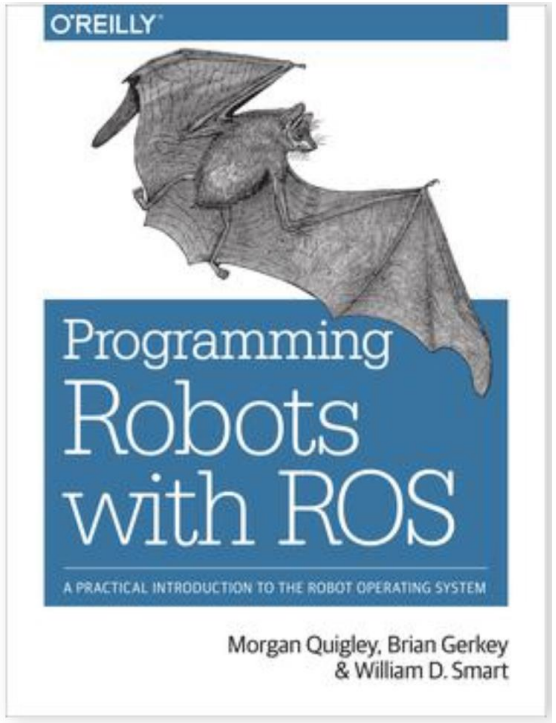
  - Assignment 4

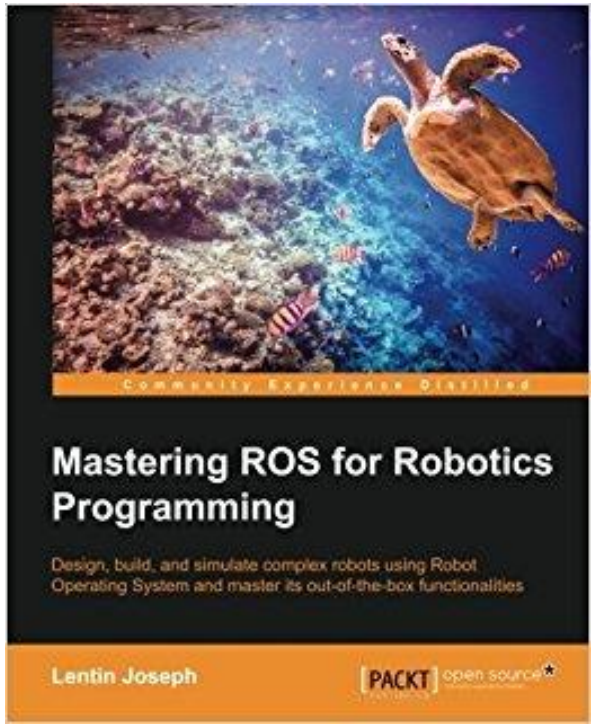
- **Part 4 - Project**
  - Rviz
  - Mapping
  - Localization
  - 2D Multi-Robot Simulator

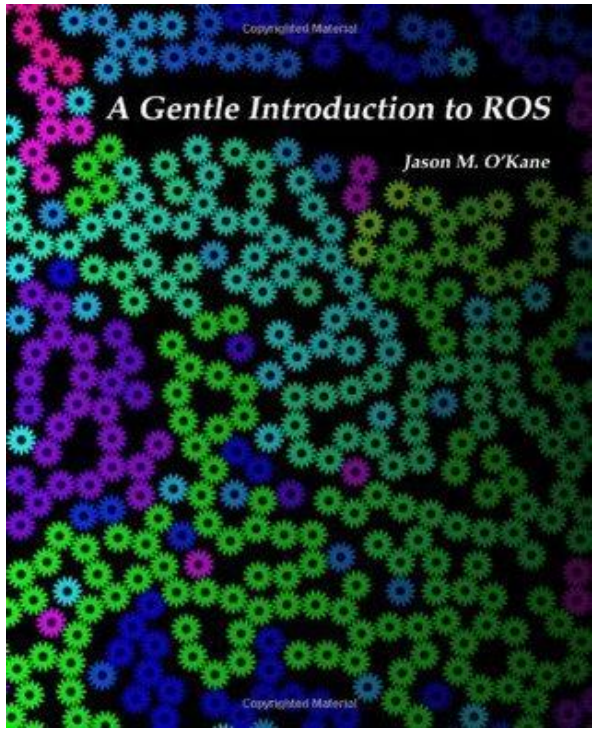
  - Assignment 5

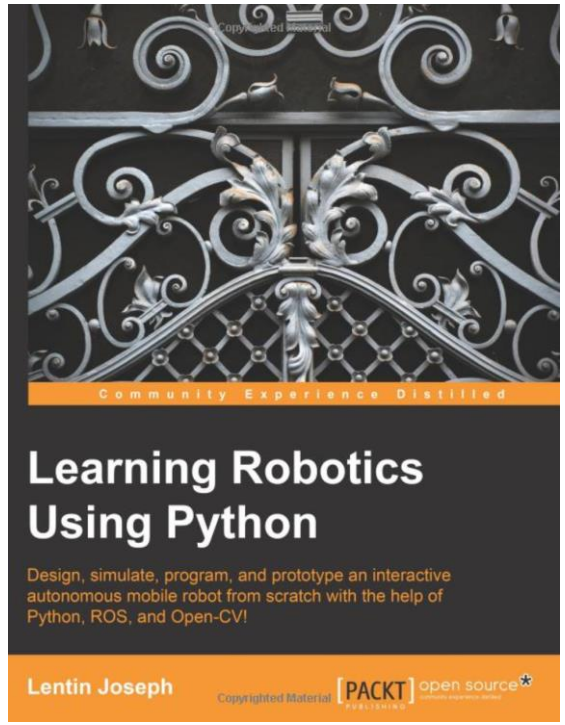
Portland State UNIVERSITY

# Books




Portland State
UNIVERSITY

# Courses

- http://wiki.ros.org/Courses

3. University (Undergraduate & Graduate) Courses

- International ROS Summer School (University of Applied Sciences in Aachen, Germany)
- 240AR060: Introduction to ROS (IRI-UPC) (Barcelona) ( IRI - UPC)
- Programming for Robotics – Introduction to ROS (ETH Zurich) ( Video lectures)
- Gaitech Education Portal, China
- Introduction to Robotics (CS460) (Prince Sultan University, Saudi Arabia)
- Introduction to Robotics (Bar-Ilan University, Israel)
- Advanced Robotics (CU Boulder)
- Advanced Robotics Systems (KU Leuven)
- Autonomous robotics (DEI - Università di Padova)
- Autonomous Vehicles - Freshman Research Initiative (University of Texas at Austin)
- CSCE 574: Robotics (University of South Carolina)
- CSE553: Mobile Robotics (Washington University in St Louis)
- CS1480: Building Intelligent Robots (Brown University)
- CS225B: Robot Programming Laboratory (Stanford University)
- CS324: Perception for Robotics (Stanford University, 2011)
- CS324: Perception for Manipulation (Stanford University, 2010)
- MEAM620: Robotics (University of Pennsylvania)
- CS445/660: Social Robotics Seminar (University at Albany)
- GEI740 - "Programmation de robots mobiles" - Behavior-based mobile navigation with Gazebo (Université de Sherbrooke)
- RTM-ROS Robotics in Japanese (Tokyo University)
- Robot Learning (Cornell University)
- Robot Programming (Sapienza University of Rome)
- Intelligent Robotics (University of Birmingham)
- RobotChallenge (Leibniz Universität Hannover)
- CSIS401: Introduction to Robotics (Siena College, Loudonville NY)
- ME 495: Embedded Systems in Robotics (Northwestern University, Evanston IL)
- Intelligent Robotics (Kyushu University, Japan)
- Programming for the Robot Operating System (University of Zagreb, Croatia)

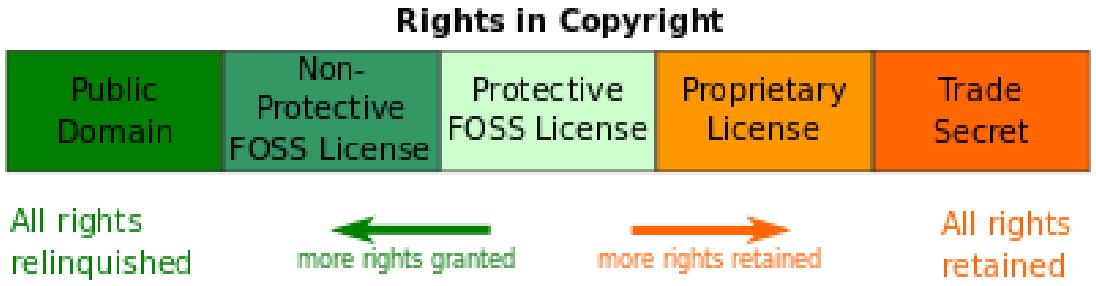
Portland State UNIVERSITY

# What is ROS?

- ROS stands for Robot Operating System.
- However, it is not a real operating system such a Linux, Windows, OSX
- It is framework.
- It is more like a middleware that sits between your operating system and your program in C++, Python, Java, and MATLAB etc.

# Why ROS is useful?

- A distributed, modular design
- A very large community
- Permissive Licensing
  - "lets people do anything they want with your code as long as they provide attribution back to you and don’t hold you liable."
- Collaborative Environment
- It is free.
- Supports different programming languages
  - Matlap, C++, Python, Lisp
- An important technical skill to have in your resume.





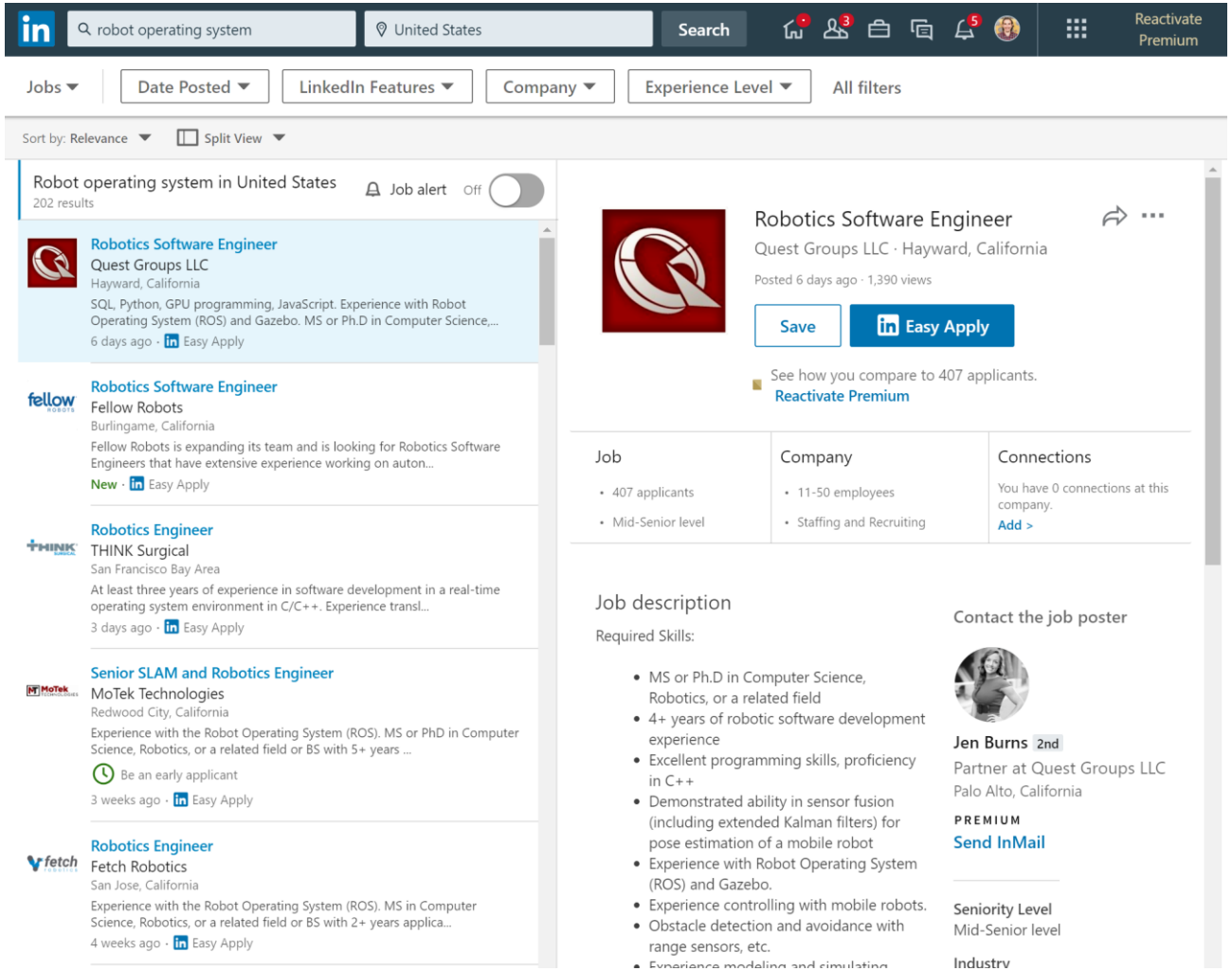
Portland State UNIVERSITY

# Why ROS is good for us?

- We use ROS in the Intelligent Robotics Lab because:
  - Some of our robots already supports ROS
  - ROS is a distributed system
  - It allows us do fast prototyping
  - There are already so many ready to use packages
  - Robot Theater project
  - It will also help you find Robotics Engineering jobs

Portland State
UNIVERSITY

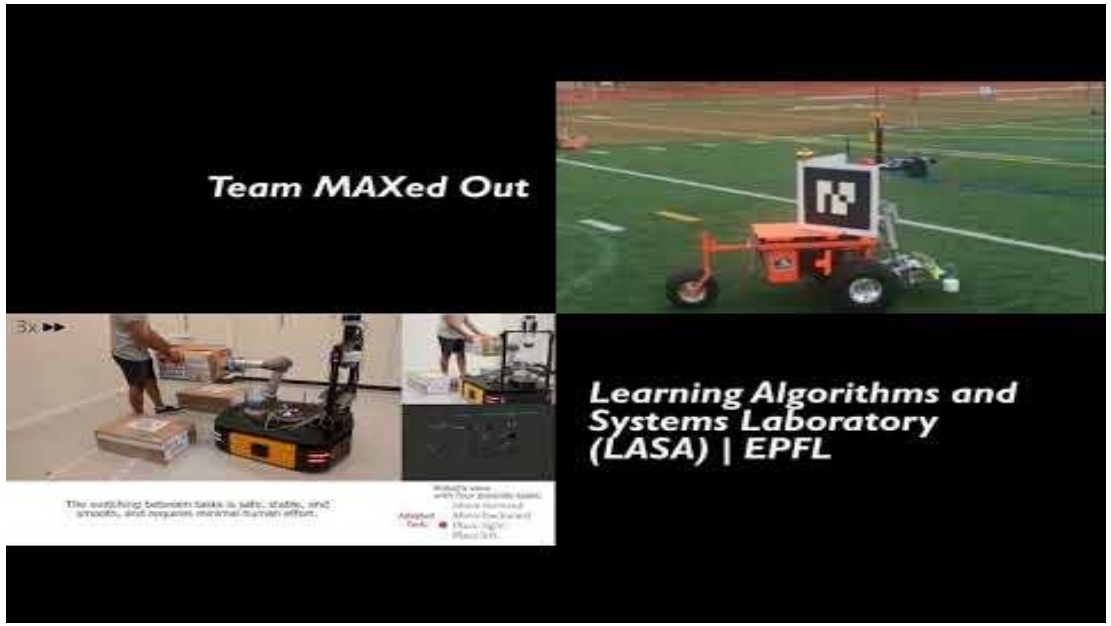
# Robotics Engineering Jobs




Portland State UNIVERSITY

# History of ROS

- It is developed in 2007 at the Stanford Artificial Intelligence Lab.
- It was managed by Willow Garage.
- Open Source Robotics Foundations has been managing it since 2013.




Portland State UNIVERSITY

# Current Distributions

## Install


Get ROS Indigo Igloo on Ubuntu Linux

Install

Donate to ROS


Get ROS Kinetic Kame on Ubuntu Linux

(Recommended for Latest LTS)

Install

Donate to ROS

For more options, consult the installation guide.


Get ROS Lunar Loggerhead on Ubuntu Linux

(Recommended for Latest)

Install

Donate to ROS

ROS Melodic Morenia

Released May, 2018

Latest LTS, supported until May, 2023


Portland State UNIVERSITY

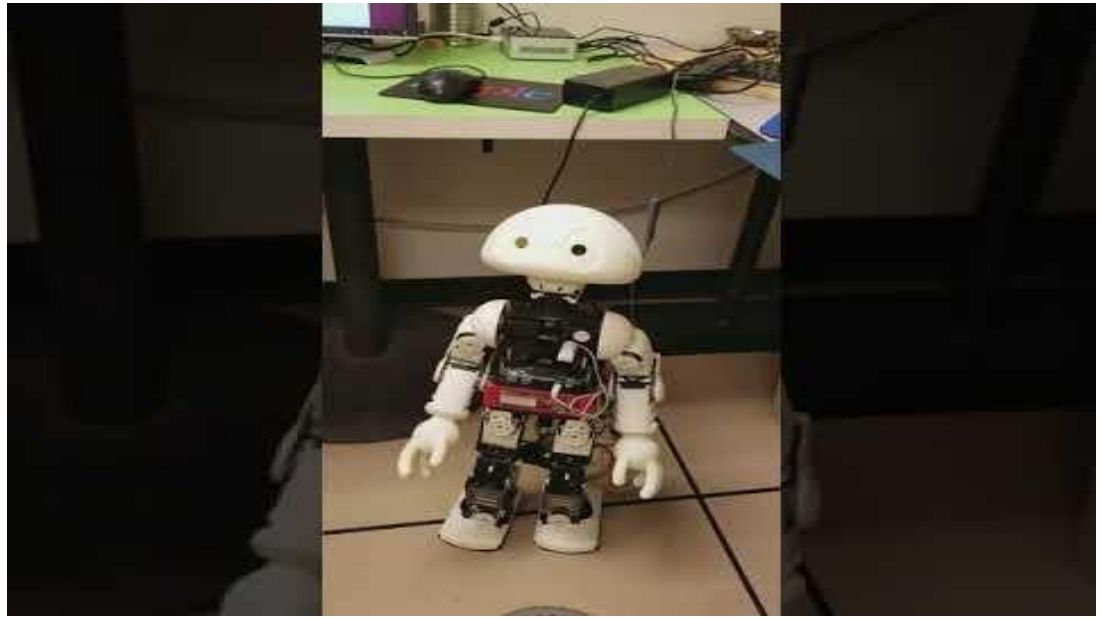
# ROS Industrial

ROS-Industrial is an open-source project that extends the advanced capabilities of ROS software to manufacturing.


http://rosindustrial.org/

Portland State UNIVERSITY

# Some of Our ROS Powered Robots



Portland State
UNIVERSITY

# ROS Philosophy

- **Peer to Peer**
  - Individual programs communicate over defined API (ROS messages, services)
- **Distributed**
  - Programs can be run on multiple computers and communicate over the network.
- **Supports Multiple Languages**
  - ROS modules can be written in any language for which a client library exist (C++, Python, MATLAB, Java, etc.)
- **Light-weight**
  - Stand-alone libraries are wrapped around with in a thin ROS layer.
  - It doesn't mean ROS doesn't use your CPU or don't need a powerful PC.
  - Some packages could use all almost your system resources.
- **Free and Open-Source**
  - It is totally free and supported by a considerably large community.
  - It is backed by many universities and industrial companies.
  - It becomes a standard in the robotics field.

# ROS Core Components

- There are three core components:

**Communications Infrastructure**:

- Message Passing
- Remote Procedure Calls
- Recording and Playback of Messages
- Distributed Parameter System

**Tools:**

- Command-Line Tools
- Rviz
- Rqt

**Robot-Specific Features:**

- Standard Message Definitions
- Robot Geometry Library
- Robot Description Language
- Localization
- Mapping
- Navigation

# Communications Infrastructure

- At the lowest level, ROS offers a message passing interface that provides inter-process communication and is commonly referred to as a middleware.

- The ROS middleware provides these facilities:
  - publish/subscribe anonymous message passing
  - recording and playback of messages
  - request/response remote procedure calls
  - distributed parameter system

# Communications Infrastructure
# Message Passing

- A communication system is often one of the first needs to arise when implementing a new robot application.
- ROS's built-in and well-tested messaging system saves you time by managing the details of communication between **distributed nodes via the anonymous publish/subscribe mechanism**.
- Another benefit of using a message passing system is that it forces you to implement clear interfaces between the nodes in your system, thereby improving encapsulation and promoting code reuse.
- The structure of these message interfaces is defined in the message IDL (Interface Description Language).

# Communications Infrastructure
# Remote Procedure Calls

- The asynchronous nature of publish/subscribe messaging works for many communication needs in robotics, but sometimes you want synchronous request/response interactions between processes.
- The ROS middleware provides this capability using **services**.
  Like topics, the data being sent between processes in a service call are defined with the same simple message IDL (Interface Description Language).

# Communications Infrastructure: Recording and Playback of Messages

- Because the publish/subscribe system is anonymous and asynchronous, the data can be easily captured and replayed without any changes to code.
- Say you have Task A that reads data from a sensor, and you are developing Task B that processes the data produced by Task A**. ROS makes it easy to capture the data published by Task A to a file, and then republish that data from the file at a later time.**
- This is a powerful design pattern that can significantly reduce your development effort and promote flexibility and modularity in your system.
- We can run the robot itself only a few times, recording the topics we care about, and then replay the messages on those topics many times, experimenting with the software that processes those data.

Portland State UNIVERSITY

# Communications Infrastructure
# Distributed Parameter System

- The ROS middleware also provides a way for tasks to share configuration information through a global key-value store.
- This system allows you to easily modify your task settings, and even allows tasks to change the configuration of other tasks.

# Tools

- One of the strongest features of ROS is the powerful development toolset.
- These tools support introspecting, debugging, plotting, and visualizing the state of the system being developed.
- The underlying publish/subscribe mechanism allows you to spontaneously do self-analysis of the data flowing through the system, making it easy to comprehend and debug issues as they occur.
- The ROS tools take advantage of this introspection capability through an extensive collection of graphical and command line utilities that simplify development and debugging.

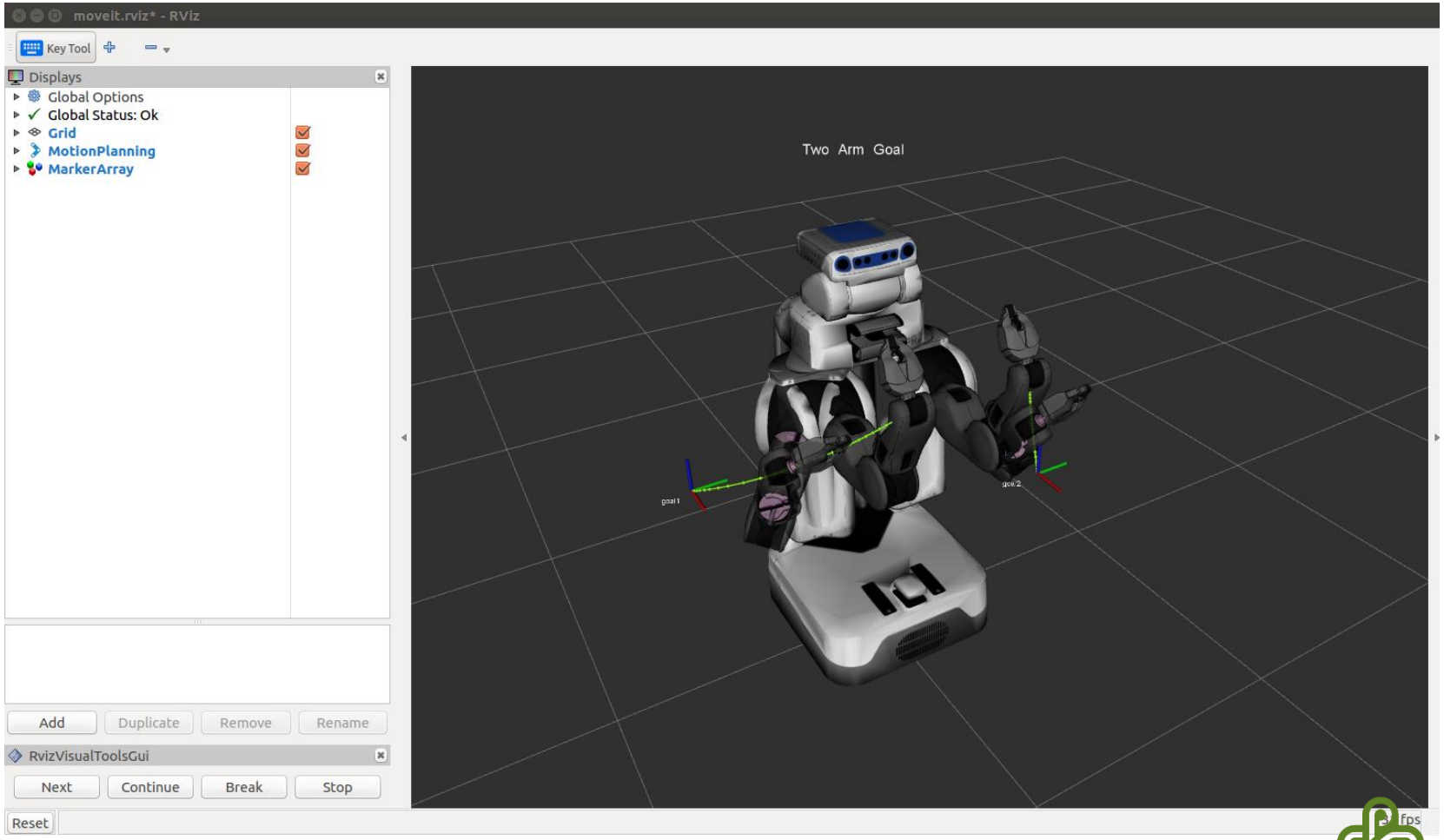
# Tools - Command-Line Tools

- ROS can be used 100% without a GUI.
- All core functionality and introspection tools are accessible via one of more than 45 ROS command line tools.
- There are commands for launching groups of nodes;
  - introspecting topics,
  - services,
- and actions;
  - recording and playing back data;
  - and a host of other situations.

# Tools - rviz

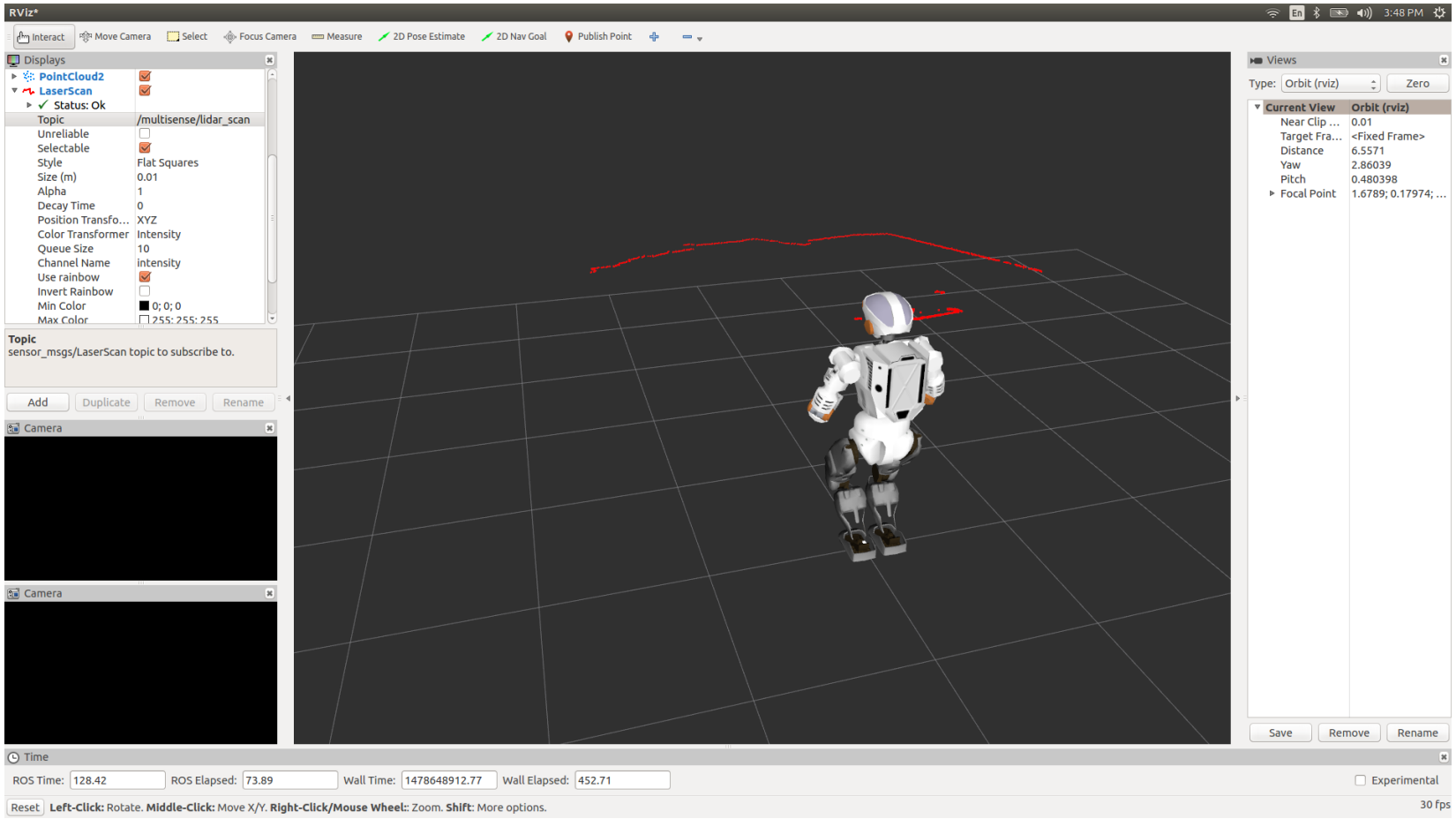
- Perhaps the most well-known tool in ROS, rviz provides general purpose, three-dimensional visualization of many sensor data types and any URDF-described robot.
- rviz can visualize many of the common message types provided in ROS, such as
  - laser scans,
  - three-dimensional point clouds,
  - and camera images.
- Visualizing all of your data in the same application not only looks impressive, but also allows you to quickly see what your robot sees, and identify problems such as sensor misalignments or robot model inaccuracies.

# rviz



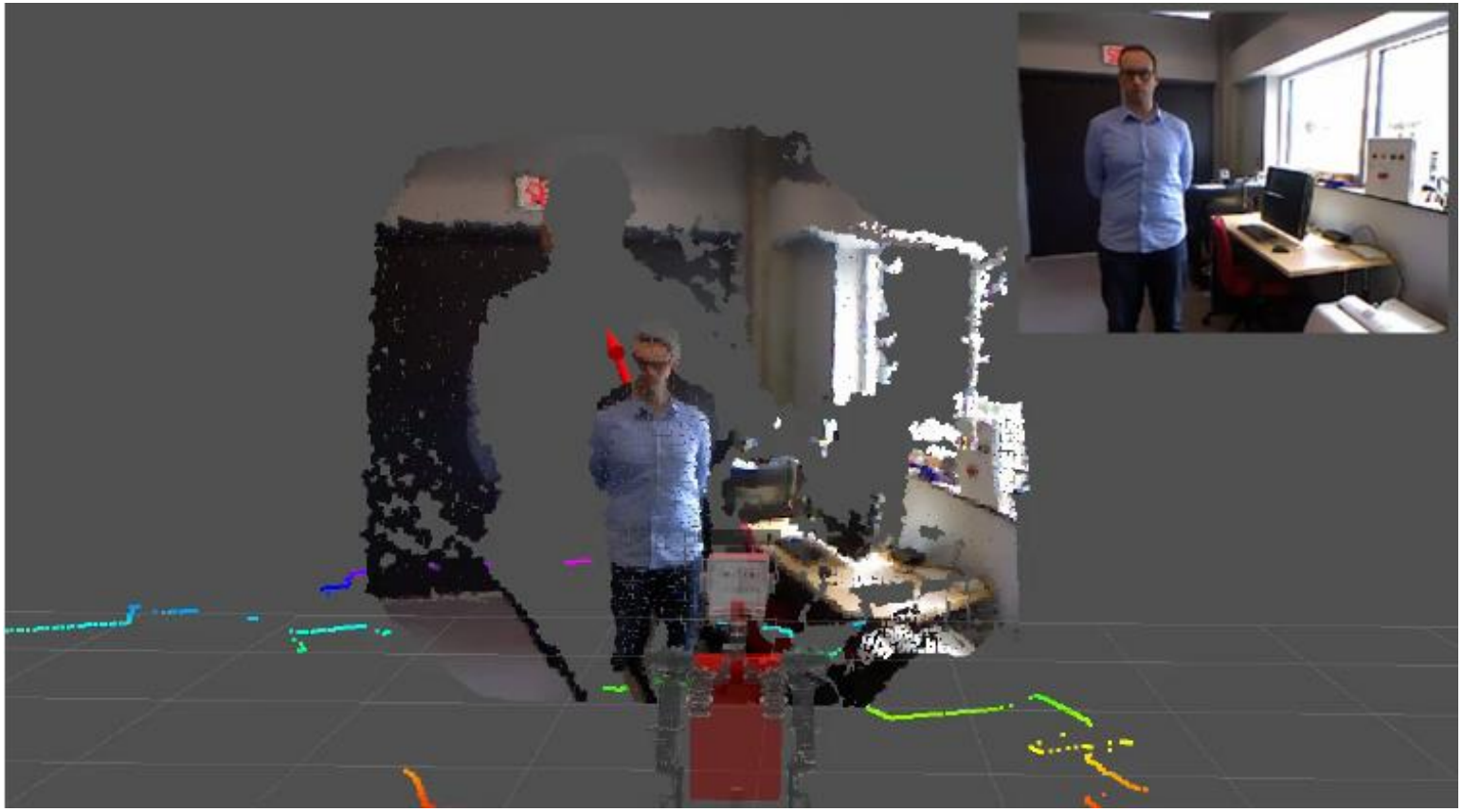

Portland State UNIVERSITY

# rviz




Portland State UNIVERSITY

# rviz

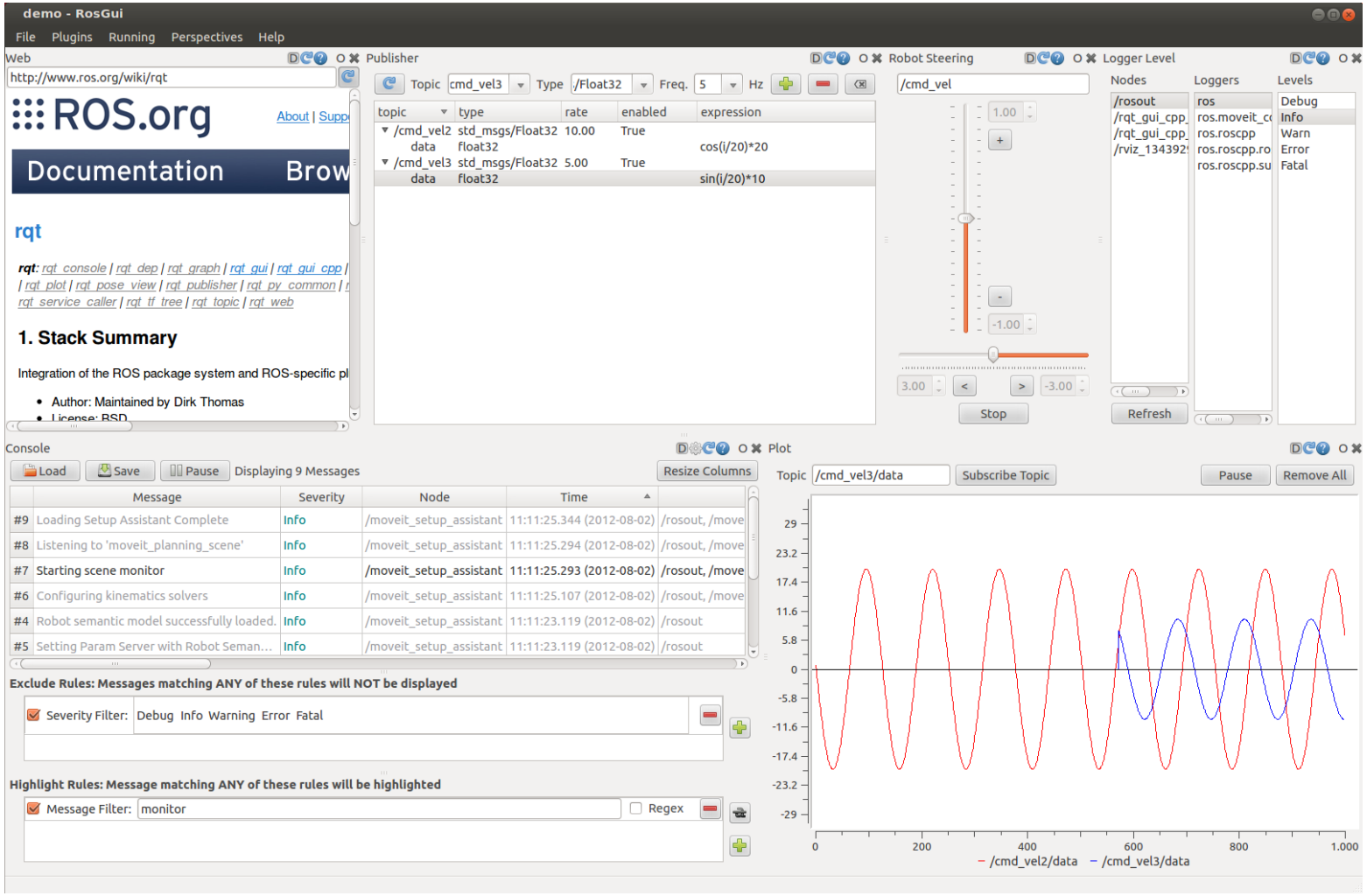


Portland State
UNIVERSITY

# Tools - rqt

- ROS provides rqt, a Qt-based framework for developing graphical interfaces for your robot.
- You can create custom interfaces by composing and configuring the extensive library of built-in rqt plugins into tabbed, split-screen, and other layouts.
- You can also introduce new interface components by writing your own rqt plugins .

# Tools - rqt

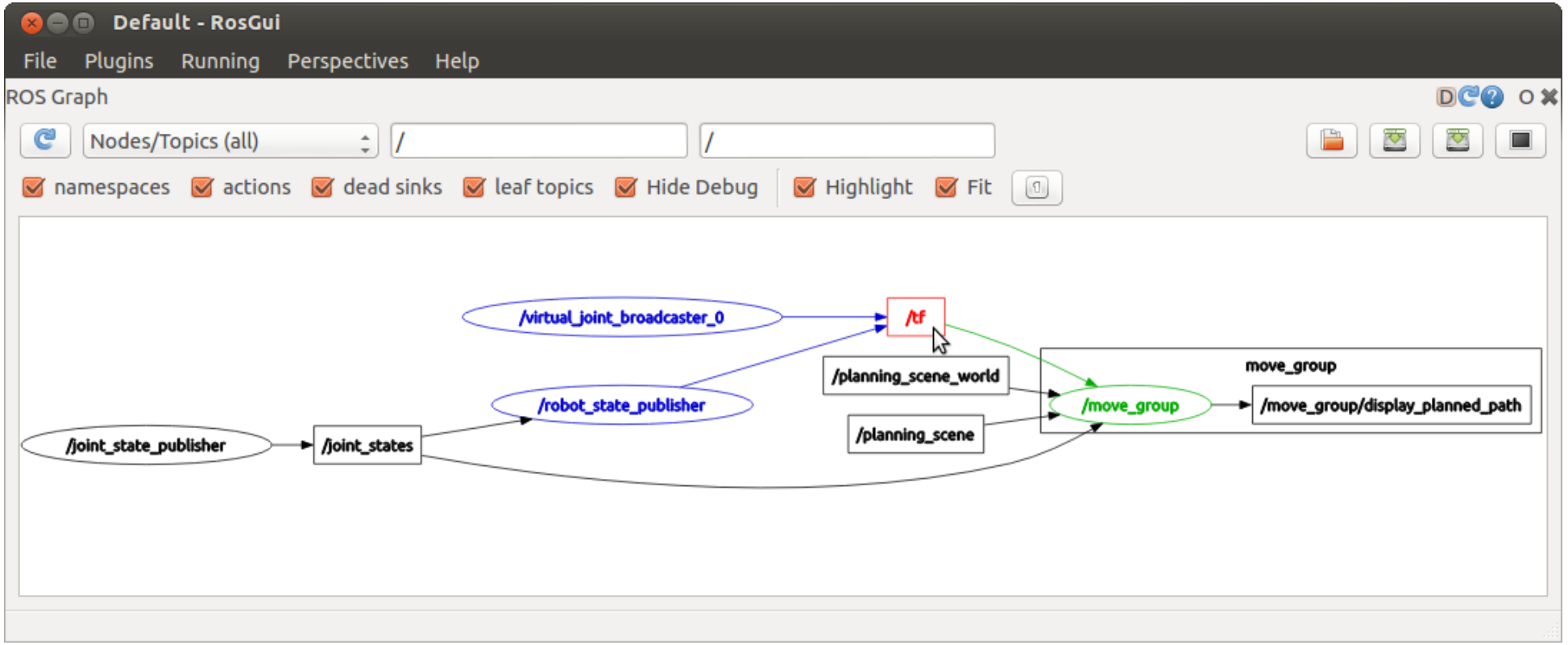



Portland State UNIVERSITY

# Tools – rqt_graph

- The rqt_graph plugin provides introspection and visualization of a live ROS system
- It shows nodes and the connections between them
- It allows you to easily debug and understand your running system and how it is structured.

# Tools – rqt_graph

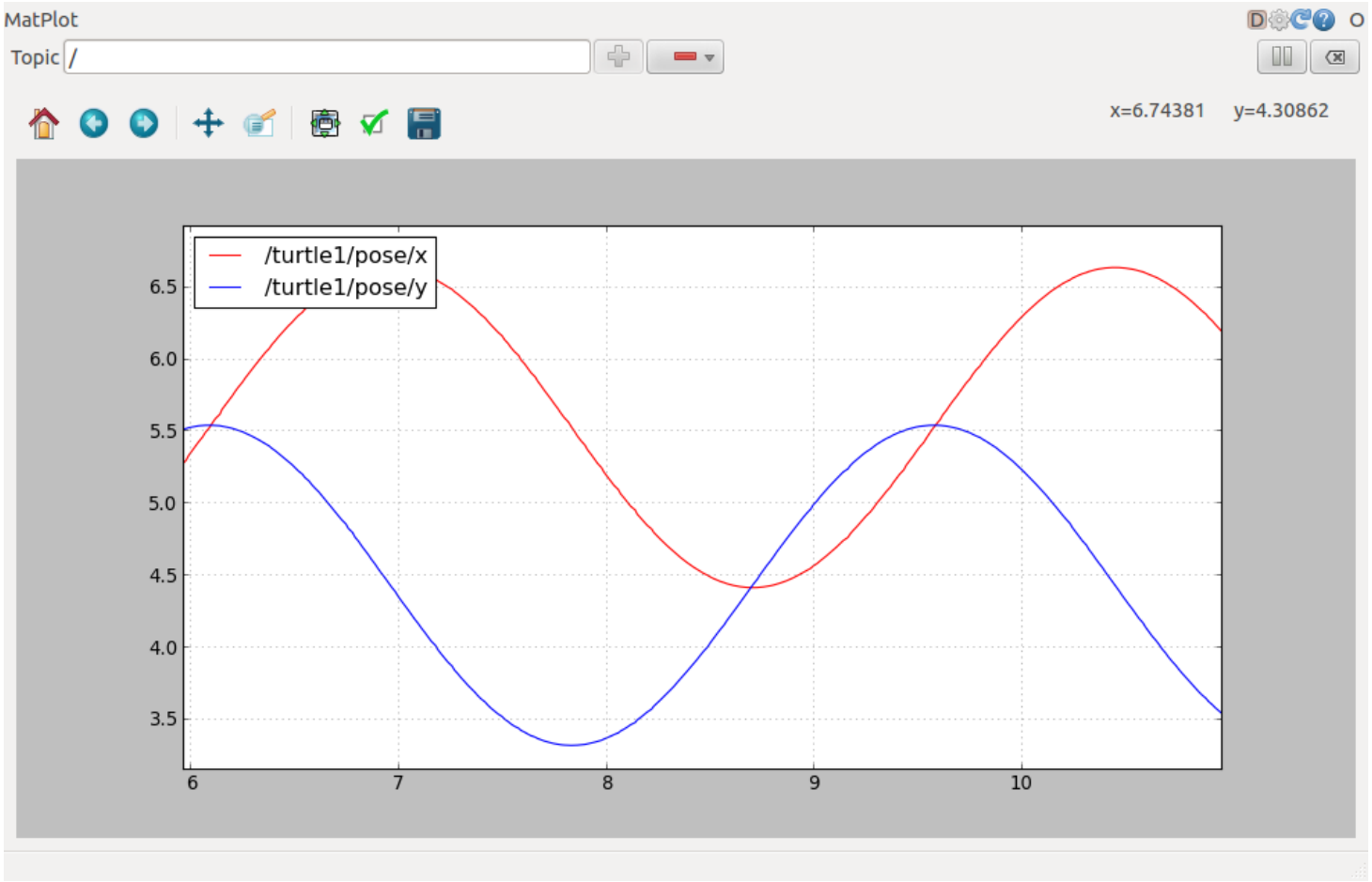



Portland State UNIVERSITY

# Tools – rqt_plot

- The rqt_plot plugin help you monitor encoders, voltages, or anything that can be represented as a number that varies over time.
- The rqt_plot plugin allows you to choose the plotting backend (e.g., matplotlib, Qwt, pyqtgraph) that best fits your needs.

# Tools – rqt_plot

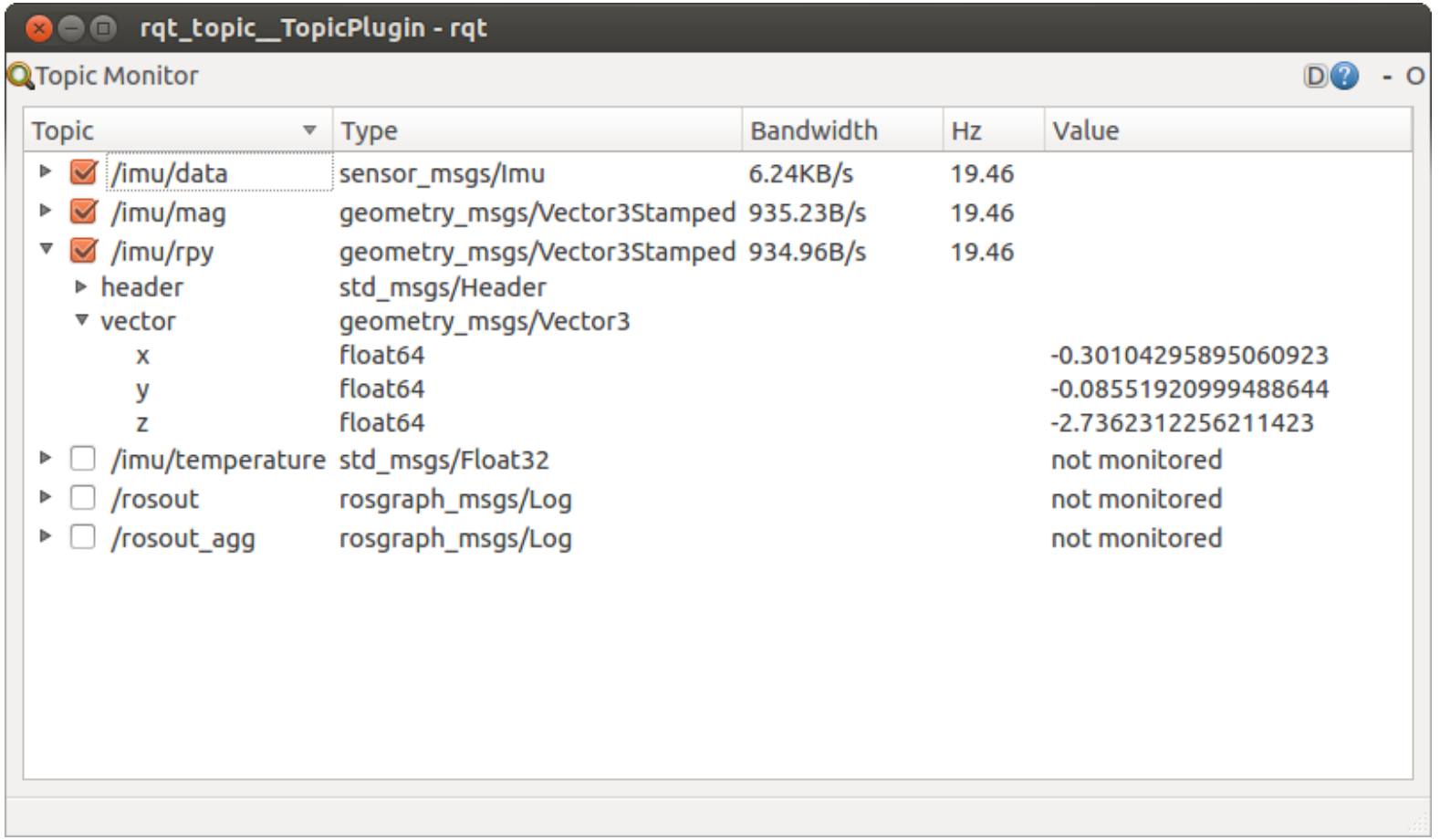



Portland State UNIVERSITY

# Tools – rqt_topic

- The rqt_plot plugin help you monitor encoders, voltages, or anything that can be represented as a number that varies over time.
- The rqt_plot plugin allows you to choose the plotting backend (e.g., matplotlib, Qwt, pyqtgraph) that best fits your needs.

# Tools – rqt_topic

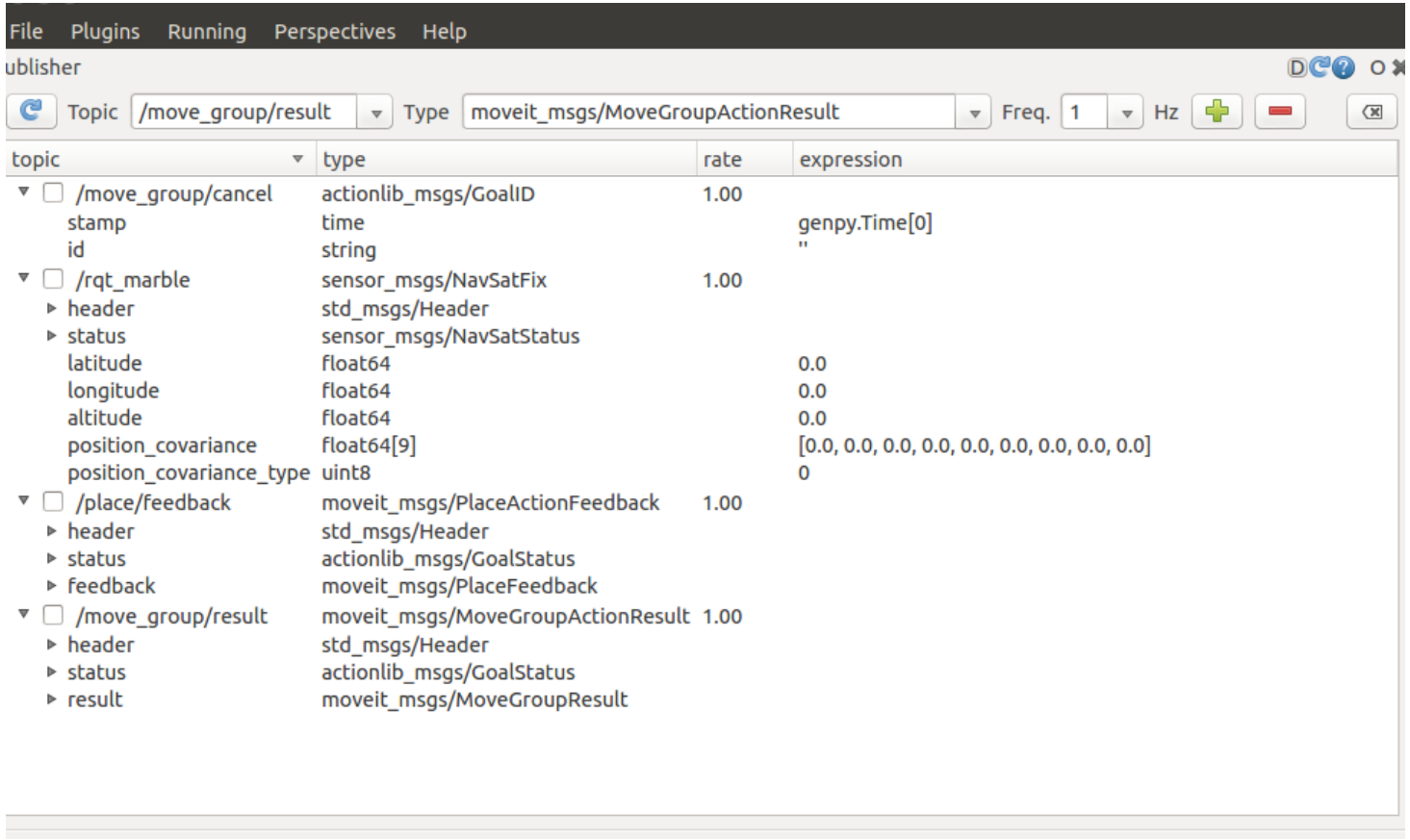



Portland State UNIVERSITY

# Tools – rqt_publisher

- The rqt_plot plugin help you monitor encoders, voltages, or anything that can be represented as a number that varies over time.
- The rqt_plot plugin allows you to choose the plotting backend (e.g., matplotlib, Qwt, pyqtgraph) that best fits your needs.

# Tools – rqt_publisher

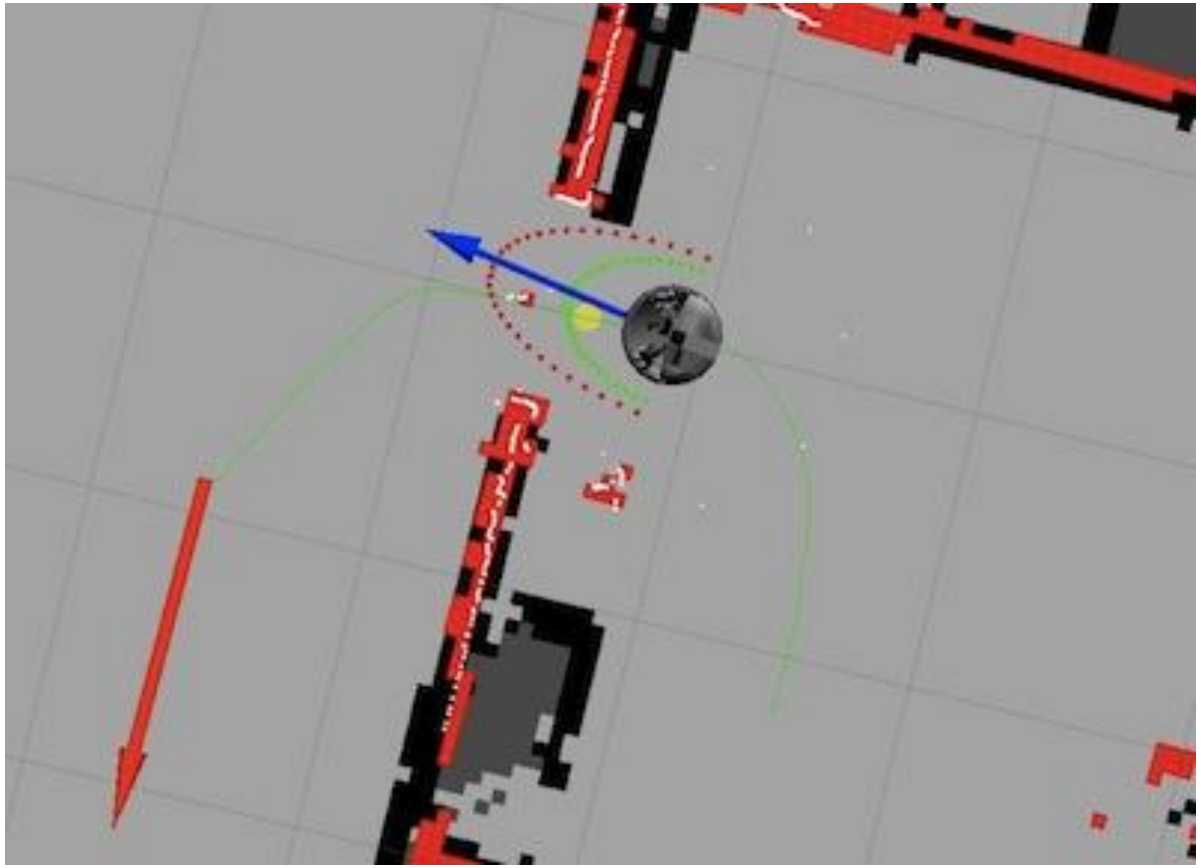



Portland State UNIVERSITY

# Robot-Specific Features

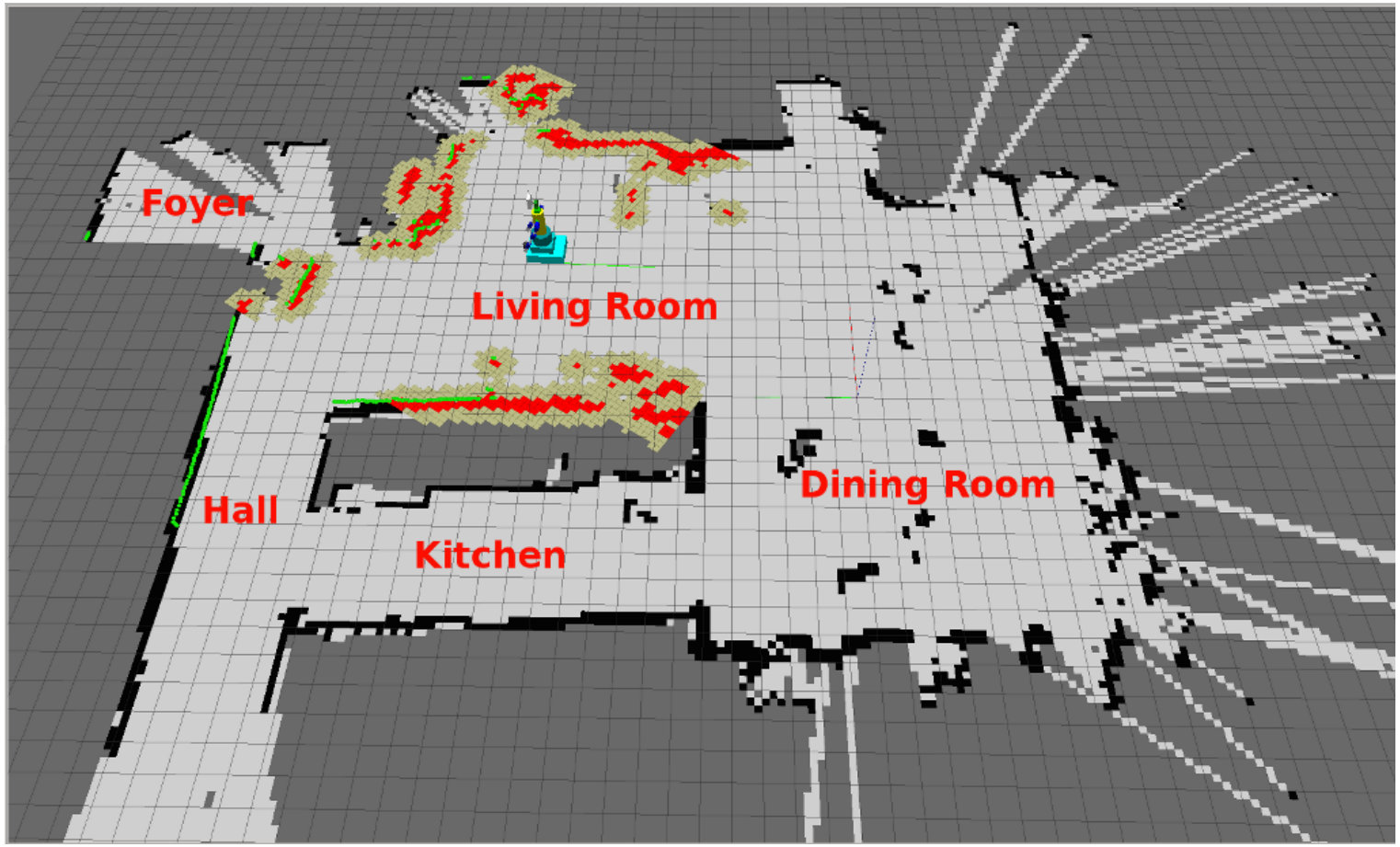
- In addition to the core middleware components, ROS provides common robot-specific libraries and tools that will get your robot up and running quickly. Here are just a few of the robot-specific capabilities that ROS provides:
  - Standard Message Definitions for Robots
  - Robot Geometry Library
  - Robot Description Language
  - Pre-emptiable Remote Procedure Calls
  - Diagnostics
  - Pose Estimation
  - Localization
  - Mapping
  - Navigation

# Robot-Specific Features – Pose Estimation



Portland State
UNIVERSITY

# Robot-Specific Features - Mapping




Portland State UNIVERSITY

# How to Install ROS?

- Install ROS Kinetic on PC - Ubuntu - Recommended
- Install ROS Kinetic on Virtual Machine
- Install ROS Kinetic on Raspberry Pi 3 - Stretch - Project2

# ROS Workspace Environment

- Standard name for it **catkin_ws**
- However, you can use whatever name you want.
- You can create it in any user location


- Before you use a work space first you need to source your setup.*sh.
- The setup.bash files setup your environment variables and paths.

Portland State
UNIVERSITY

# ROS Workspace Environment

- setup.bash Environment Setup File
- The binary catkin package includes a set of environment setup files that are used to extend your shell environment, so that your terminal can find and use any resources that have been installed to that location.
- the setup file included in the root of the distribution install directory (usually /opt/ros/<ros_distribution_name>)
- For example with ROS Kinetic on Ubuntu the files would be located as such:

```
/
  opt/
    ros/
      kinetic/
        setup.bash -- Environment setup file for Bash shell
        setup.sh   -- Environment setup file for Bourne shell
        setup.zsh  -- Environment setup file for zshell
        ...
```

Portland State UNIVERSITY

# ROS Workspace Environment

- If you want to change the environment of your current shell
  - **source /opt/ros/kinetic/setup.bash**

- If you want to overlay a workspace on top of your environment.
  - **source devel/setup.bash**

- If you want to source your terminal automatically.
  - **echo "source /opt/ros/kinetic/setup.bash" >> ~/.bashrc**
  - **source ~/.bashrc**

Portland State UNIVERSITY

# Catkin Workspace

**What is catkin?**

A catkin workspace is a folder where you modify, build, and install catkin packages. (wiki.ros.org)

Catkin packages can be built as a standalone project, in the same way that normal cmake projects can be built, but ***catkin also provides the concept of workspaces, where you can build multiple, interdependent packages together all at once.***

# Create a Catkin Workspace

- **mkdir -p ~/catkin_ws/src**

(create a new directory called src in another directory called catkin ws)

- **cd ~/catkin_ws/src**

( go into the src directory)

- **catkin_init_workspace**

(initialize the workspace)

- **cd ~/catkin_ws/**

(go into your brand new workspace)

- **catkin_make**

(build your code in a catkin workspace)

- **source devel/setup.bash**

(Before continuing make sure you source your new setup.*sh file)

- **echo $ROS_PACKAGE_PATH**

Test your path: /home/youruser/catkin_ws/src:/opt/ros/kinetic/share

# Catkin Workspace

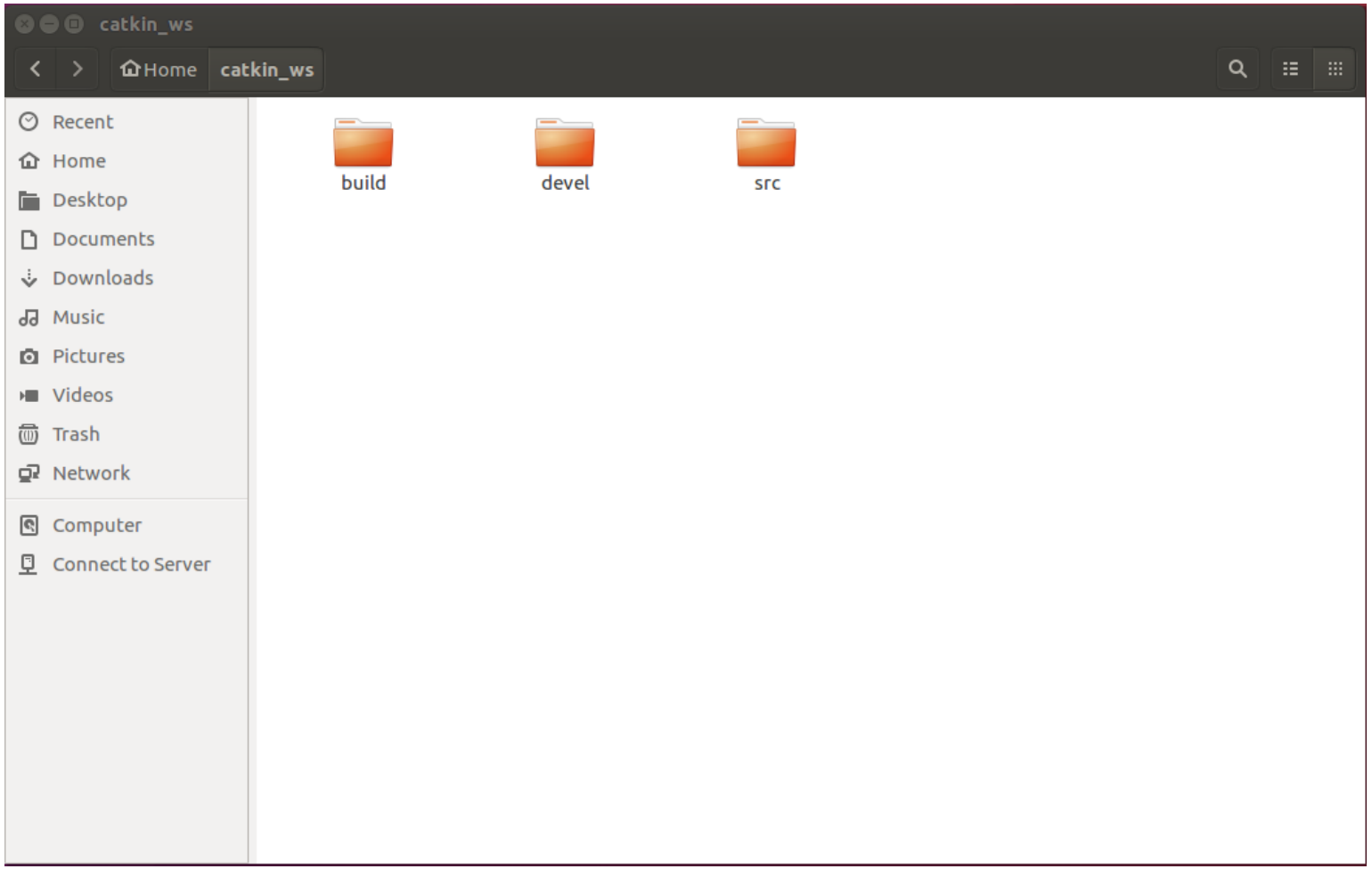
- You can also use **catkin build** command to build your code in your package.
- It is as same as the **catkin_make** command**.**

# Catkin Workspace


- ***Do not use catkin_make and catkin build at the same time***
- ***Do not mix them***

Portland State UNIVERSITY

# Catkin Workspace




Portland State
UNIVERSITY

# Catkin Workspace – src folder

- The source space is the folder is where catkin will be expected to look for packages when building.
- Each catkin project desired to be compiled from source should be checked out into subdirectories inside this directory.
- Packages are found recursively so they do not have to be direct subfolders.

# Catkin Workspace – dev folder

- The development space is where catkin generates the binaries and runtime libraries which are executable before installation.
- After the build step, inside this folder is expected everything needed to run nodes in packages which have been built.
- The development space can not be a folder which contains ROS packages in subfolders.
  - Ex: It can not equal to the workspace root as this would make the source space a subfolder which would lead to packages being found multiple times.

# Catkin Workspace – build folder

- The build space is the folder in which cmake is invoked and generates artifacts such as the CMakeCache

# Catkin Workspace – Overlying Workspaces

- In catkin style workspaces, overlaying of one workspace on top of another workspace is supported. E
- Each overlay can be built on top of another overlay, and any packages in a higher overlay will mask out packages in lower overlays.
- When overlaying any package all packages which depend on that package must be overlayed if they are intended to be used.
- This must be checked by the user creating the overlays. If a core package is overlayed and changes how it works, any package which depends on it and relies on the old behavior (whether runtime, or link time) will crash.

# Catkin Workspace – Overlying Workspaces

- mkdir ws1/src
- mkdir ws2/src
- mkdir ws3/src

- ws1 as an overlay of /opt/ros/kinetic/setup.bash
- ws2 as an overlay of ws1
- ws3 as an overlay of ws2

- http://wiki.ros.org/catkin/Tutorials/workspace_overlaying

# Catkin Workspace – Overlying Workspaces

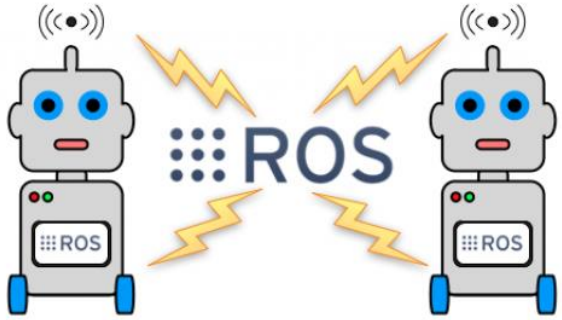
- cd ~/catkin_ws1
- catkin_make
- source /opt/ros/kinetic/setup.bash

- cd ~/catkin_ws2
- catkin_make
- source ~/catkin_ws1/devel/setup.bash

- cd ~/catkin_ws3
- catkin_make
- source ~/catkin_ws2/devel/setup.bash

# ROS Master

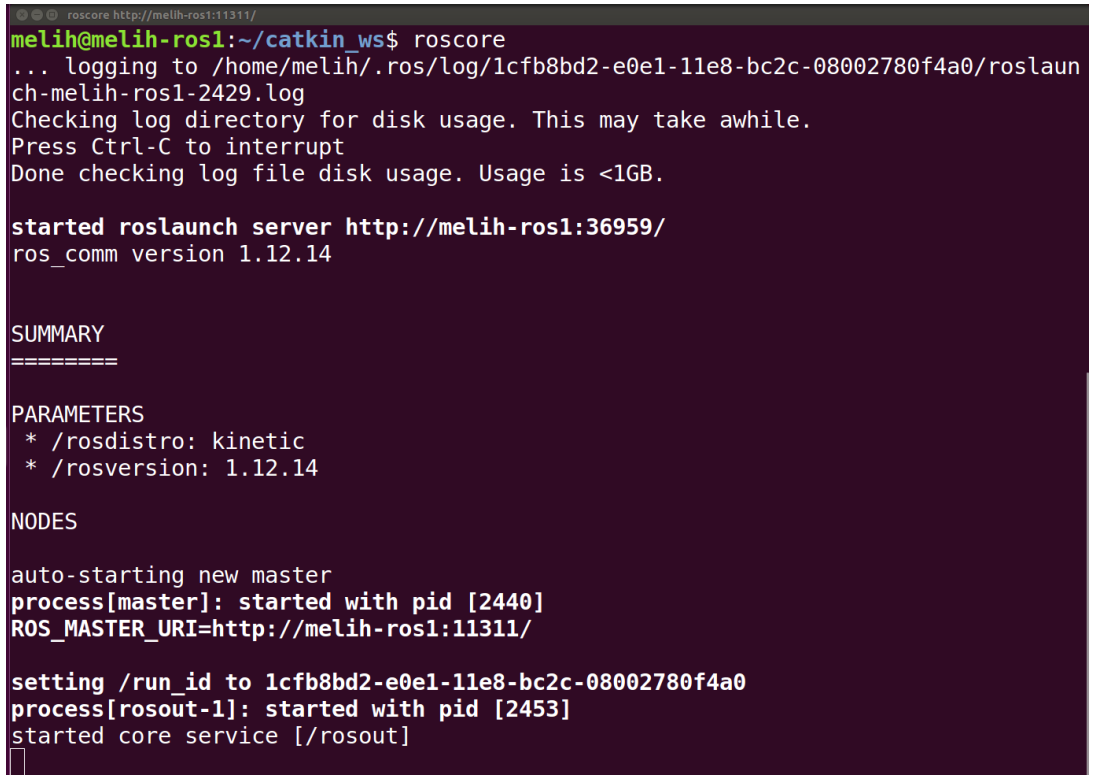
- In charge of managing the communication between nodes
- Every node registers with the master when you run them
- It tracks publishers and subscribers to topics as well as services.
- The role of the Master is to enable individual ROS nodes to locate one another.
- Once these nodes have located each other they communicate with each other peer-to-peer.
- The Master also provides the parameter server.




Portland State UNIVERSITY

# ROS Master

- In order to start a master type:
  - **roscore**



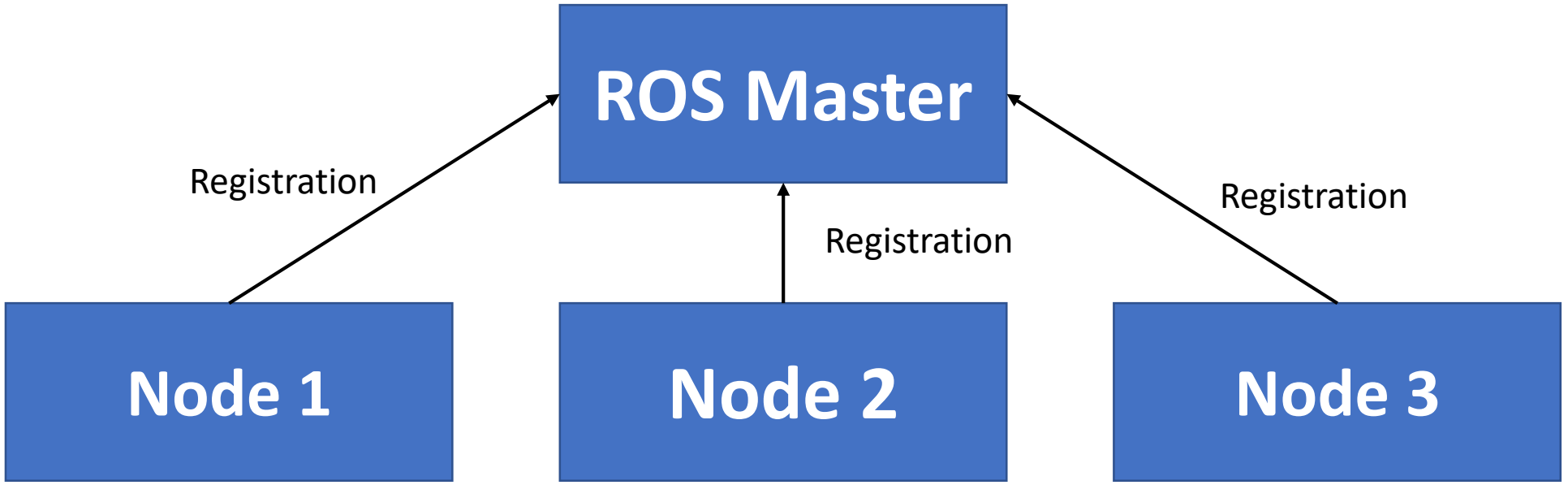

Portland State UNIVERSITY

# Ros Nodes

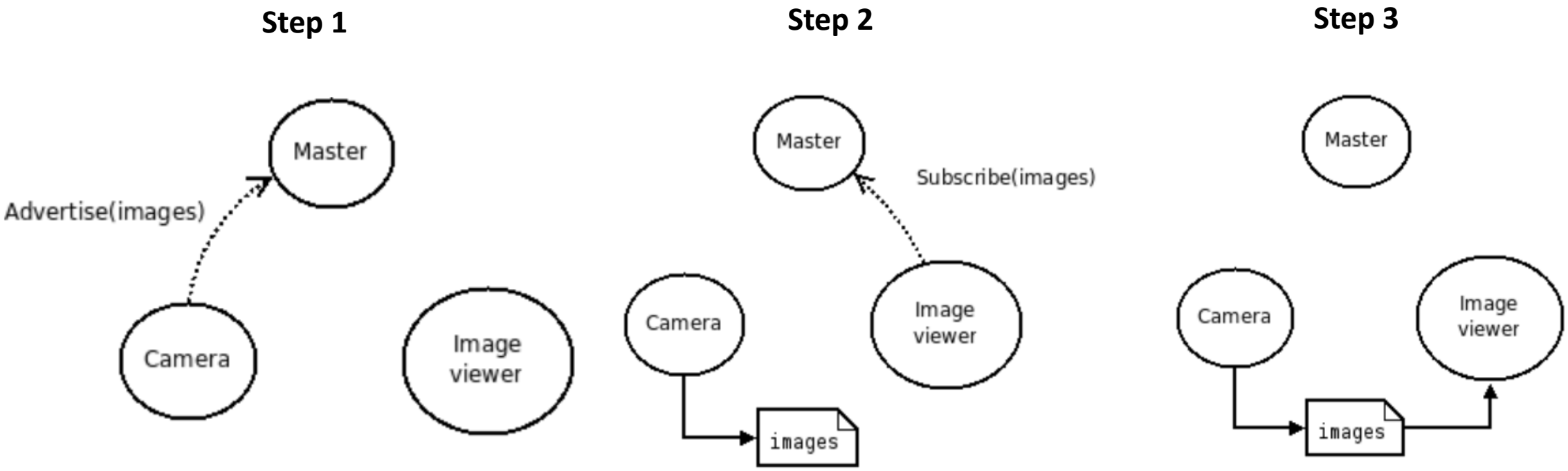
- Single-purpose, executable program written in Python, C++, Matlab etc
- Individually complied, executed and managed.
- You can run, stop, and kill nodes whenever you want.
- They run in parallel.
- In order to run a node type:
  - **rosrun package_name node_name**
- See active nodes running on ROS
  - **rosnode list**
- In order to get more information about nodes type:
  - **rosnode info node_name**

Portland State UNIVERSITY

# ROS Master and Nodes



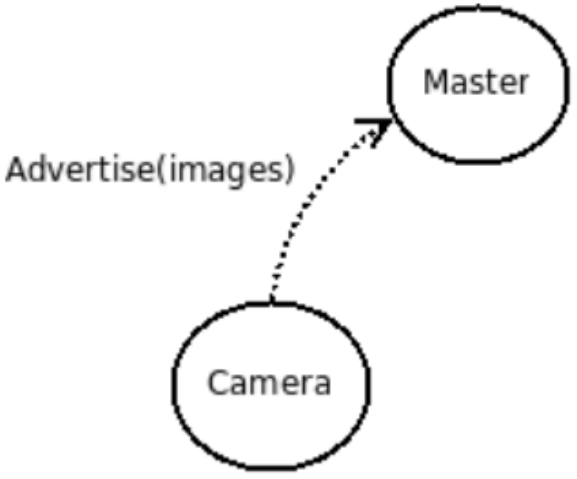
# ROS Master Example



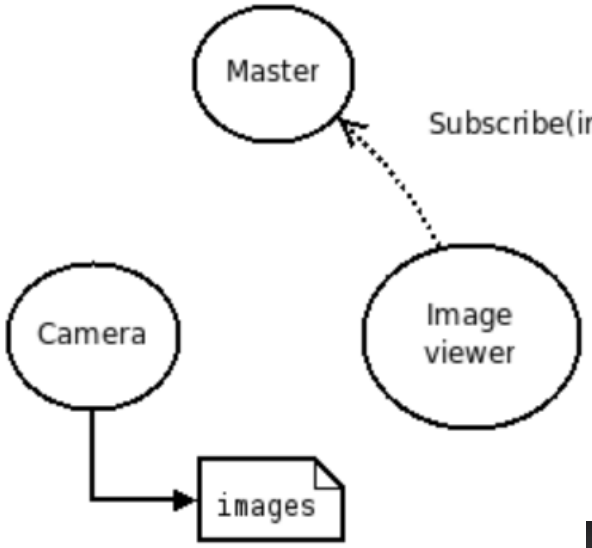

Portland State UNIVERSITY

# ROS Master Example

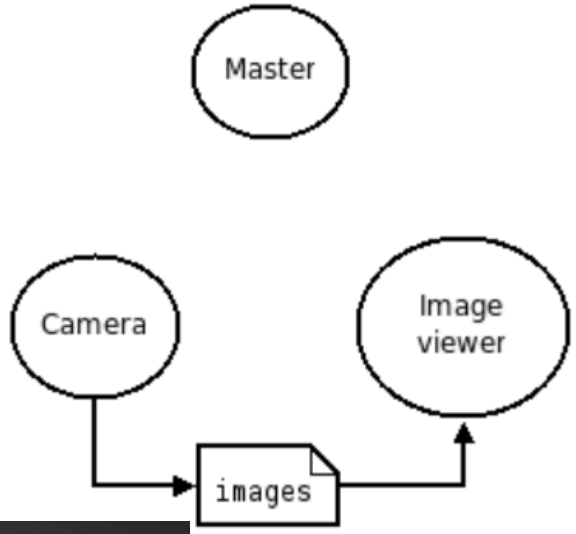
**Step 1**

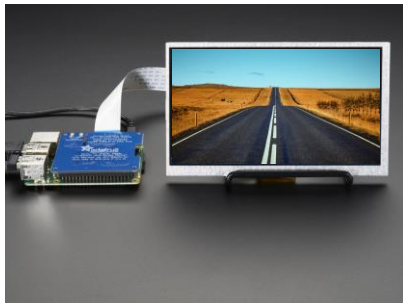



**Step 2**




**Step 3**



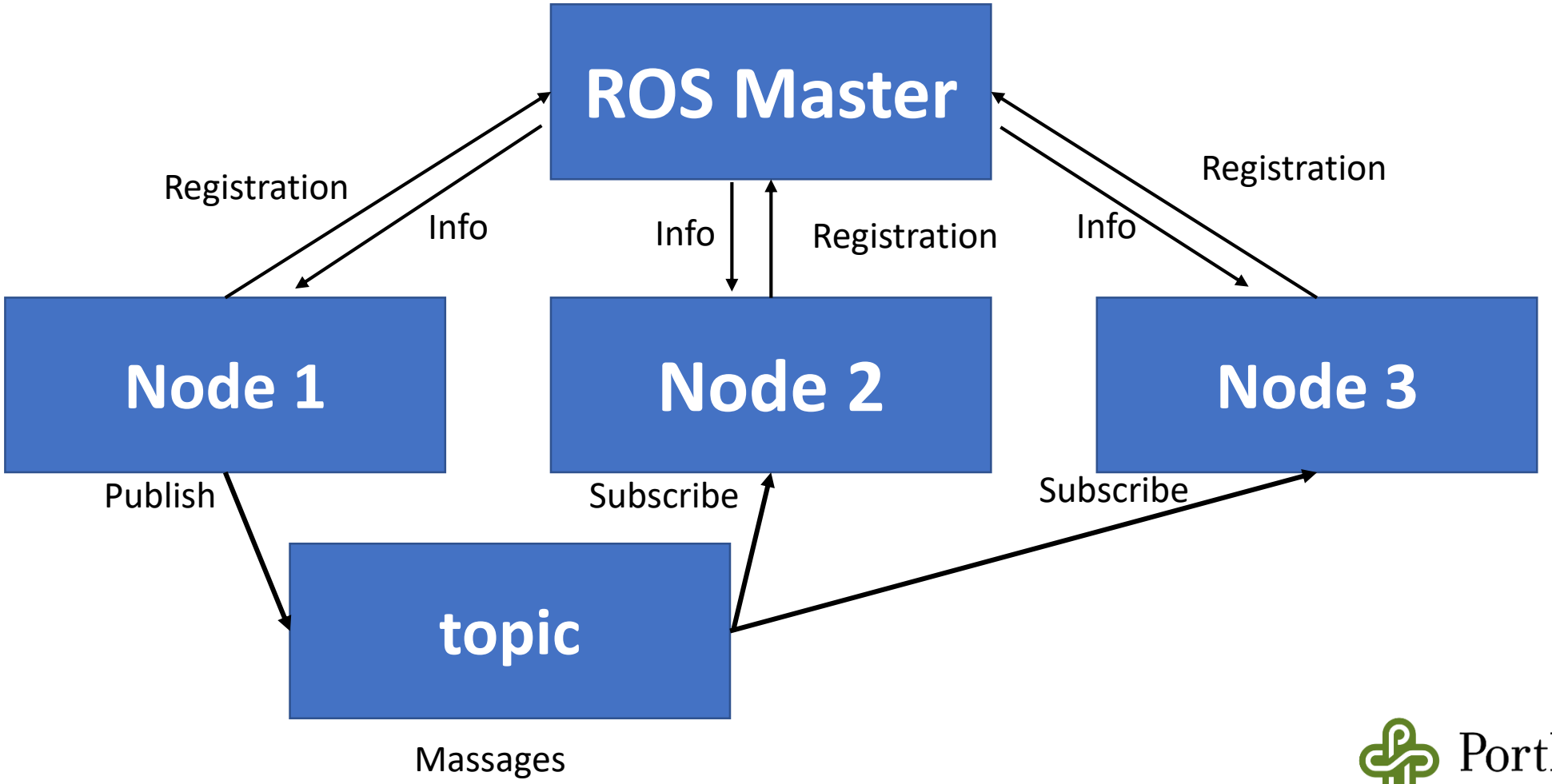
Portland State UNIVERSITY

# ROS Topics

- ROS Nodes communicate over topics.
- Each node can **publish** or **subscribe** topics
- When one of the nodes register with the master, node also tells master that I am the publisher of these topics. Therefore, other nodes can know who publishes that topic and subtribe that topic.
- Topics is used to transfer data from publisher node to subscriber node
- List all active topics :
  - **rostopic list**
- Subscribe and print the contents of a topic
  - **rostopic echo /topic-name**
- Show information about the topic
  - **rostopic info /topic-name**
- Show all the other possible rostopic commands
  - **rostopic -h**

Portland State UNIVERSITY

# ROS Master and Nodes




Portland State UNIVERSITY

# Other ROS Topic Commands

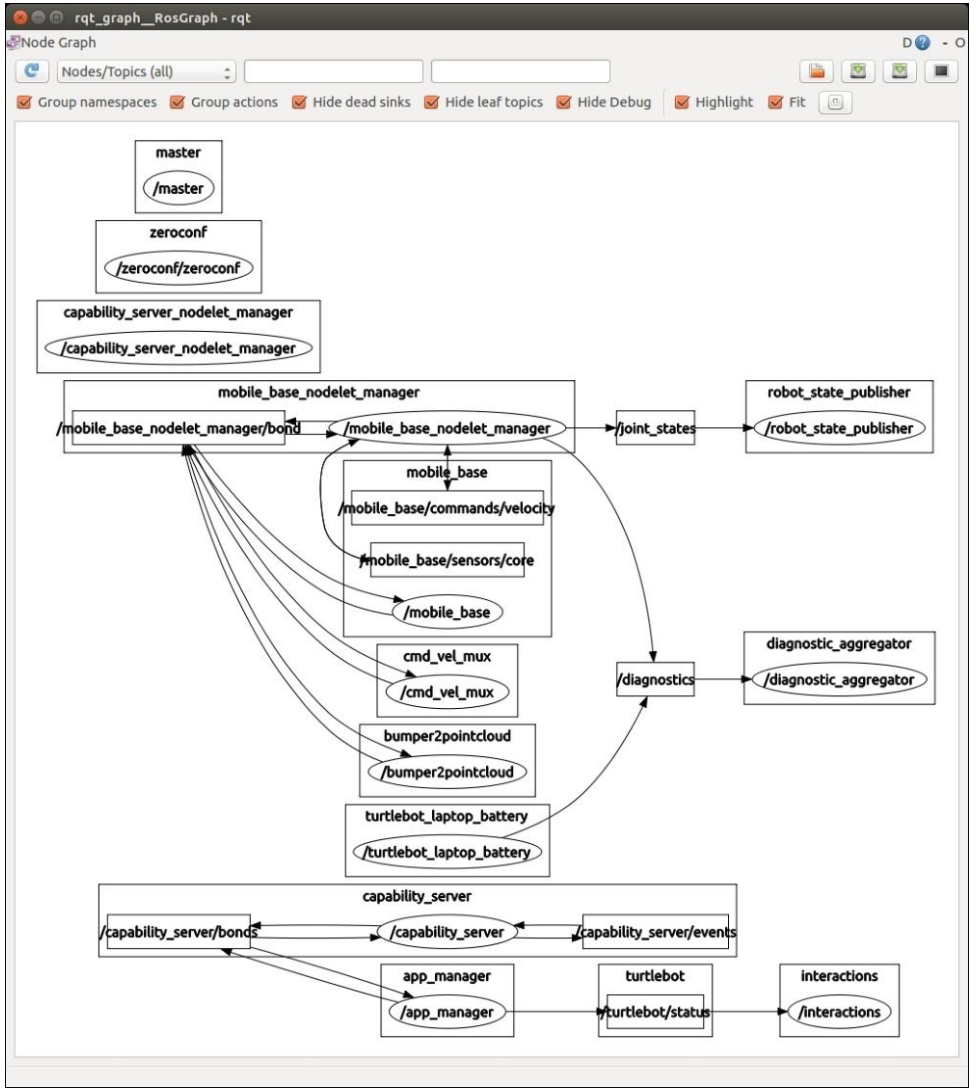
- **rostopic bw** display bandwidth used by topic
- **rostopic echo** print messages to screen
- **rostopic hz** display publishing rate of topic
- **rostopic list** print information about active topics
- **rostopic pub** publish data to topic
- **rostopic type** print topic type

# Using rqt-graph

- rqt_graph tool is used to create a dynamic graph of what's going on in the system that includes all nodes and topics
- To run rqt-graph
  - **rosrun rqt_graph rqt_graph**


Portland State UNIVERSITY

# Using rqt-graph




Portland State UNIVERSITY

# ROS Messages

- ROS message describes the data values that ROS nodes publish.
- Messages help ROS tolls to automatically generate code for the message type in several target languages.
- Message descriptions are in .msg files in the msg/ directory.
- Messages can be nested in each other.
- Each ROS distribution can have a different description for a message
  - You can go to http://wiki.ros.org/msg and check the message type
- Publish a message to a subscriber ROS node on command line.
  - rostopic pub /mytopic std_msg/String “data: ‘Portland State University’”

# Summary of ROS Messages

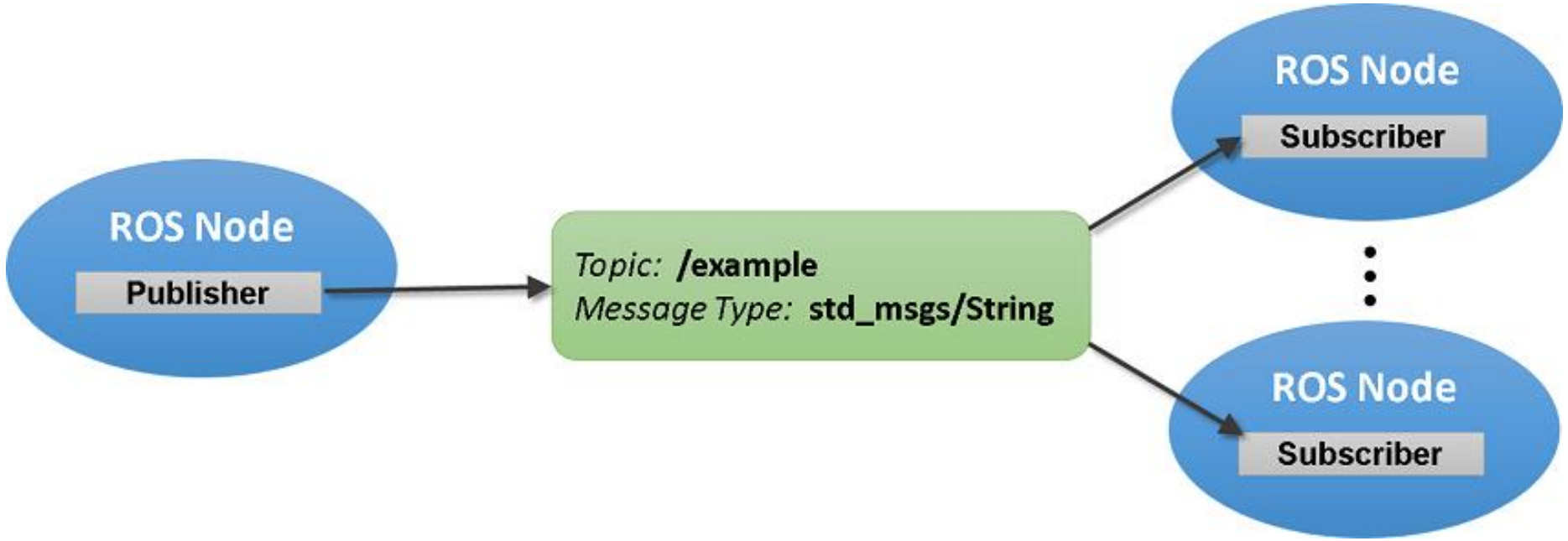



Image Source Link

Portland State UNIVERSITY

# Example ROS Message Definition:

## 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
```

*autogenerated on Tue, 17 Jan 2017 20:57:22*

Portland State UNIVERSITY

# Message Types

You must not use the names of built-in types or header when constructing your own message types.

**Built-in types**:

| Primitive Type | Serialization | C++ | Python |
|---|---|---|---|
| bool (1) | unsigned 8-bit int | `uint8_t` (2) | `bool` |
| int8 | signed 8-bit int | `int8_t` | `int` |
| uint8 | unsigned 8-bit int | `uint8_t` | `int` (3) |
| int16 | signed 16-bit int | `int16_t` | `int` |
| uint16 | unsigned 16-bit int | `uint16_t` | `int` |
| int32 | signed 32-bit int | `int32_t` | `int` |
| uint32 | unsigned 32-bit int | `uint32_t` | `int` |
| int64 | signed 64-bit int | `int64_t` | `long` |
| uint64 | unsigned 64-bit int | `uint64_t` | `long` |
| float32 | 32-bit IEEE float | `float` | `float` |
| float64 | 64-bit IEEE float | `double` | `float` |
| string | ascii string (4) | `std::string` | `str` |
| time | secs/nsecs unsigned 32-bit ints | ros::Time | rospy.Time |
| duration | secs/nsecs signed 32-bit ints | ros::Duration | rospy.Duration |

# ROS Messages - Remapping

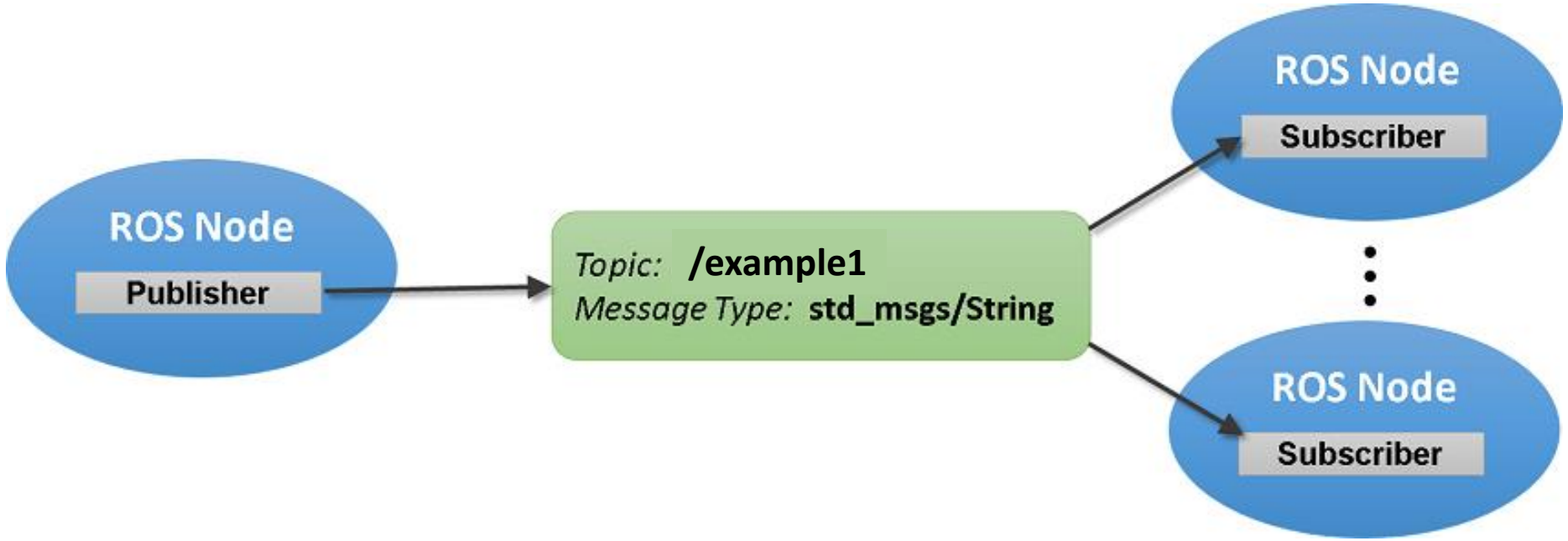
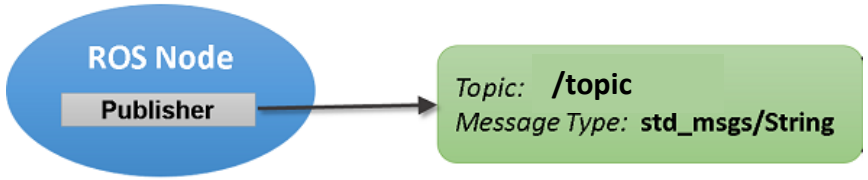



Image Source Link

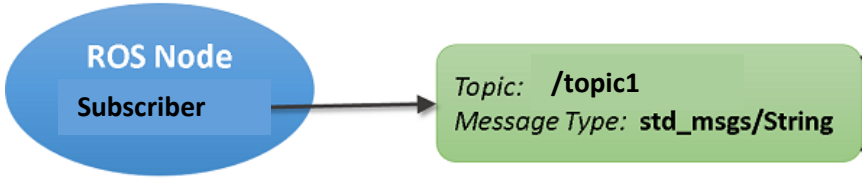
Portland State UNIVERSITY

# ROS Messages - Remapping

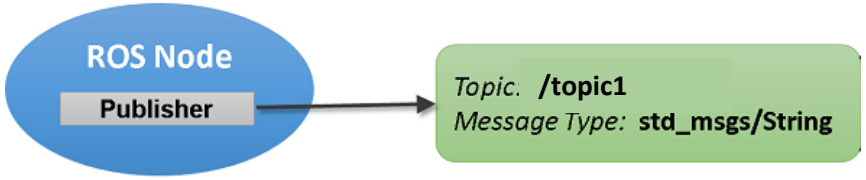
**Node 1 :** rosrun tutorials listener.py /topic:=/topic1
**Node 2 :** rosrun tutorials talker.py /topic:=/topic1



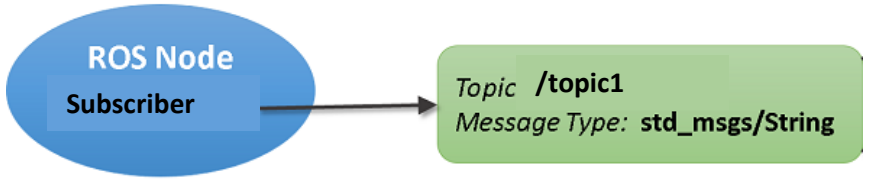

talker.py




listener.py




talker.py




listener.py

Portland State UNIVERSITY

# ROS Package

- Software in ROS is organized in packages.
- A package can contain your ROS nodes, any libraries, a dataset, configuration files, message types, scripts, service types, some third-party software.
- When you want to create a project in ROS, first you need to create a package
- You can also download and install many packages written by other ROS users around the world
- You can submit the ROS package you write to the ROS package repository
- Every ROS package is for certain ROS distributions
- Packages have dependencies
- http://www.ros.org/browse/list.php?package_type=package&distro=indigo

Portland State UNIVERSITY

# ROS Package – Package.xml

Toggle line numbers

```
 1 <?xml version="1.0"?>
 2 <package>
 3   <name>beginner_tutorials</name>
 4   <version>0.1.0</version>
 5   <description>The beginner_tutorials package</description>
 6
 7   <maintainer email="you@yourdomain.tld">Your Name</maintainer>
 8   <license>BSD</license>
 9   <url type="website">http://wiki.ros.org/beginner_tutorials</url>
10   <author email="you@yourdomain.tld">Jane Doe</author>
11
12   <buildtool_depend>catkin</buildtool_depend>
13
14   <build_depend>roscpp</build_depend>
15   <build_depend>rospy</build_depend>
16   <build_depend>std_msgs</build_depend>
17
18   <run_depend>roscpp</run_depend>
19   <run_depend>rospy</run_depend>
20   <run_depend>std_msgs</run_depend>
21
22 </package>
```

Portland State UNIVERSITY

# ROS Launch

- Sometimes you have so many nodes with many different arguments and parameters to run in a project.
- Roslaunch is a tool for easily launching multiple ROS nodes as well as setting parameters in the parameter server.
- The basic idea is to list, in a specific XML format, a group of nodes that should be started at the same time.

# ROS Launch – Simple Example

```
<launch>
        <node name="telephone1" pkg="ros_lecture1" type="telephone.py" output="screen"/>
</launch>
```

Portland State UNIVERSITY

# Example Launch File

```
1  <launch>
2    <node
3      pkg="turtlesim"
4      type="turtlesim_node"
5      name="turtlesim"
6      respawn="true"
7    />
8    <node
9      pkg="turtlesim"
10     type="turtle_teleop_key"
11     name="teleop_key"
12     required="true"
13     launch-prefix="xterm -e"
14   />
15   <node
16     pkg="agitr"
17     type="subpose"
18     name="pose_subscriber"
19     output="screen"
20   />
21 </launch>
```

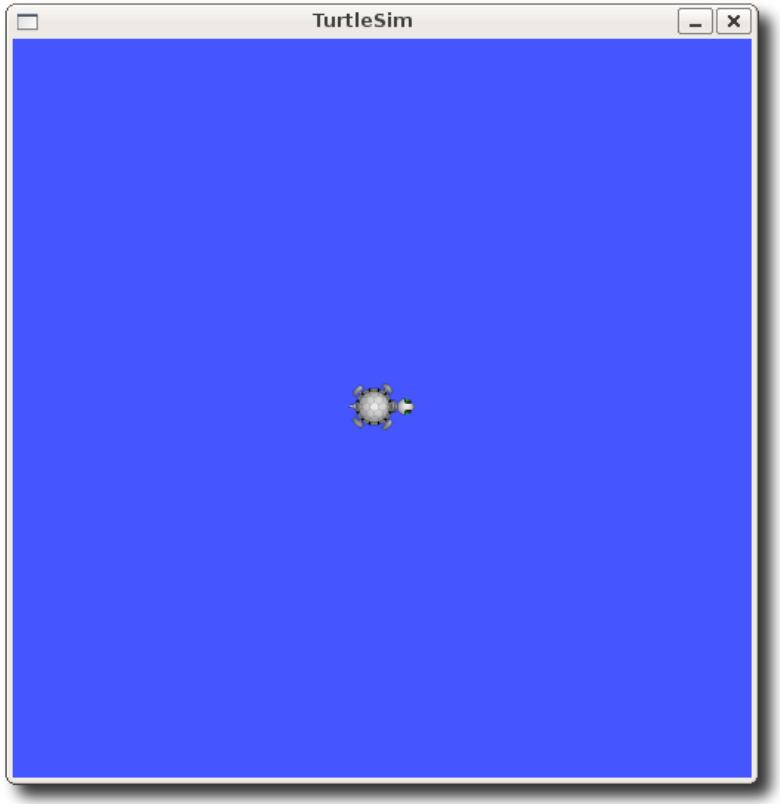
Listing 6.1: A launch file called example.launch that starts three nodes at once.

Portland State UNIVERSITY

# Some useful ROS command-line tools

- rospack: find and retrieve information about packages
- catkin_create_pkg: create a new package
- catkin_make: build a workspace of packages
- rosdep: install system dependencies of a package
- rqt: In rqt there is a plugin called "Introspection/Package Graph", which visualizes package dependencies as a graph

# Turtlesim Example




http://wiki.ros.org/turtlesim

- **Roscore**
- **Nodes**
  - rosrun turtlesim turtlesim_node
  - rosrun turtlesim_turtle_teleop_key
- **Topic**
- rostopic pub -1 /turtle1/cmd_vel geometry_msgs/Twist -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, 1.8]'

Portland State UNIVERSITY

# geometry_msgs/Twist

- Twist message should have:
  - a linear component for the x, y, and z axes velocities,
  - an angular component for the for the x, y, and z axes angular rate

## geometry_msgs/Twist Message

**File:** geometry_msgs/Twist.msg

### Raw Message Definition

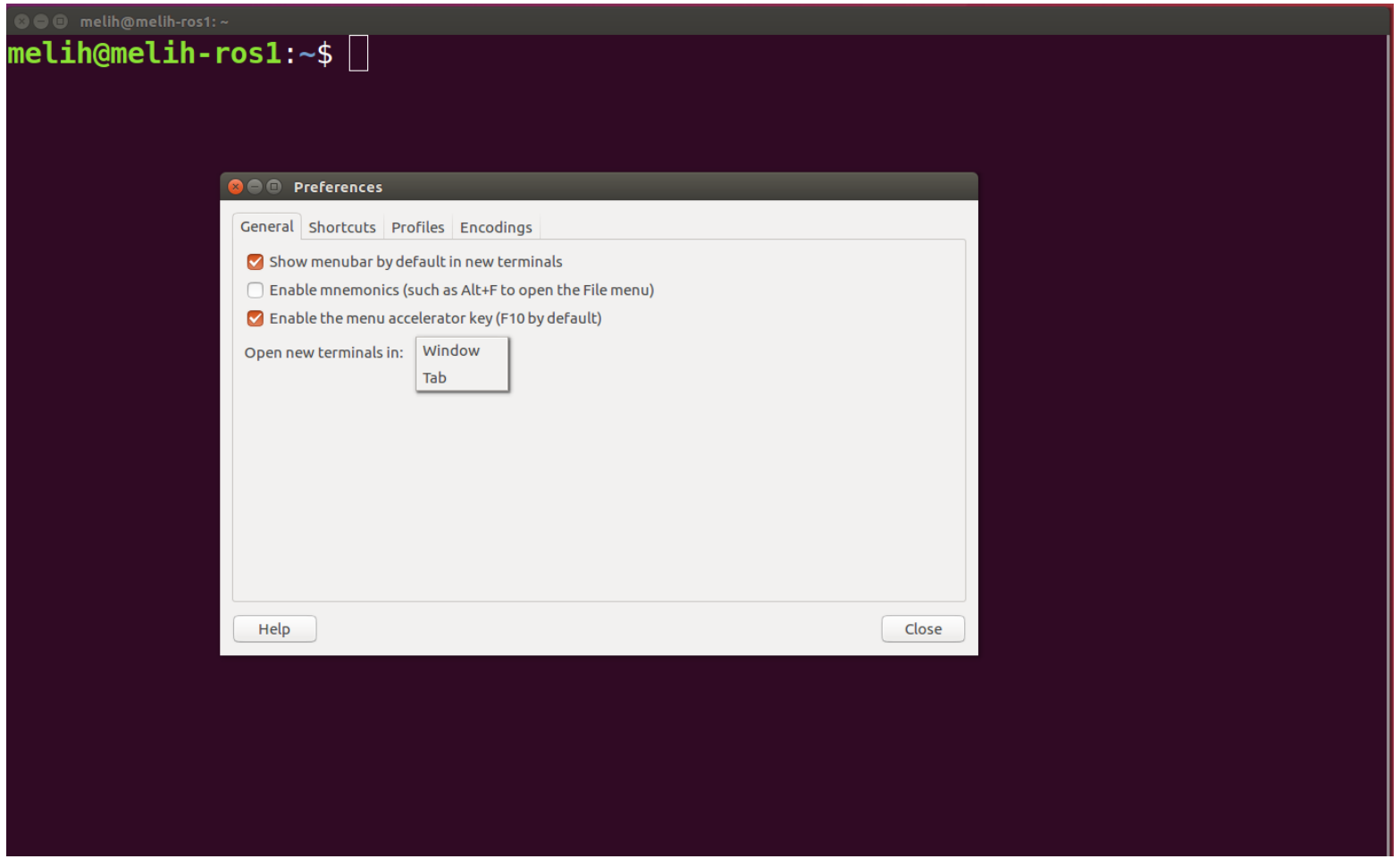
```
# This expresses velocity in free space broken into its linear and angular parts.
Vector3  linear
Vector3  angular
```

### Compact Message Definition

```
geometry_msgs/Vector3 linear
geometry_msgs/Vector3 angular
```

Portland State UNIVERSITY

# Ubuntu Terminal – Open New Tab




Portland State UNIVERSITY

# Resources

- http://www.iris.ethz.ch/the-institute/robotics-systems-lab.html
- http://www.ros.org/

Portland State
UNIVERSITY