Getting Started With ROS

MRE/EME 5983 Robot Operating Systems

Overview

- Introduction to ROS
- ROS Example
- ROS packages, nodes, topics and messages
- ROS utilities
- Summary

Overview

- Introduction to ROS
- ROS Example
- ROS packages, nodes, topics and messages
- ROS utilities
- Summary

What is ROS?

- **R**obot **O**perating **S**ystem

- ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.

Source: https://www.ros.org

Why ROS?

- Distributed computation
  - Many modern robot systems rely on software that spans many different processes and runs across several different computers
- Software reuse
  - ROS's standard packages provide stable, debugged implementations of many important robotics algorithms.
  - ROS's message passing interface is becoming a de facto standard for robot software interoperability.
- Rapid testing
  - Well-designed ROS systems separate the low-level direct control of the hardware and high-level processing and decision making into separate programs.
  - ROS also provides a simple way to record and play back sensor data and other kinds of messages

Where is ROS Being Used?

- Sample of supported robots

Spot
Boston Dynamics

Remote Combat Vehicle
DoD

Which Sensors Does ROS Support?

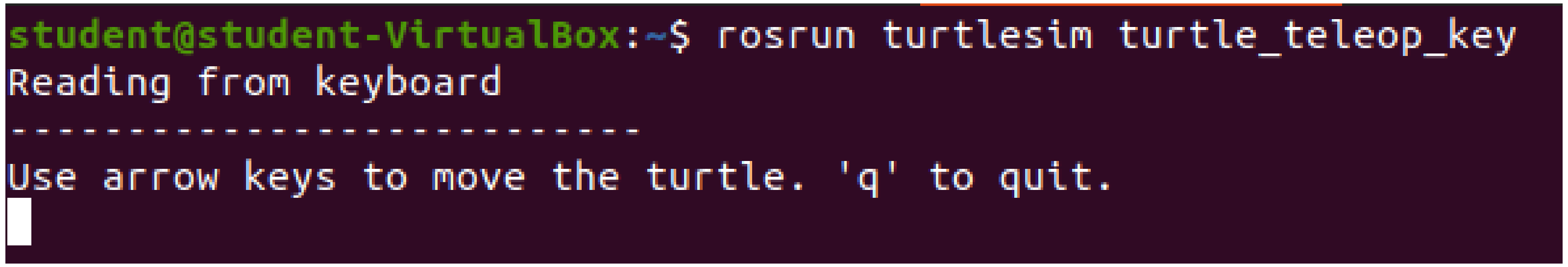
- 1D/2D/3D range finders
  - Sharp IR range finder
  - Hokuyo, SICK laser scanners
  - Microsoft Kinect
  - Velodyne LIDARs
- Cameras
  - monocular and stereo
  - USB and Ethernet
- Force/torque/touch sensors
- Motion capture systems
- Pose estimation (IMU/GPS)
- Audio/Speech recognition
- RFID
- Sensor/actuator interfaces: Arduino, ...

Overview

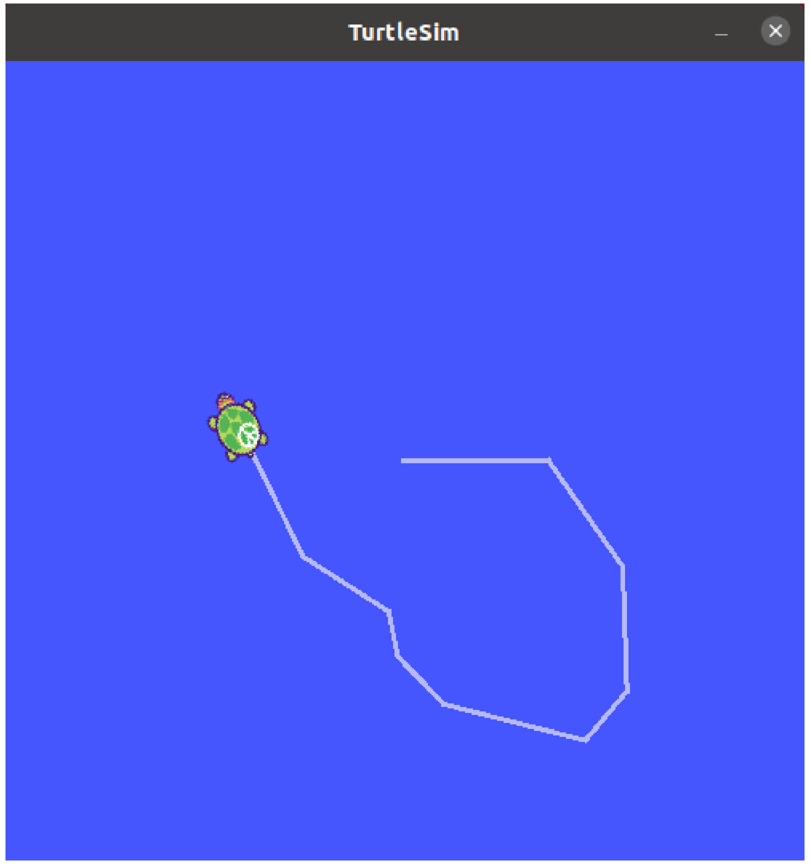
- Introduction to ROS
- ROS Example
- ROS packages, nodes, topics and messages
- ROS utilities
- Summary

ROS In Action – A Small Example

- Small example using the turtlesim simulator
- Use three terminals to execute the following command
    - Terminal 1: roscore
    - Terminal 2: rosrun turtlesim turtlesim_node
    - Terminal 3: rosrun turtlesim turtle_teleop_key

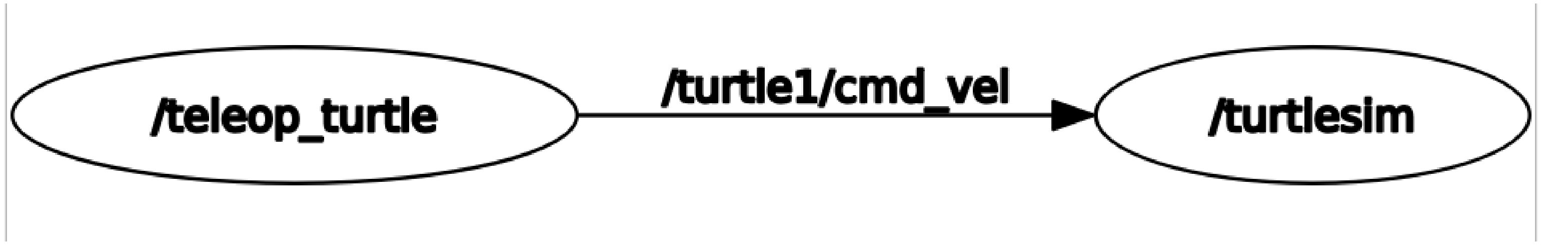


```
student@student-VirtualBox:~$ rosrun turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle. 'q' to quit.
```

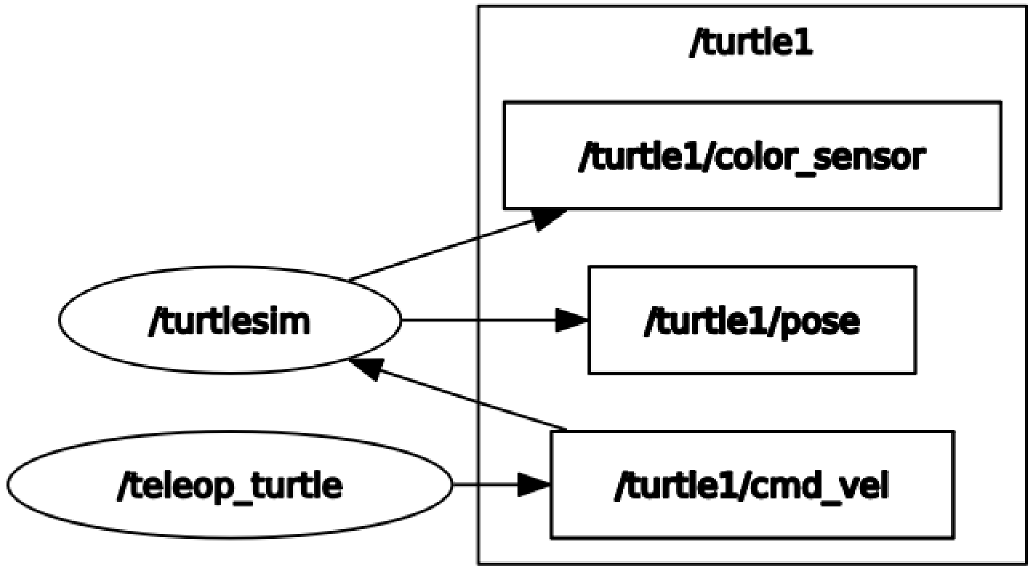


ROS In Action – A Small Example

- Diagram of the ROS system



- More details about our system



Overview

- Introduction to ROS
- ROS Example
- ROS packages, nodes, topics and messages
- ROS utilities
- Summary

Elements of ROS – Packages

- Software in ROS is organized in packages. A package may contain ROS nodes, a ROS-independent library, a dataset, configuration files, a third-party piece of software, or anything else that logically constitutes a useful module. The goal of these packages it to provide this useful functionality in an easy-to-consume manner so that software can be easily reused. In general, 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.

Source: http://wiki.ros.org/Packages

Elements of ROS – Packages

- Example package contents
  - We will create many packages in this course and learn various elements of ROS packages
  - One important file is package.xml
  - This file contains details about the package name, version, maintainer, and dependencies

- Useful package commands
  - **rospack list** – lists all installed ROS packages
  - **rospack find <package>** – returns the location of a package
  - **roscd <package>** – changes the active directory to the package location

Elements of ROS – Packages

- Package naming conventions - **Mandatory Rules**
  - **only consist of lowercase alphanumerics and _ separators and start with an alphabetic character.** This allows it to be used in generated symbols and functions in all supported languages. Lowercase is required because the names are used in directories and filenames and some platforms do not have case sensitive filesystems.
  - **not use multiple _ separators consecutively.** This allows generated symbols to use the __ separator to guaranteed the avoidance of collisions with symbols from other packages, for example in the message generators.
  - **be at least two characters long.** This rule is simply to force the name of the package to be more human understandable. It's recommended to be notably longer.

Source: https://www.ros.org/reps/rep-0144.html

Elements of ROS – Packages

- Package naming conventions - **Global Rules**
  - **Package names should be specific enough to identify what the package does.** For example, a motion planner should not be called planner. If it implements the wavefront propagation algorithm, it might be called wavefront_planner. There's obviously tension between making a name specific and keeping it from becoming overly verbose
  - **Using catchall names such as utils should be avoided.** They do not scope what goes into the package or what should be outside the package
  - **A package name should not contain** ros **as it is redundant.** Exceptions include core packages and ROS bindings of an upstream library (e.g. moveit_ros)
  - **The package name should describe what the package does, not where it came from.** One of ROS's goals is to develop a canonical set of tools for making robots do interesting things. Then again, as stated in the rules below, if a package is specialized by an entity (lab, company, ...), prepend the name of the entity. But once the package is commonly used, owned and maintained, that name can be dropped as the package becomes the reference
  - D**o not use a name that's already been taken.**

Source: https://www.ros.org/reps/rep-0144.html

Elements of ROS – Nodes

- ROS nodes are programs that run under ROS

- Our example has three nodes
  - rosout (executed by roscore)
  - turtlesim (executed by rosrun turtlesim turtlesim_node)
  - teleop_turtle (executed by rosrun turtlesim turtle_teleop_key)

- Useful node commands
  - **rosrun <package> <node>** – runs an instance of a node in a given package
  - **rosnode list** – provides a list of executing nodes
  - **rosnode info <node>** – provides info about an executing node
  - **rosnode kill <node>** – stops execution of a given node (ctrl-c works as well)

Elements of ROS – Nodes

- Node naming conventions
  - Nodes have both a type and name. The type is the name of the executable to launch the node. The name is what is passed to other ROS nodes when it starts up. We separate these two concepts because names must be unique, whereas you may have multiple nodes of the same type.

  - When possible, the default name of a node should follow from the name of the executable used to launch the node. This default name can be remapped at startup to something unique.

  - In general, we encourage the node type names to be short because they are scoped by the package name. For example, if your laser_scan package has a viewer for laser scans, simply call it view (instead of laser_scan_viewer).

Source: http://wiki.ros.org/ROS/Patterns/Conventions

Elements of ROS – Topics and Messages

- ROS topics and messages are the primary method to send/receive data
- ROS nodes send and receive **messages** that are organized in named **topics**

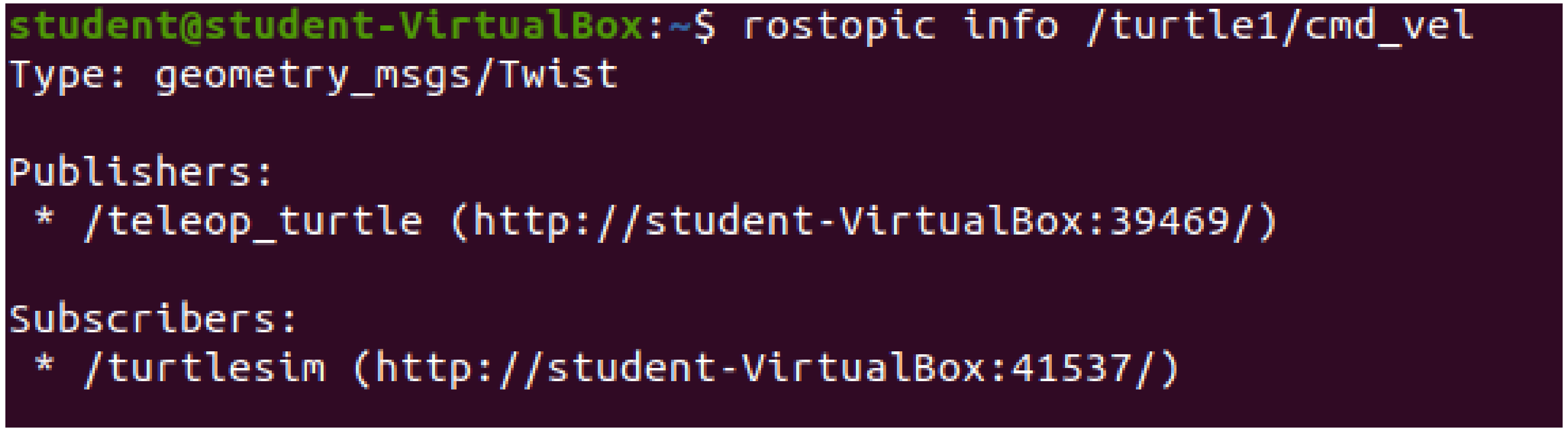
- Our example contains the following **topics**
  - /rosout
  - /rosout_agg
  - /turtle1/cmd_vel
  - /turtle1/color_sensor
  - /turtle1/pose

- **Messages** are sent on these topics to communicate between nodes

Elements of ROS – Topics and Messages

- Useful topic commands
  - **rostopic list** – provides a list of all the topics
  - **rostopic info <topic>** – provides information about the topic
  - **rostopic echo <topic>** – echoes messages on topic to the terminal
  - **rostopic pub <topic>** – publish messages to a topic

- In our example



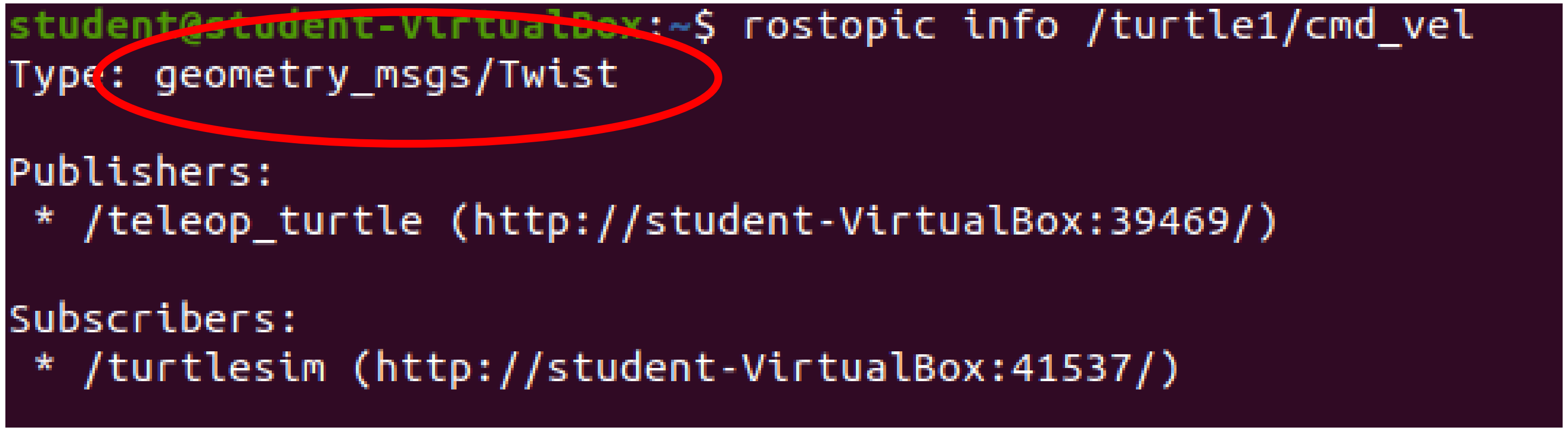
```
student@student-VirtualBox:~$ rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist

Publishers:
 * /teleop_turtle (http://student-VirtualBox:39469/)

Subscribers:
 * /turtlesim (http://student-VirtualBox:41537/)
```

Elements of ROS – Topics and Messages

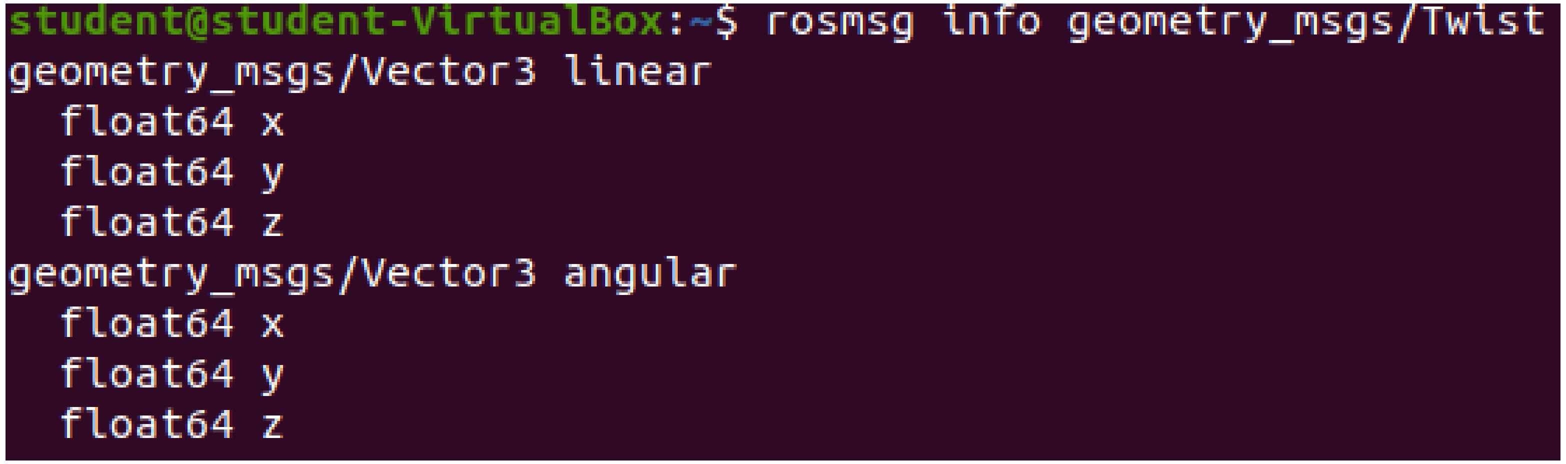
- To learn more about message types, we can use **rosmsg**



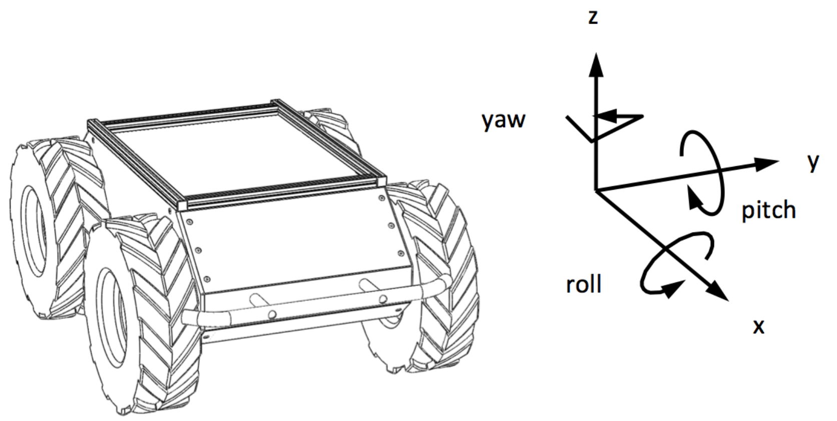
```
student@student-VirtualBox:~$ rostopic info /turtle1/cmd_vel
Type: geometry_msgs/Twist

Publishers:
 * /teleop_turtle (http://student-VirtualBox:39469/)

Subscribers:
 * /turtlesim (http://student-VirtualBox:41537/)
```



```
student@student-VirtualBox:~$ rosmsg info geometry_msgs/Twist
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```



Elements of ROS – Topics and Messages

- Useful message commands
  - **rosmsg show <message>** – show message description
  - **rosmsg info <message>** – Alias for rosmsg show
  - **rosmsg list <message>** – List all messages

- Remember
  - You cannot publish messages with **rosmsg** command
  - You can publish messages with **rostopic** command (**rostopic pub**)

Elements of ROS – Topics and Messages

- Topic naming conventions
  - Topic and service names live in a hierarchical namespace, and client libraries provide mechanisms for remapping them at runtime, so there is more flexibility than with packages. However, it's best to minimize the need for namespacing and name remapping.
    - Topic and service names should follow common C variable naming conventions: lower case, with underscore separators, e.g. laser_scan
    - Topic and service names should be reasonably descriptive. If a planner node publishes a message containing its current state, the associated topic should be called planner_state, not just state.

Source: http://wiki.ros.org/ROS/Patterns/Conventions

Elements of ROS – Topics and Messages

- Message naming conventions
  - Message files are used to determine the class name of the autogenerated code. As such, they must be CamelCase. e.g. LaserScan.msg
    - NOTE: This is an exception to the convention that all filenames are lower case and underscore separated. Using CamelCase message names will prevent issues from arising due to inconsistent support for filename case sensitivity across various operating systems.
  - Message fields should be lowercase with underscore separation. e.g. range_min

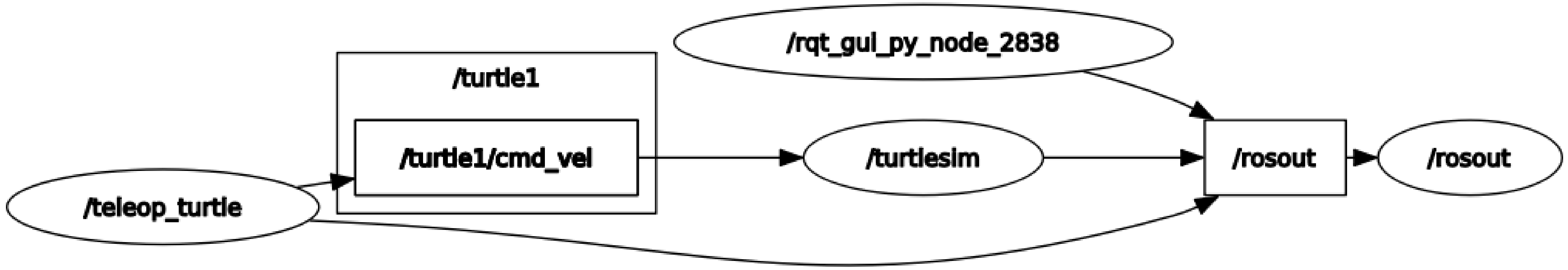
Source: http://wiki.ros.org/ROS/Patterns/Conventions

Overview

- Introduction to ROS
- ROS Example
- ROS packages, nodes, topics and messages
- ROS utilities
- Summary

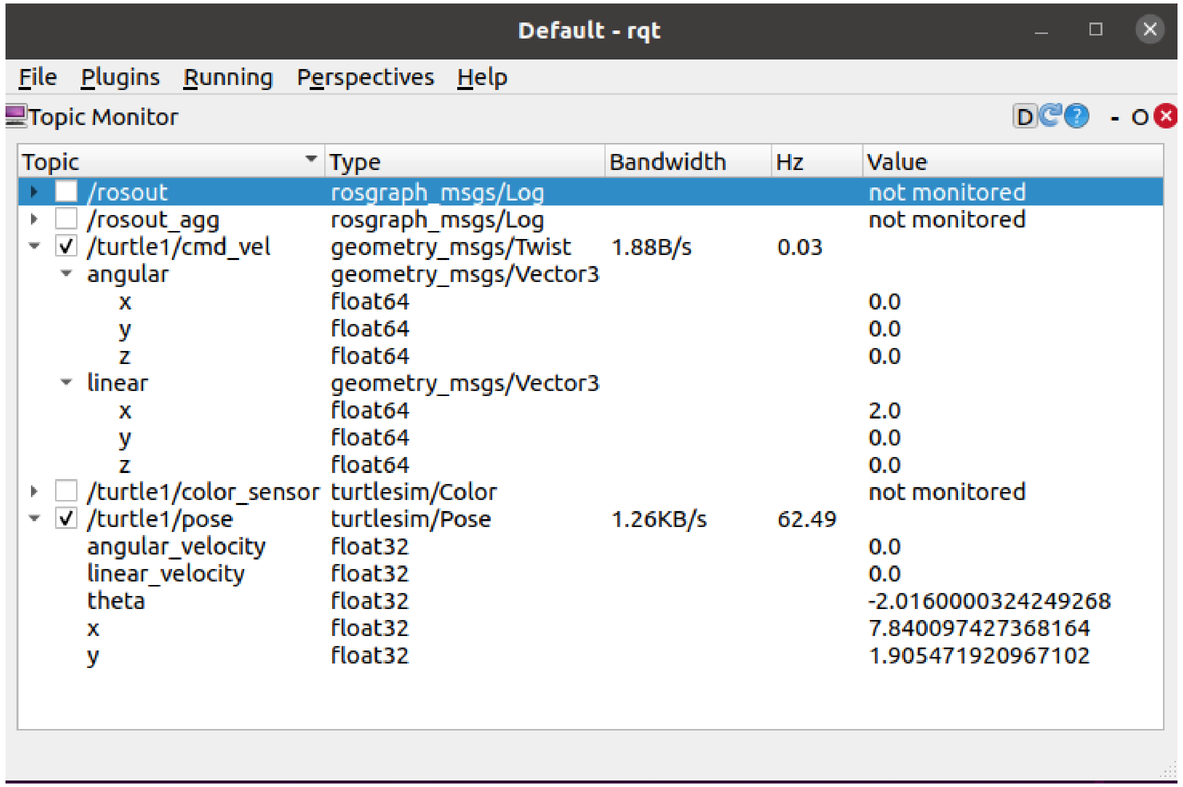
ROS Utilities – rqt_graph

- rqt_graph provides a GUI plugin for visualizing the ROS computation graph

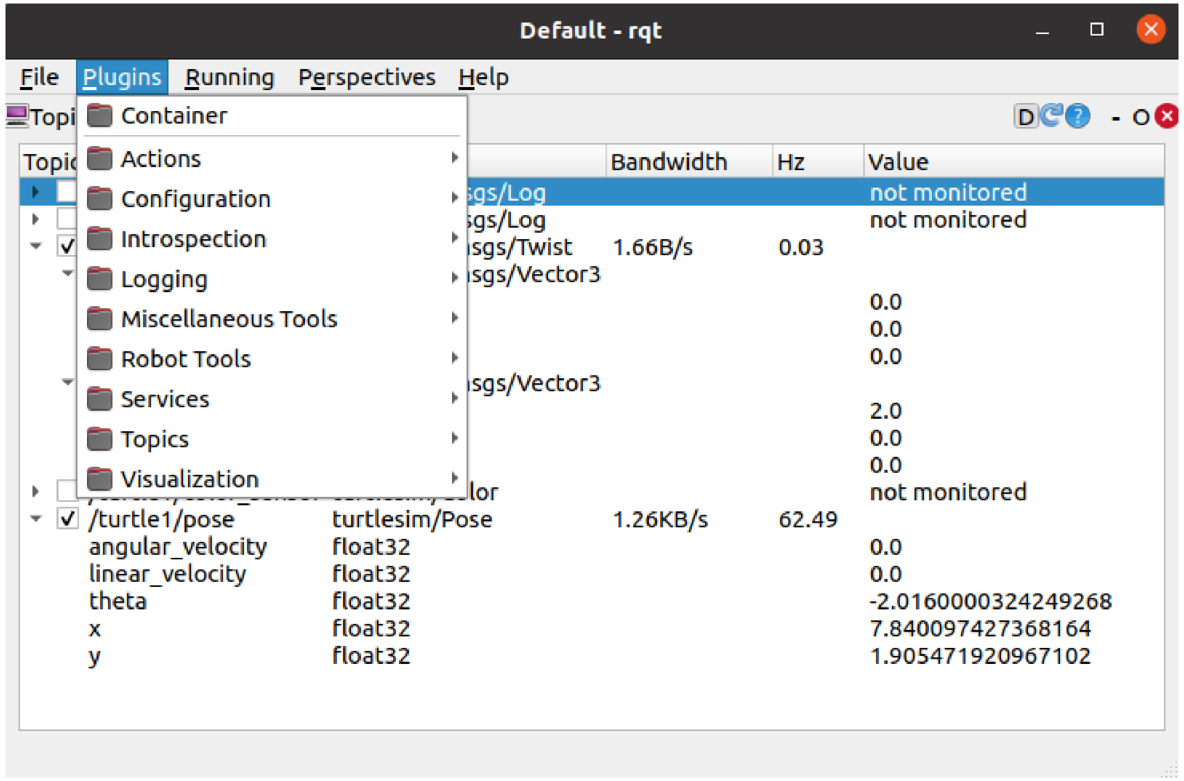


ROS Utilities – rqt

- rqt is a Qt-based framework for GUI development for ROS
- Supports many plug-ins for monitoring topics



Topic	Type	Bandwidth	Hz	Value
/rosout	rosgraph_msgs/Log			not monitored
/rosout_agg	rosgraph_msgs/Log			not monitored
✓ /turtle1/cmd_vel	geometry_msgs/Twist	1.88B/s	0.03	
angular	geometry_msgs/Vector3			
x	float64			0.0
y	float64			0.0
z	float64			0.0
linear	geometry_msgs/Vector3			
x	float64			2.0
y	float64			0.0
z	float64			0.0
/turtle1/color_sensor	turtlesim/Color			not monitored
✓ /turtle1/pose	turtlesim/Pose	1.26KB/s	62.49	
angular_velocity	float32			0.0
linear_velocity	float32			0.0
theta	float32			-2.0160000324249268
x	float32			7.840097427368164
y	float32			1.905471920967102



Topic	Type	Bandwidth	Hz	Value
	sgs/Log			not monitored
	sgs/Log			not monitored
✓	sgs/Twist	1.66B/s	0.03	
	sgs/Vector3			
				0.0
				0.0
				0.0
	sgs/Vector3			
				2.0
				0.0
				0.0
	lor			not monitored
✓ /turtle1/pose	turtlesim/Pose	1.26KB/s	62.49	
angular_velocity	float32			0.0
linear_velocity	float32			0.0
theta	float32			-2.0160000324249268
x	float32			7.840097427368164
y	float32			1.905471920967102

Elements of ROS – Other Resources

- ROS is developed by a community of users
- There are many resources on-line that help
  - http://wiki.ros.org/
  - http://wiki.ros.org/ROS/StartGuide
  - http://wiki.ros.org/ROS/Tutorials

Overview

- Introduction to ROS
- ROS Example
- ROS packages, nodes, topics and messages
- ROS utilities
- Summary

Summary

- We complete an introduction to ROS

- We ran our first ROS nodes and discussed the concepts of ROS packages, topics and messages

- We also introduced some usage ROS commands and utilities

- We will spend the rest of the course working in ROS. Becoming comfortable with these concepts is crucial for success!