COMP 417 – Tutorial 1

September 19, 2019

ROS Overview

What is ROS?

ROS is an open-source, meta-operating system for your robot. It provides the services you would expect from an operating system, including:

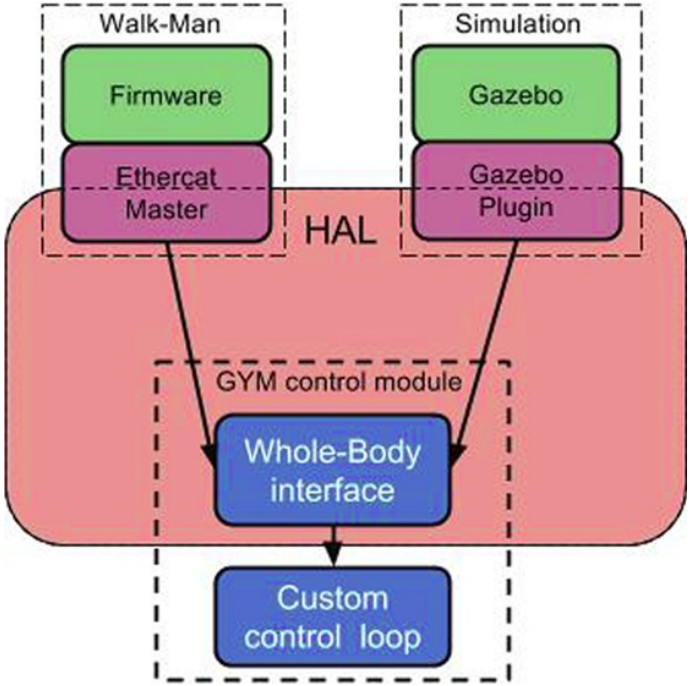
- **hardware abstraction**
- **message-passing between processes**
- low-level device control
- implementation of commonly-used functionality
- package management

Source: http://wiki.ros.org/ROS/Introduction

Hardware Abstraction

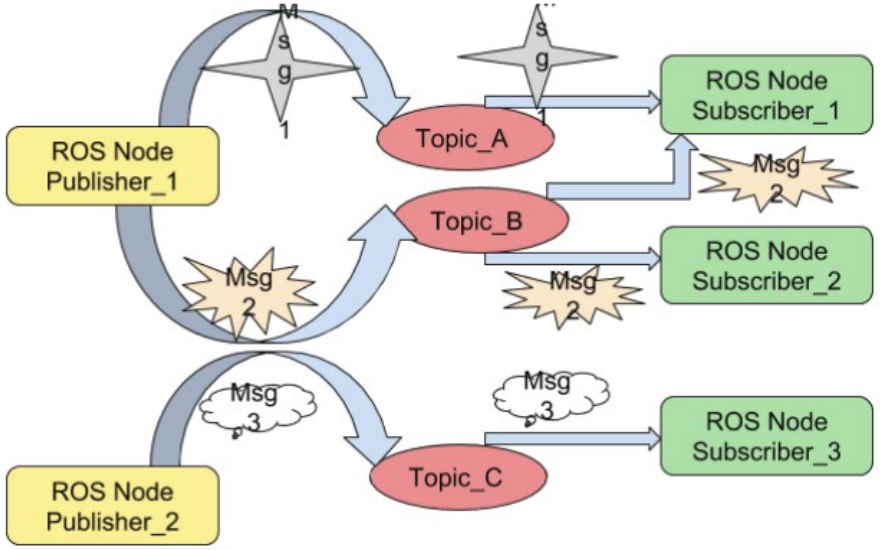
Goal:

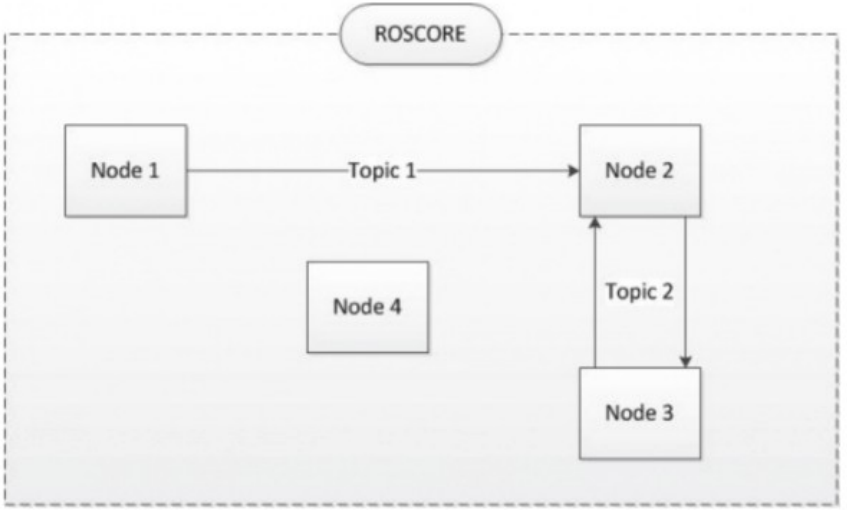
- User code should not depend on specific hardware details.
- In ROS, code that we write portal between simulation and real robots.



Message Passing and Node Graph (1/2)

- ROS application made up of collection of **nodes.**
- Nodes exchange messages in **decoupled** fashion.
- Only require knowledge of common message types.

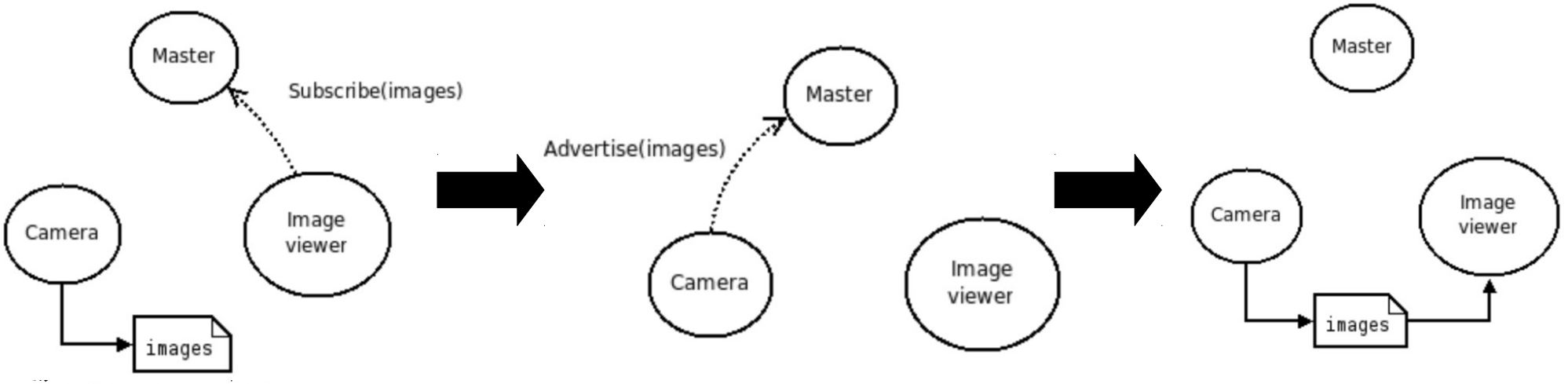




Master Node

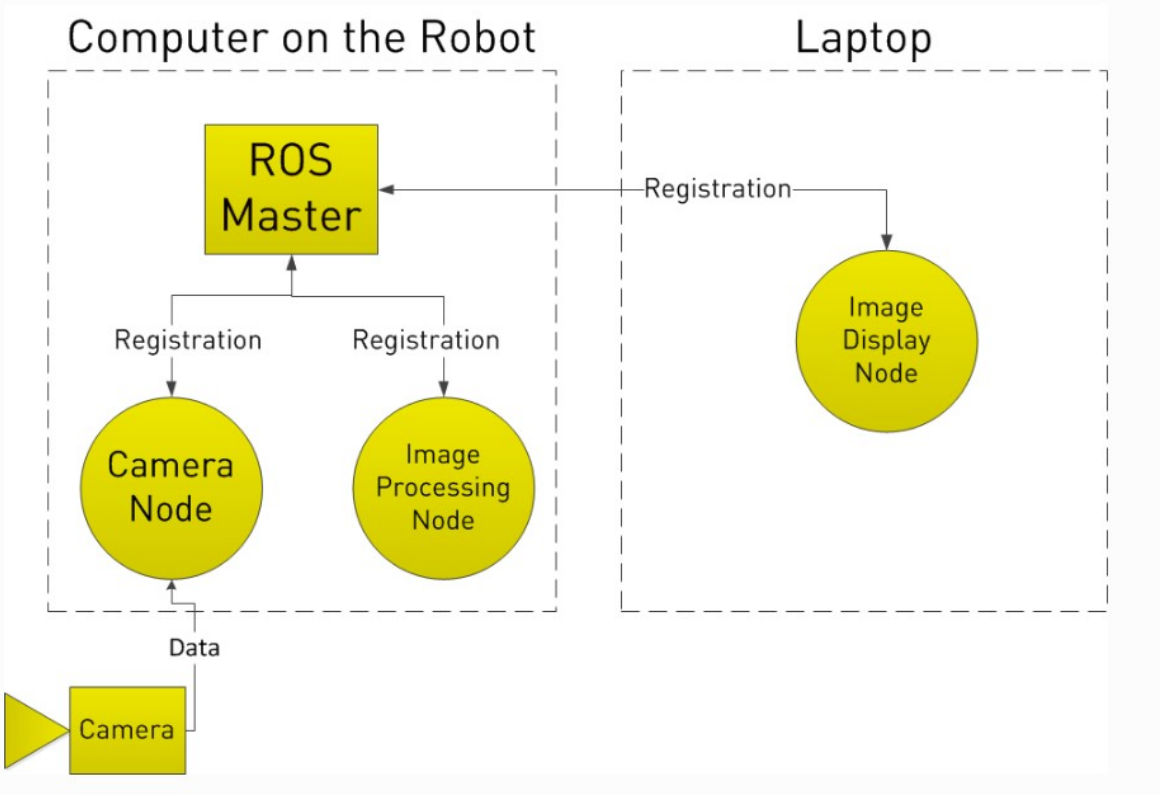
- Master node responsible for registry and lookup of other nodes
- Enables other ROS nodes to locate eachother
- Once nodes have located eachother, communiate peer-to-peer
- **roscore** command starts master node

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



Nodes on Multiple Computers

- Nodes can exist on multiple machines
- But only 1 master node manages entire environment



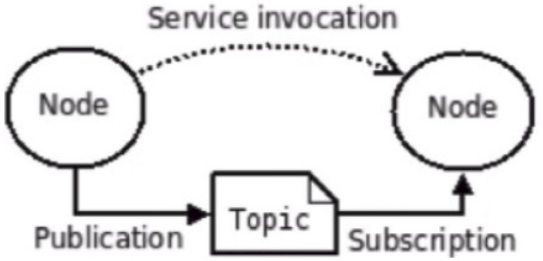
Node Components + Communication Paradigms

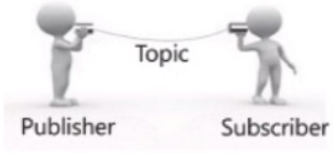
- Node is basic ROS process
- Internally, node can host 1) topics 2) services 3) actions client/server

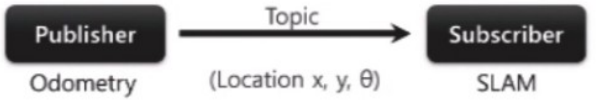
Communication Type	Description	Best used for
1) Topic	Publisher-Subscriber	Constant streams of data.
2) Service	Blocking Request-Response	One-off short-running tasks
3) Action Client/Server	Async Request-Response	One-off long running tasks

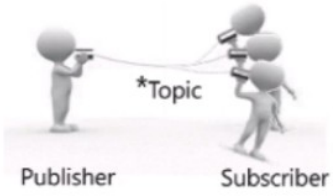
Topics

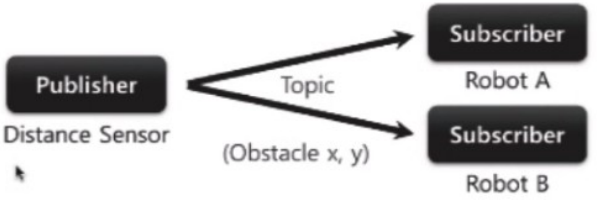
- Publisher / Subscriber communication paradigm
- Topic forms a pipe between publishers and subscribers
- Multiple subscribers possible





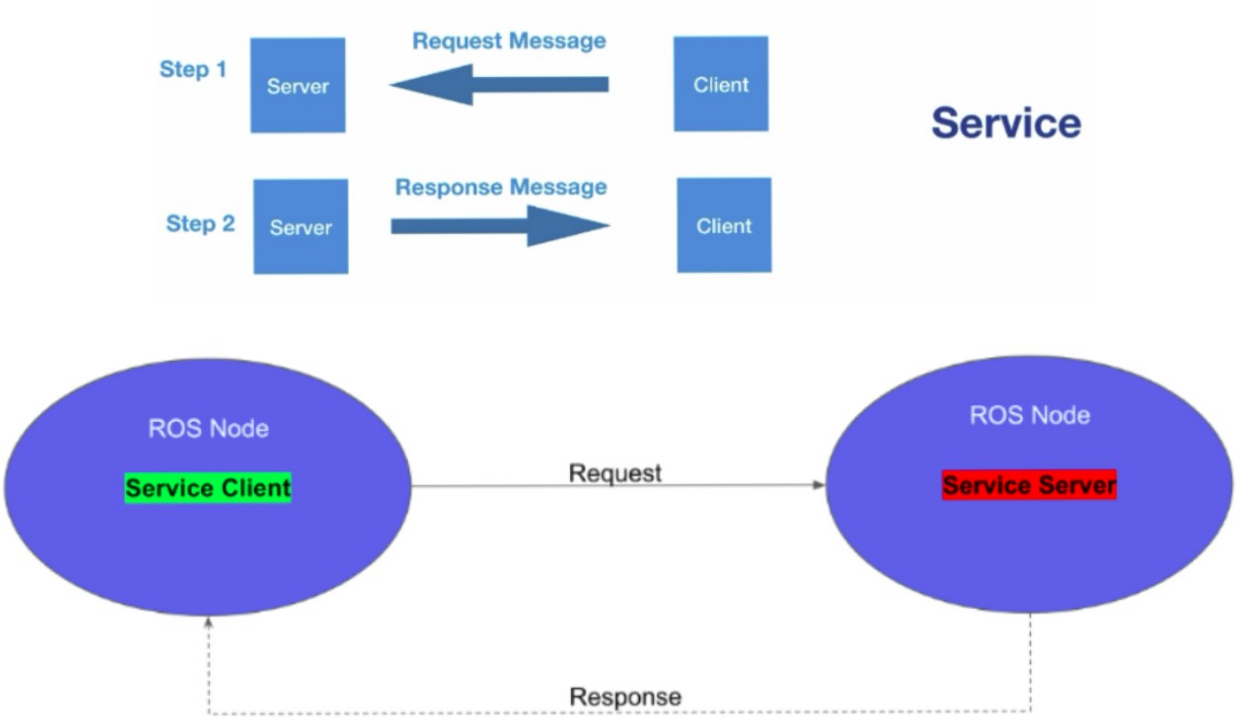






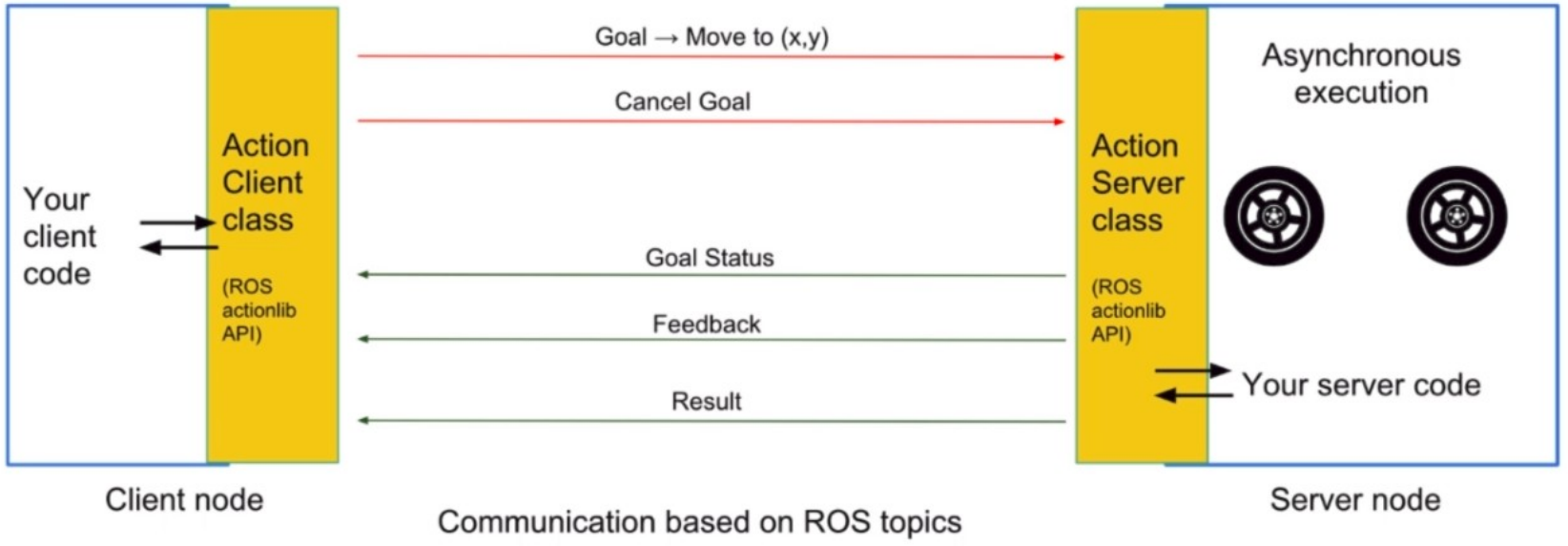
Services

- Request / Response communication paradigm
- Request will **block** will response is received



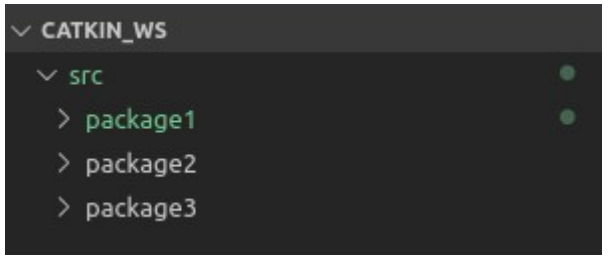
Action Clients/Servers

- Actions trigger asynchronous tasks
- Client code receives async, non-blocking callbacks with status updates.

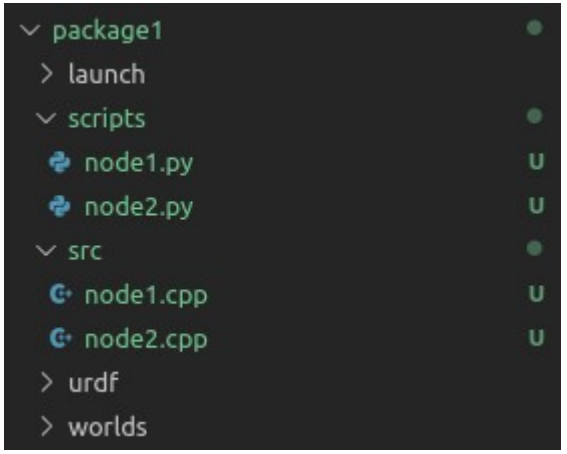


Anatony of a ROS Project

Workspace folders



Package folders



catkin_ws: Workspace folder where packages (projects) are stored

- Similar to Eclipse Workspace
- Packages can be implemented in Python or C++

Folder	Description
scripts or python	Stores python scripts and nodes
src	Stores c++ files and nodes
urdf	Markup for graphical models
worlds	Markup for simulation worlds
launch	Stores launch files for running nodes

Subscriber / Publisher Example

Publisher Node

```python
import rospy
from std_msgs.msg import String

if __name__ == '__main__':

    rospy.init_node('string_publisher', anonymous=True)
    rospy.loginfo("String Publisher node starting")
    message_to_send = "Default Message"
    msg_pub = rospy.Publisher('/msg', String, queue_size=1)

    rate = rospy.Rate(10)
    while not rospy.is_shutdown():
        msg = String()
        msg.data = message_to_send
        rospy.loginfo("Publisher sending %s" % msg.data)
        msg_pub.publish(msg)
        rate.sleep()
```

Subscriber Node

```python
import rospy
from std_msgs.msg import String


def on_msg_received(msg):
    rospy.loginfo("Subscriber got %s" % msg.data)


if __name__ == '__main__':

    rospy.init_node('string_subscriber', anonymous=True)
    rospy.loginfo("String Subscriber node starting")
    msg_pub = rospy.Subscriber('/msg', String, on_msg_received)

    while not rospy.is_shutdown():
        rospy.spin()
```

Rosrun

- Used to run individual nodes

  `rosrun` `<package> <node>`

- **Example:** run 1) master node, 2) publisher node, 3) subscriber node

```
roscore

rosrun ros_tutorials publisher.py

rosrun ros_tutorials subscriber.py
```

Roslaunch

Motivation:

- Running nodes 1-by-1 cumbersome
- Can run many nodes in batch using **launch** files

roslaunch <package> <node>

File path: /launch/pub_sub.launch

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
    <node name="string_publisher_node" pkg="ros_tutorial" type="string_publisher_node.py" output="screen">
    </node>
    <node name="string_subscriber_node" pkg="ros_tutorial" type="string_subscriber_node.py" output="screen">
    </node>
</launch>
```

Roslaunch with Parameters

- Can run parameters to launch files

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
    <node name="string_publisher_node" pkg="ros_tutorial" type="string_publisher_node.py" output="screen">
        <param name="msg_to_publish" type="string" value="Hello World"/>
    </node>
    <node name="string_subscriber_node" pkg="ros_tutorial" type="string_subscriber_node.py" output="screen">
    </node>
</launch>
```

- Read in code with **get_param**

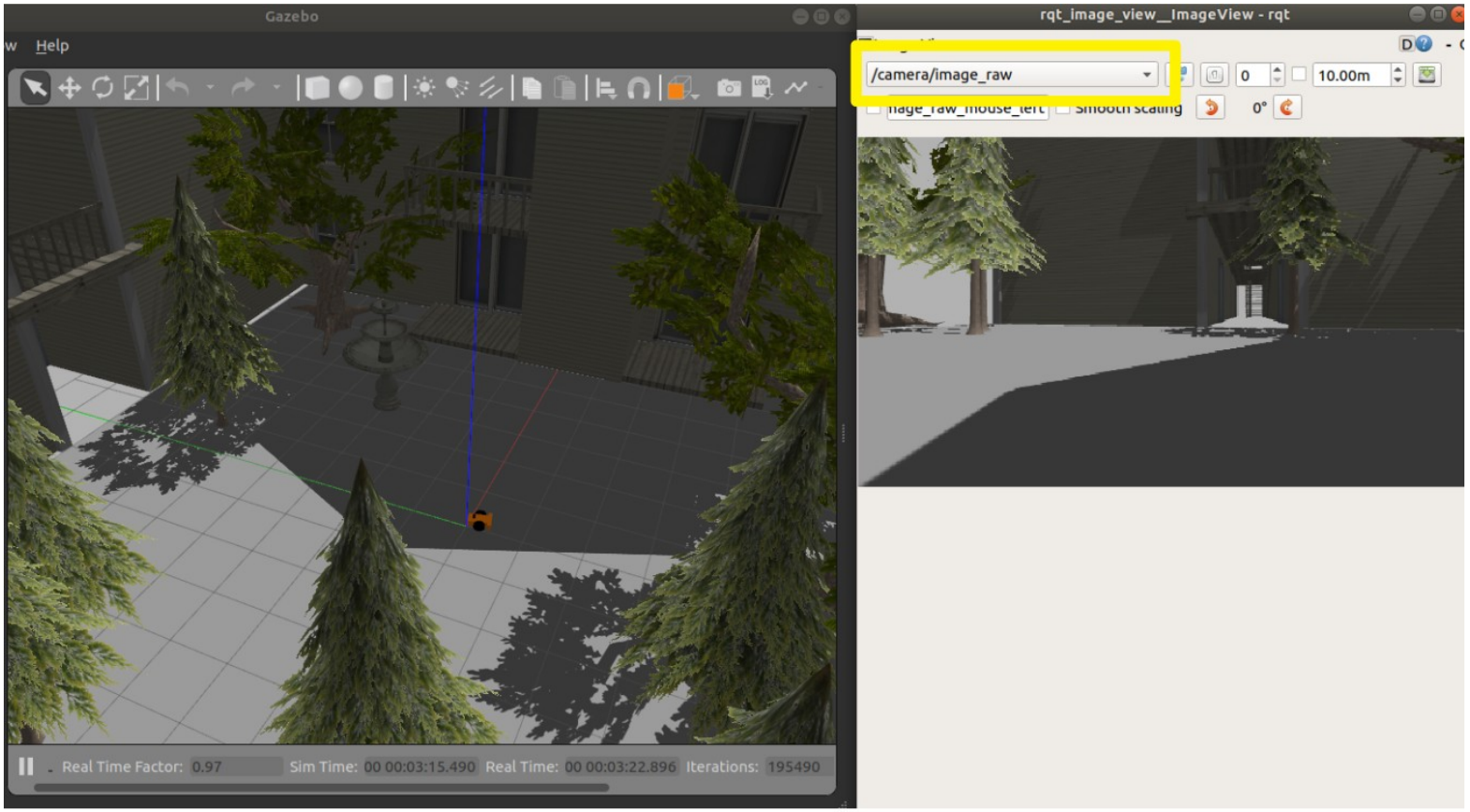
```
rospy.init_node('string_publisher', anonymous=True)
message_to_send = rospy.get_param("~msg_to_publish", "Default Message")
```

Tools for Debugging and Data Analysis

- rqt_image_view
- rqt_graph
- rqt_plot
- RVIZ

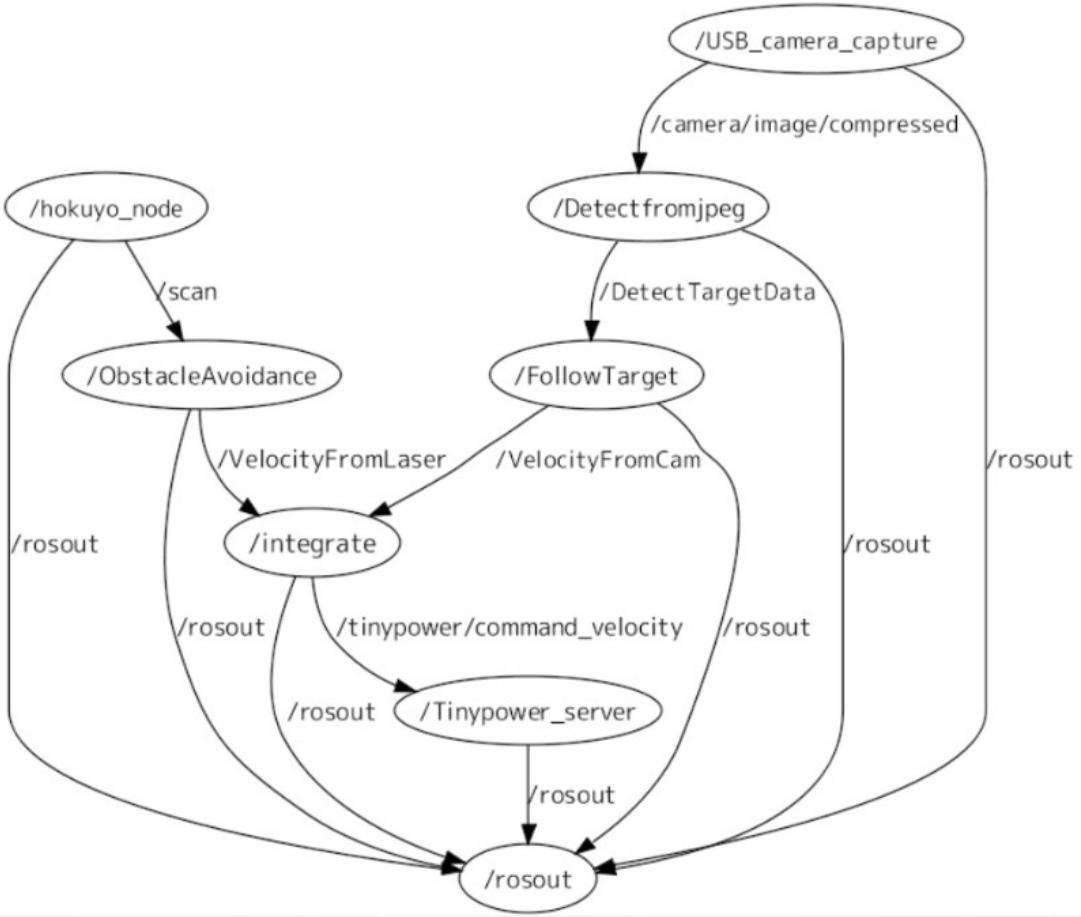
rqt_image_view

- Visualize image topics.



rqt_graph

- Shows list of nodes and their connections



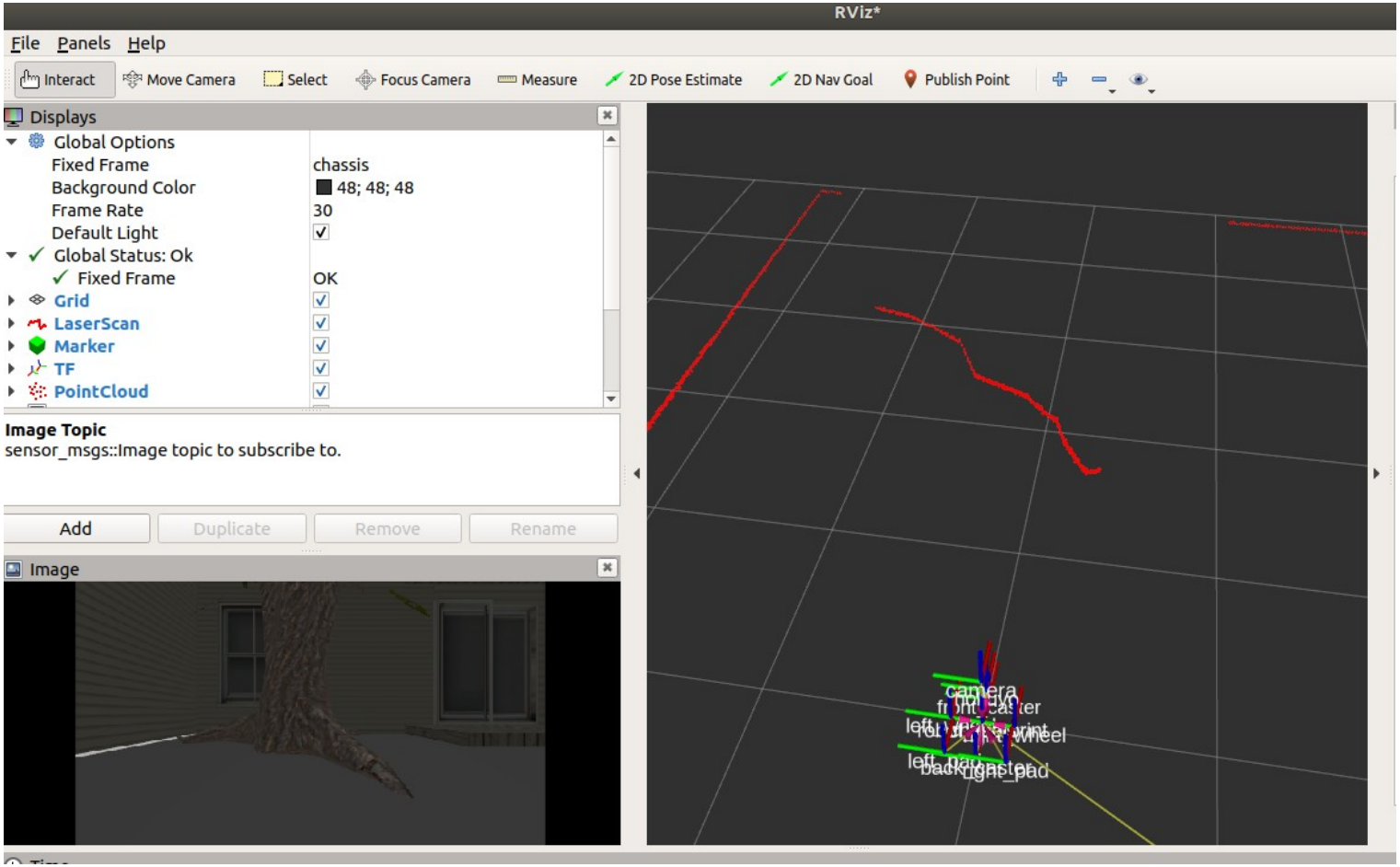
rqt_plot

- Graphs data from one or more topic fields
- Example: Graph x,y fields of trutle1/pose topic over time.

```
rqt_plot turtle1/pose/x, turtle1/pose/y
```

RVIZ (ROS Visualization)

- General purpose 3D visualization for ROS
- Can visualize lidar scans, coordinate frames, point clouds, etc.

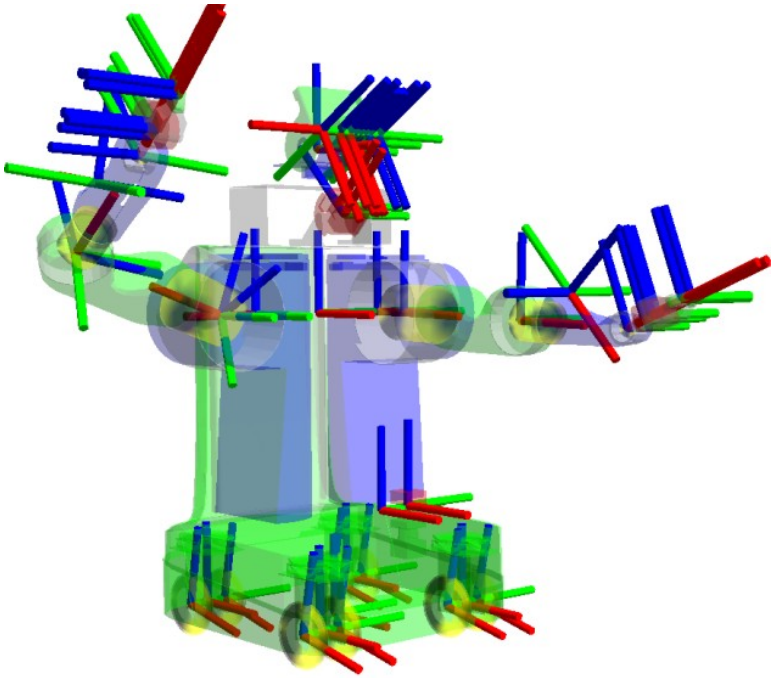


TF Package for Coordinate Frame Transforms

- TF package handles transforms between coordinate frames over time
- tf_echo: Prints updated transforms in console
- Example:

```
rosrun tf tf_echo [reference_frame]
[target_frame]
```

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



Rosbag

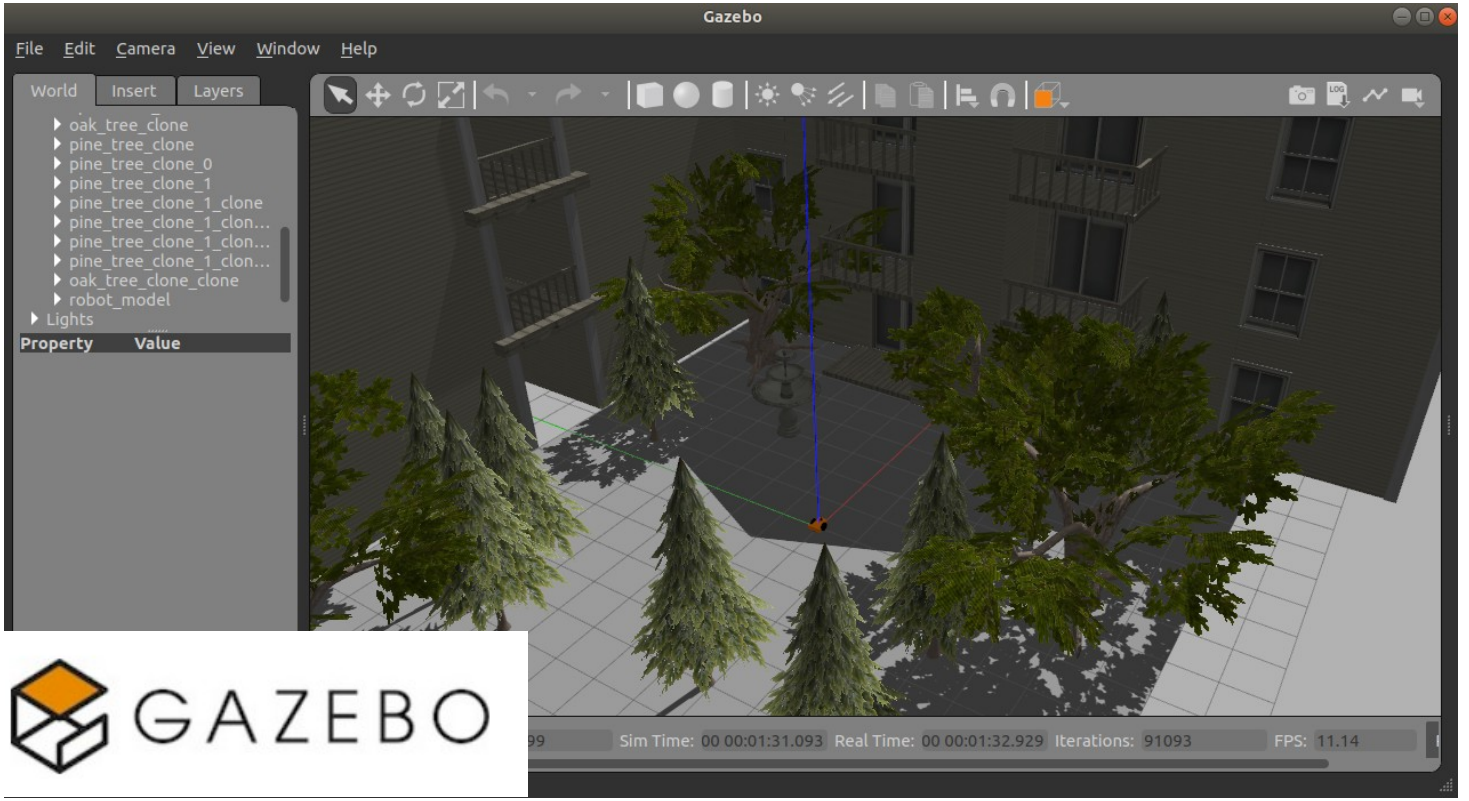
Motivation:

- Real-world robotics experiments subject to high amounts of variability
- Want re-run (replay) experiments deterministically

Record all topics	`rosbag record -a`
Record topics to output file	`rosbag record [topics] -o <output_file.bag>`
Play back recorded data	`rosbag play <input_file.bag>`

Gazebo Simulator

- ROS alone does not do physics-based simulation
- ROS often used in conjuction with **Gazebo** for simulation
- Run empty gazebo world with command: `gazebo`



Definining Graphical Models

- Graphical models defined in xml style format
- Commonly found in urdf/ (for robot models) or world/ (for environments) project folders.

Format	Description
urdf	Unified Robot Definiton Format • Typically only used to represent robot models (not environments).
sdf	Simulation Definition Format • Can represent robot models and environments.

Additional Tutorials

Resource	Link
ROS Wiki	http://wiki.ros.org/ROS/Tutorials
ClearPath Tutorials	http://www.clearpathrobotics.com/assets/guides/ros/
Husarion Tutorials	https://husarion.com/tutorials/ros-tutorials/1-ros-introduction/
Ohio State Nodes (See ROS Tutorial Section)	http://www2.ece.ohio-state.edu/~zhang/RoboticsClass/
Github Repos for Udacity Robotics Nanodegree	https://github.com/topics/udacity-robotics-nanodegree
Udemy Courses	https://www.udemy.com/courses/search/?src=ukw&q=ros