Course Updates

- Website set-up at https://ae640a.github.io/
  - Lectures uploaded
- Canvas invite (for assignments & submissions) will be sent once course list (after add-drop) is available
- Attendance: 70% Minimum
- Last 2 lectures introduced you to the kind of possible projects:
  - Course Project Deadlines:
    - Selection (next 2 weeks)
    - Abstract submission with timeline (by end of January)
    - Hard deadlines to be conveyed via course web-page/email
- Upcoming lectures will focus more on mathematics/algorithms

Introduction to Robot Operating System (ROS)

AE640A - Autonomous Navigation
Presenter: Aalap Shah
Slides: Mayank Mittal, Aalap Shah

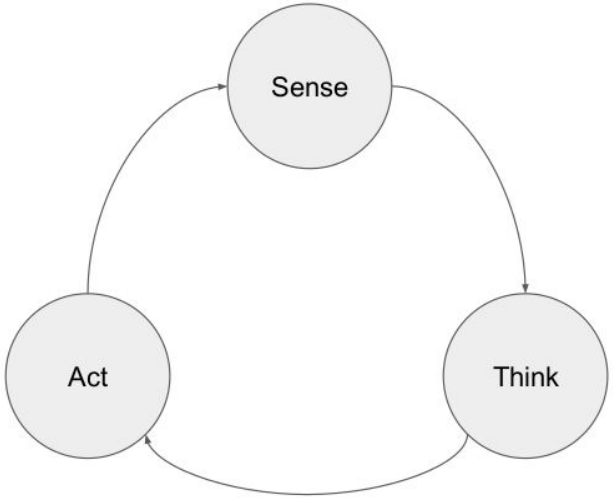
12th January, 2019

Lecture Outline

- Motivation for using ROS
- ROS features overview
- ROS Communication Layer
  - Nodes, Messages, Topics, Parameters
  - Demo 1: Image Viewer
- ROS Ecosystem
  - ROS Packages
  - *Catkin* build system
- Tools in ROS
  - RViz, rqt (Demo 2: IMU)
  - ROS bags (Demo 3: Recording Image Data)
- Content not covered: Services, Actions (not necessary for this course)

Motivation: Robotic Systems

- Simple model of a robot:
  - Sensing → Computation → Actuation
  - Can be implemented sequentially for simple systems (eg: servo motor control using Arduino)



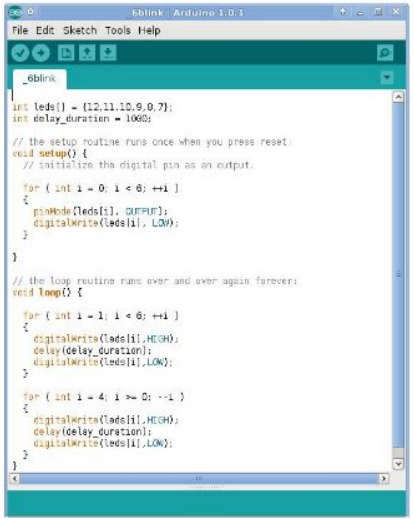


Illustration by: IGVC IITK

Motivation: Robotic Systems

- Complex, parallel model of a robot:
  - Multiple actions based on multiple sensors
- Requirements:
  - Multiple programs (with different inputs and outputs) should run concurrently
  - Can still achieve this sequentially using an arduino with some effort

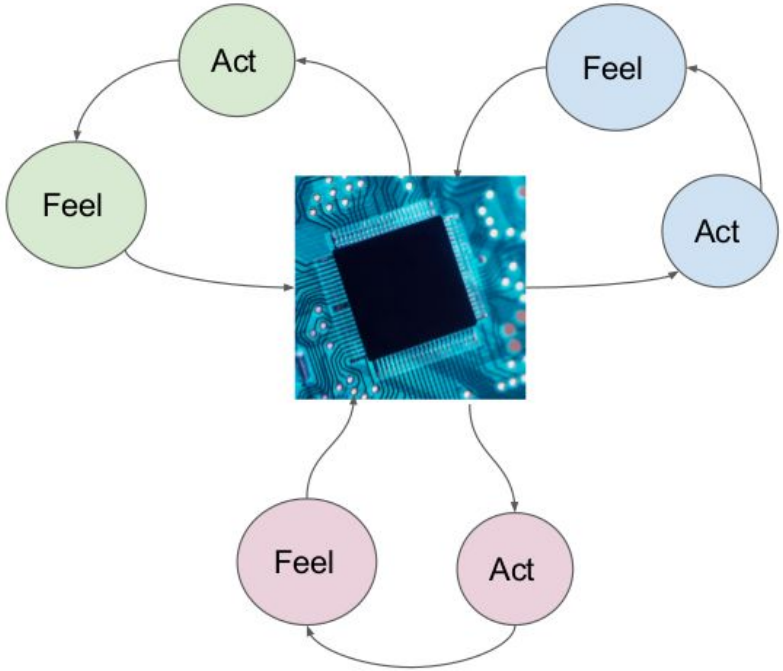


Illustration by: IGVC IITK

Camera

IMU

Laser scanner

Motor and Encoder

TEAM IGVC

Robot

How to perform multiple ***inter-related*** sensing and actuation tasks?

GPS

Motivation: Robotic Systems

- More complex, parallel model of a robot:
  - Multiple **inter-related** actions based on multiple sensors
- Requirements:
  - Multiple programs should run concurrently
  - Inter-process communication

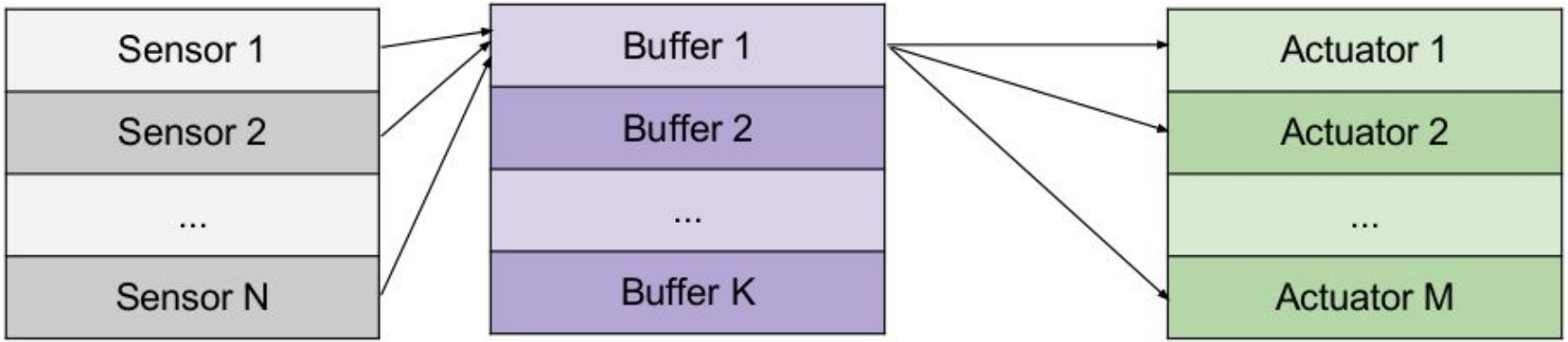


Illustration by: IGVC IITK

Motivation: Robotic Systems

- Yet more complexity: Swarm robotics
  - Sensors and actuators distributed over multiple computers
- A software that satisfies these requirements?
  - Multiple inputs and outputs (preferable to have drivers for each type of hardware)
  - Multiple programs should run concurrently
  - Inter-process communication
  - Inter-machine communication

Motivation: Robotic Systems

- Yet more complexity: Swarm robotics
  - Sensors and actuators distributed over multiple computers
- A software that satisfies these requirements?
  - Multiple inputs and outputs (preferable to have drivers for each type of hardware)
  - Multiple programs should run concurrently
  - Inter-process communication
  - Inter-machine communication

An operating system!

Motivation: Robotic Systems

- Yet more complexity: Swarm robotics
  - Sensors and actuators distributed over multiple computers
- A software that satisfies these requirements?
  - Multiple inputs and outputs (preferable to have drivers for each type of hardware)
  - Multiple programs should run concurrently
  - Inter-process communication
  - Inter-machine communication

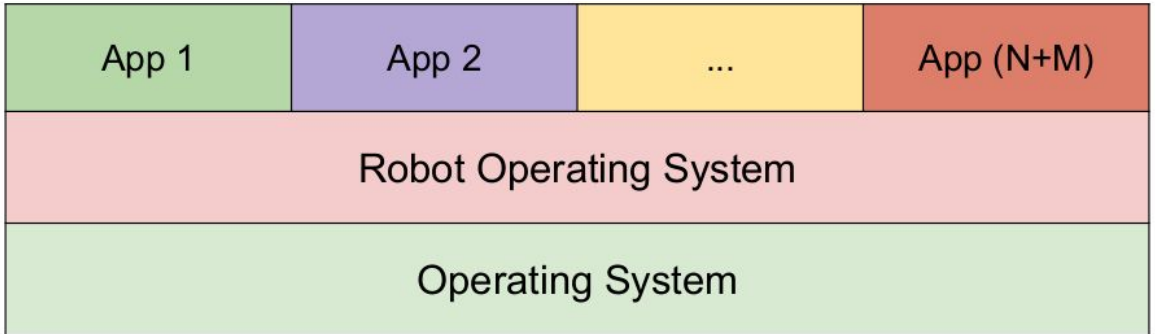


Illustration by: IGVC IITK

What is ROS?

- A “meta” operating system for robots
  - Communication layer
- A collection of tools for:
  - Software building - catkin build system
  - Debugging - Command-line tools
  - Data Visualization - RViz, rqt
- A language-independent architecture (there are libraries for C++, python, lisp, java, and more)

Slide Credit: Lorenz Mösenlechner, TU Munich

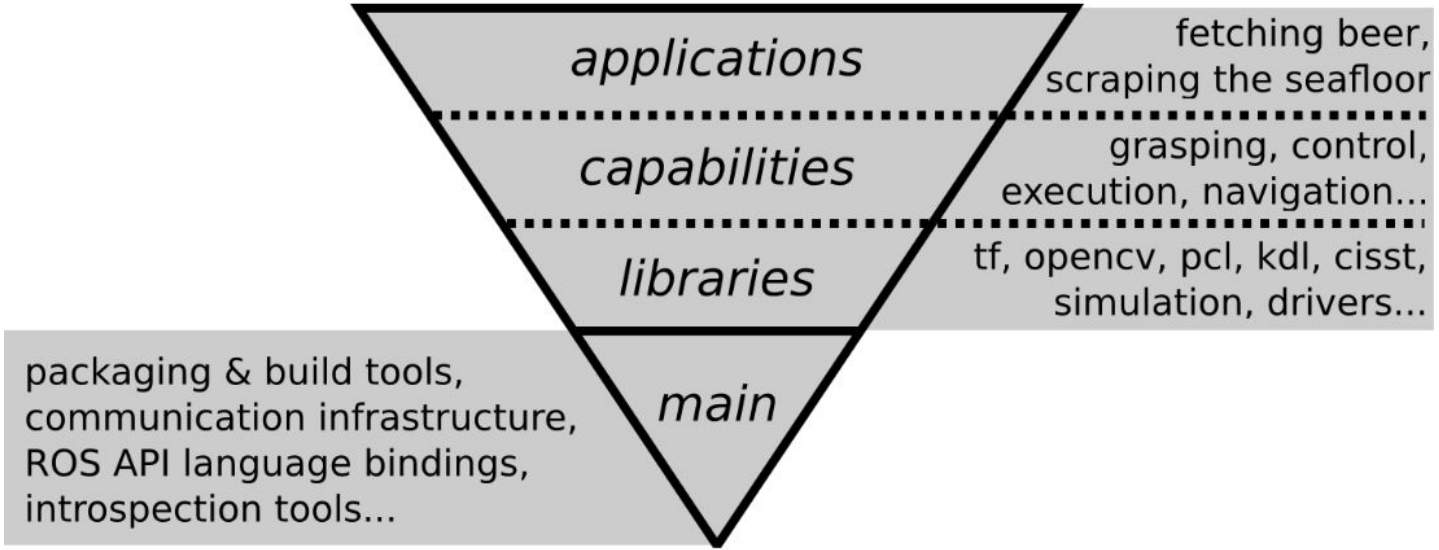
What is ROS not?

- An actual operating system
  - Does not have disk management, user access control, security, etc.
- A programming language
  - Rather it provides libraries for common programming languages like C++, Python, etc.
- A programming environment/IDE
- A hard real-time architecture (like an RTOS)

Slide Credit: Lorenz Mösenlechner, TU Munich

What does ROS get you?

All levels of development



Slide Credit: Lorenz Mösenlechner, TU Munich

ROS Communication Layer : Terminology

- In ROS terminology, it is common to use sentences like:
  - A *node* N1 *publishes* a *message* M on a *topic* T.
  - Another *node* N2 *subscribes* to *topic* T, receiving the *message* M.
- In layman terms:
  - Node = program
  - Message = data (in a specific format like image, point, etc)
  - Topic = a place where messages are sent to and received from
  - Publishing = sending data to a topic
  - Subscribing = trying to receive data from a topic

ROS Communication Layer : ROS Core

- **ROS Master**
  - Centralized Communication Server based on XML and RPC
  - Negotiates communications between nodes
- **Parameter Server**
  - Stores persistent configuration parameters such as camera parameters, robot dimensions, etc.
- **Rosout**
  - Network based `*stdout*` for human readable messages.

Slide Credit: Lorenz Mösenlechner, TU Munich

ROS Communication Layer : Graph Resources

- **Nodes**
  - Processes distributed over the network.
  - Serves as source and sink for the data sent over the network
- **Parameters**
  - Data stored on the parameter server.
- **Topics**
  - Asynchronous many-to-many communication stream

Slide Credit: Lorenz Mösenlechner, TU Munich

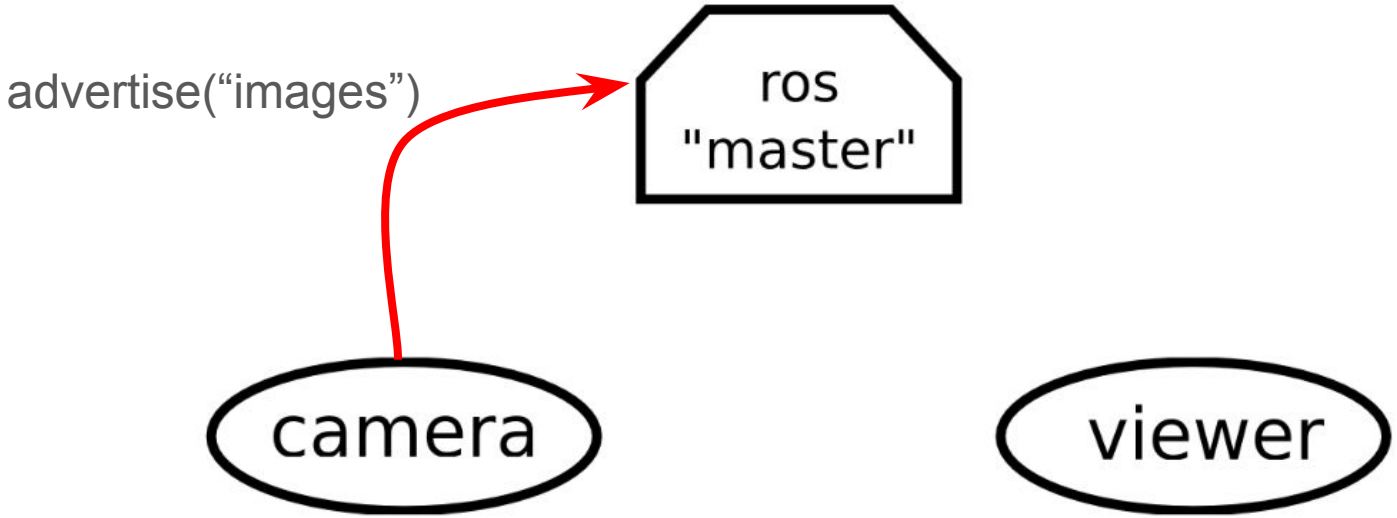
Asynchronous Distributed Communication: Example

Interfaces with the camera hardware and reads the data transmitted by the sensor

Used to display images

Image Courtesy: Lorenz Mösenlechner, TU Munich

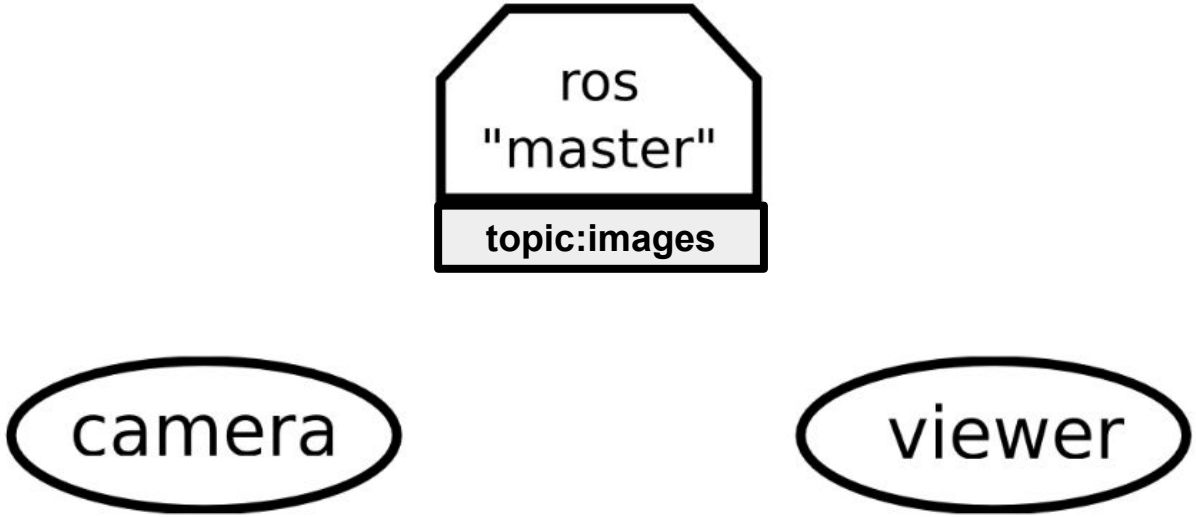
Asynchronous Distributed Communication: Example



camera node is run. It starts advertising the data it has received

Image Courtesy: Lorenz Mösenlechner, TU Munich

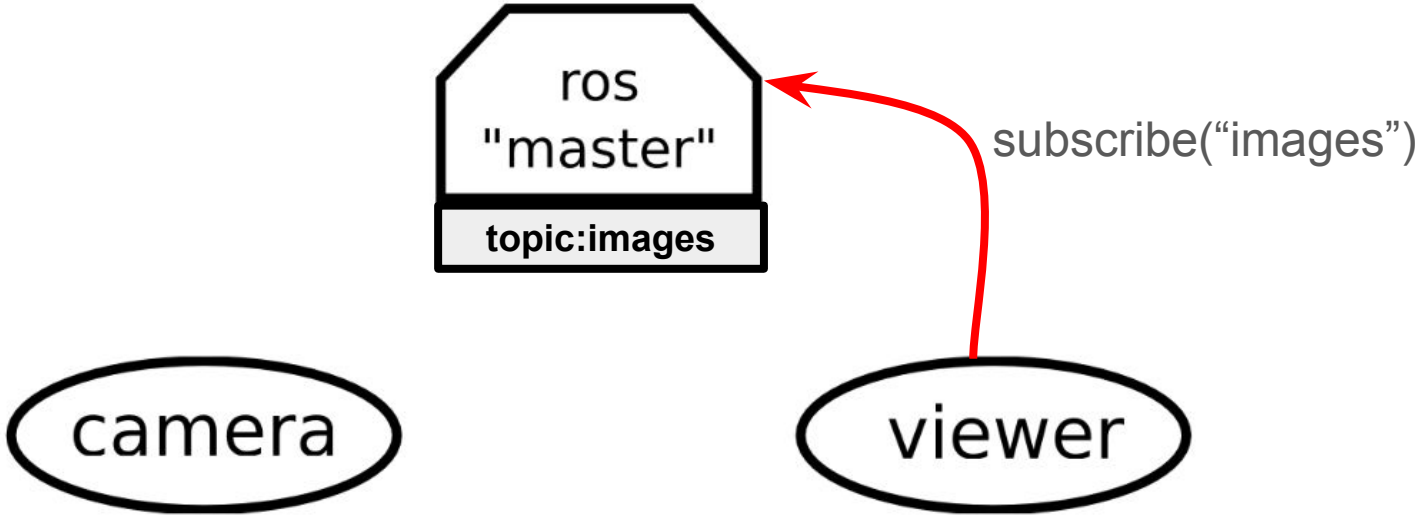
Asynchronous Distributed Communication: Example



master registers the topic with name **images**

Image Courtesy: Lorenz Mösenlechner, TU Munich

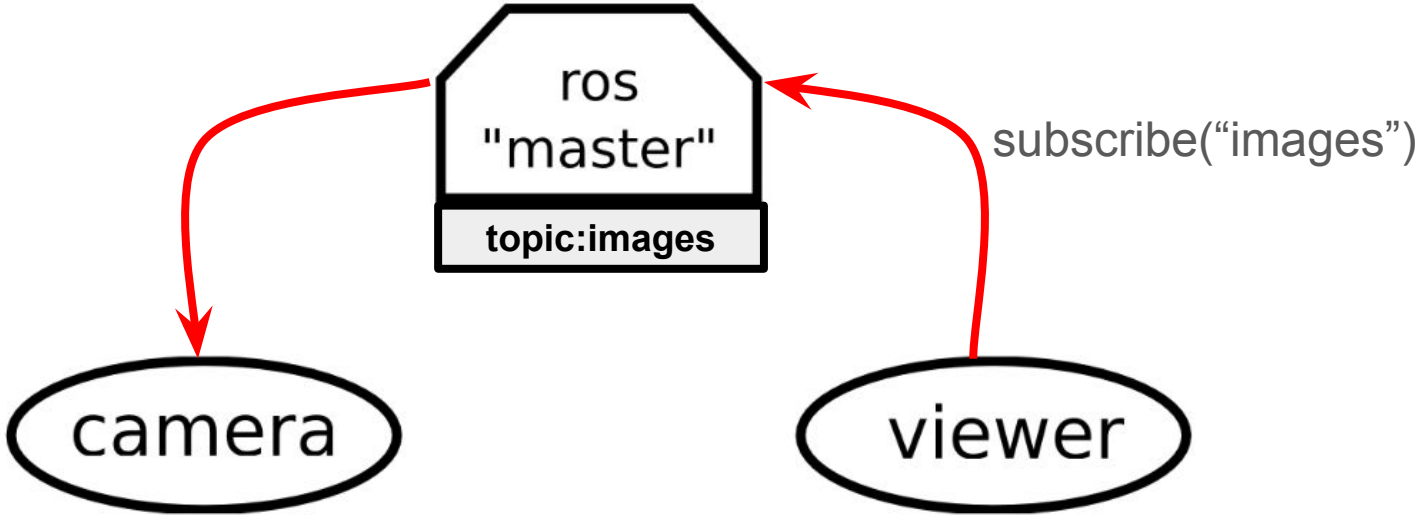
Asynchronous Distributed Communication: Example



viewer node is run. It asks for data being published in topic with name **images**

Image Courtesy: Lorenz Mösenlechner, TU Munich

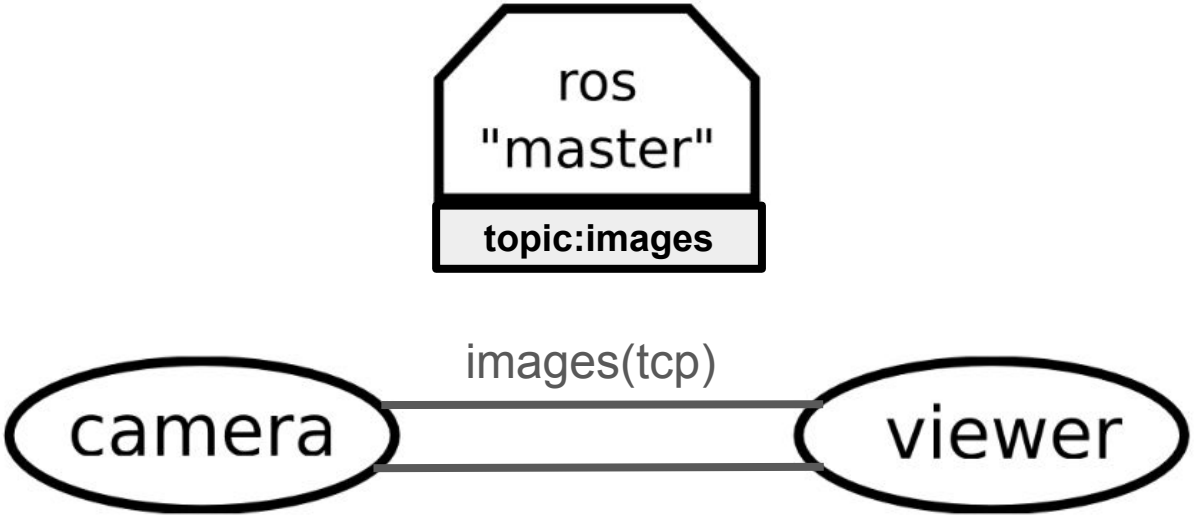
Asynchronous Distributed Communication: Example



master ‘connects’ the *viewer* node to the *camera* node.

Image Courtesy: Lorenz Mösenlechner, TU Munich

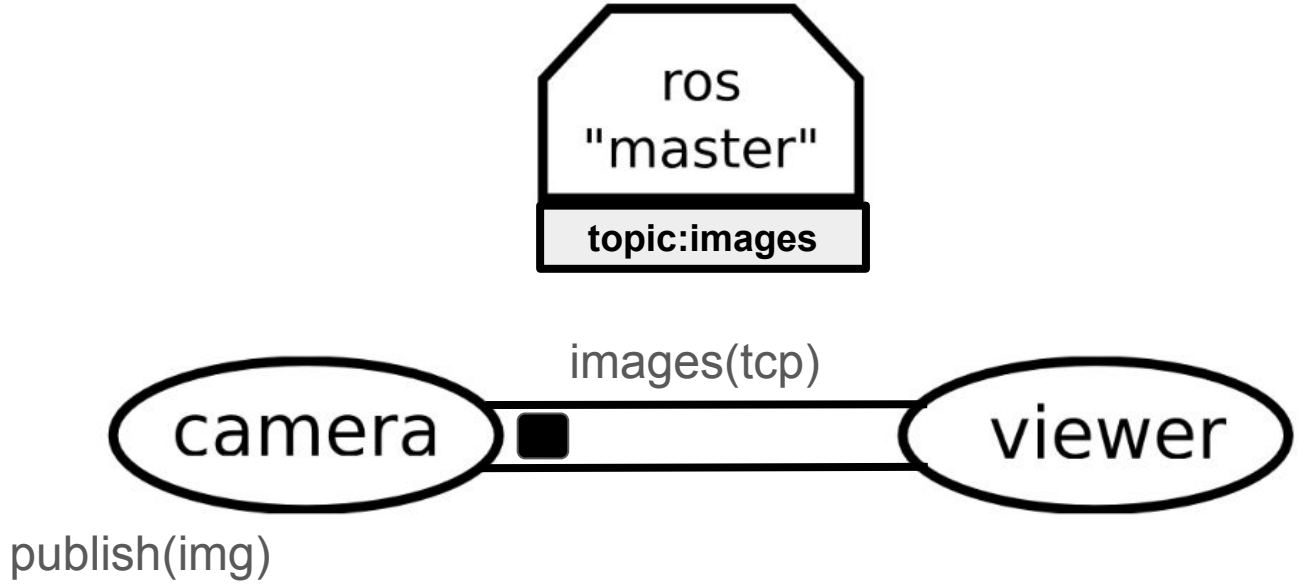
Asynchronous Distributed Communication: Example



master 'connects' the *viewer* node to the *camera* node.

Image Courtesy: Lorenz Mösenlechner, TU Munich

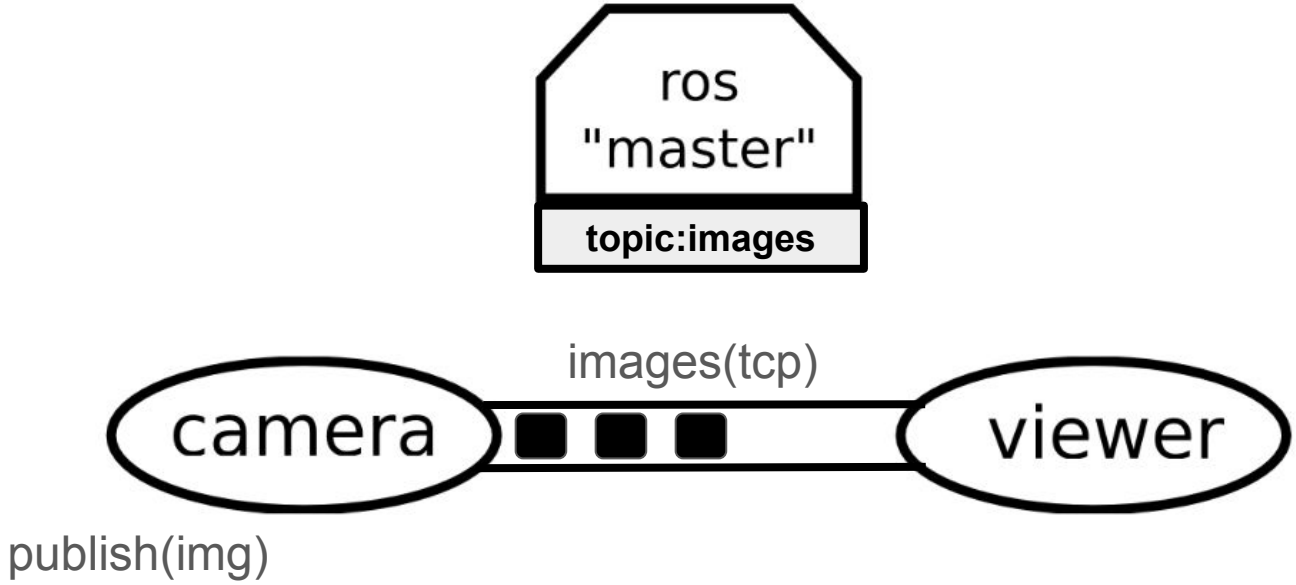
Asynchronous Distributed Communication: Example



camera node sends the data to the *viewer* node using TCP/IP based protocol

Image Courtesy: Lorenz Mösenlechner, TU Munich

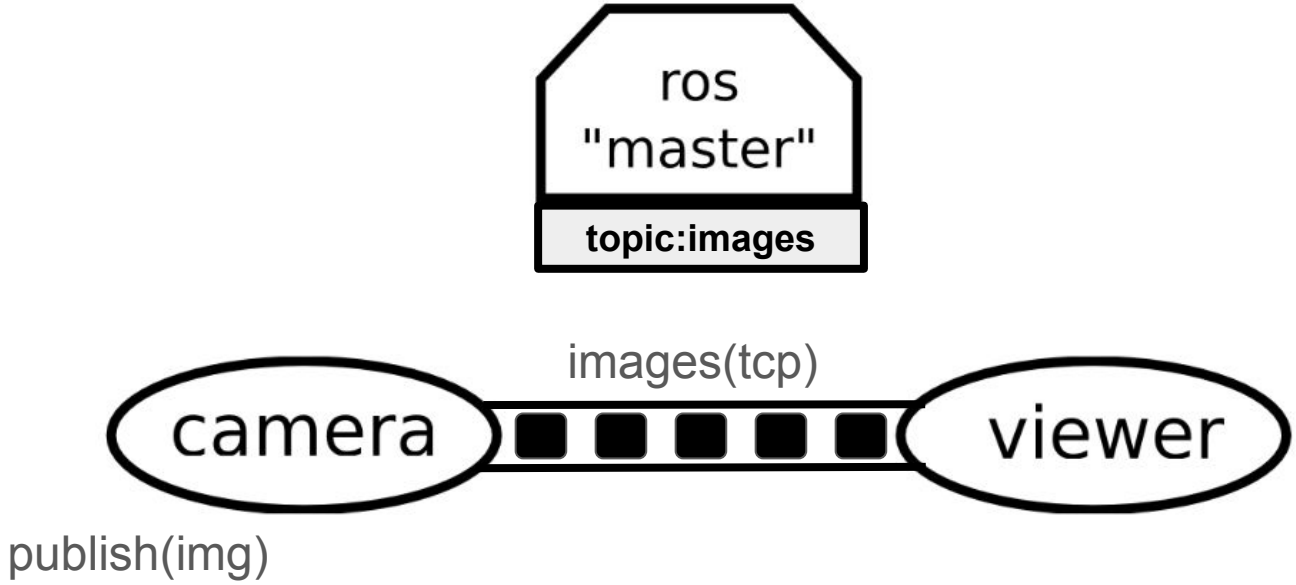
Asynchronous Distributed Communication: Example



camera node sends the data to the *viewer* node using TCP/IP based protocol

Image Courtesy: Lorenz Mösenlechner, TU Munich

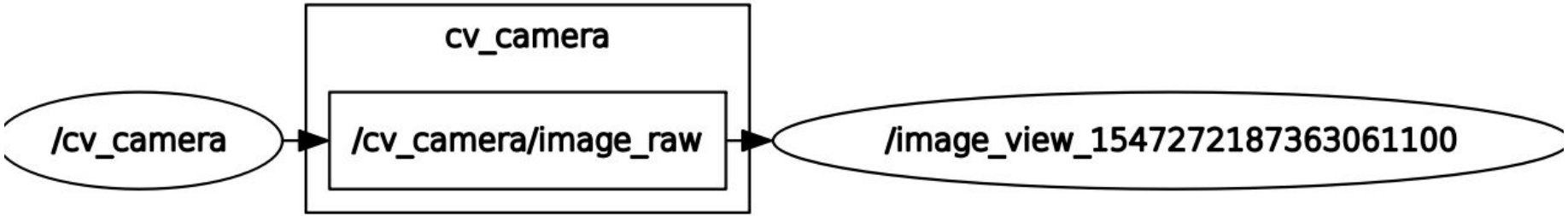
Asynchronous Distributed Communication: Example



camera node sends the data to the *viewer* node using TCP/IP based protocol

Image Courtesy: Lorenz Mösenlechner, TU Munich

Live Demo 1: Image Viewer



ROS Master

- Manages the communication between nodes
- Every node registers at startup with the master

Start a master with

```
$ roscore
```

More info:
http://wiki.ros.org/Master

Slide Credit: Marco Hutter, ETH Zurich

ROS Nodes

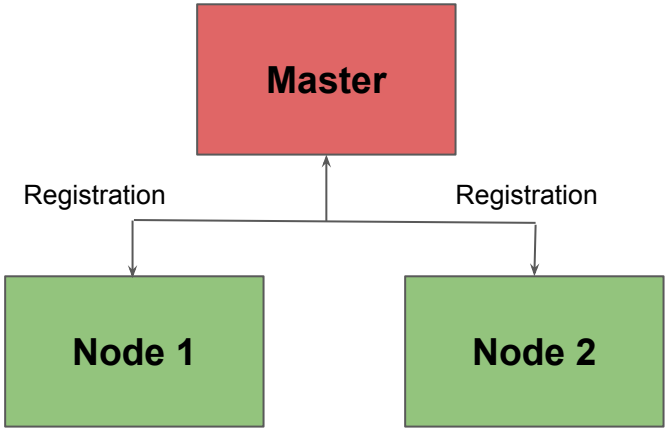
- Single-purpose, executable program
- Individually compiled, executed, and managed
- Organized in packages

Run a node with

```
$ rosrun package_name node_name
```

See active nodes with

```
$ rosnode list
```

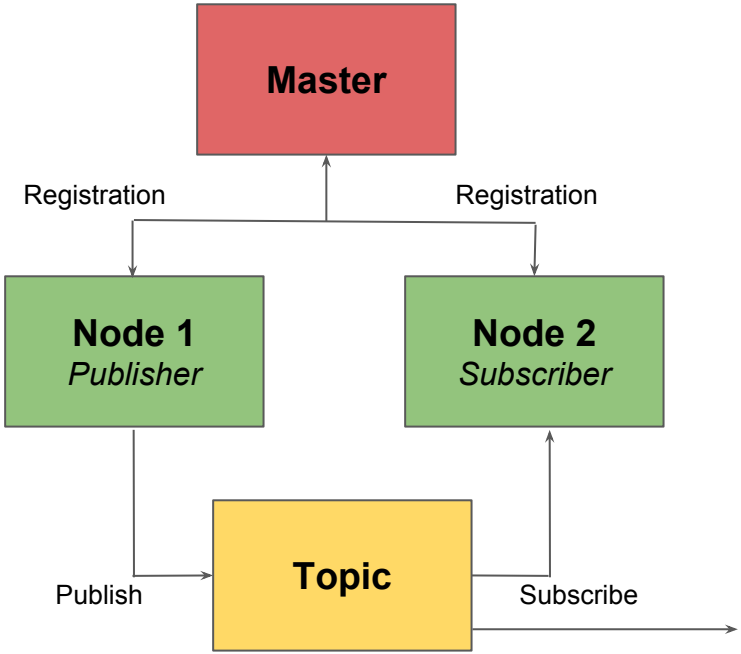


More info:
http://wiki.ros.org/rosnode

Slide Credit: Marco Hutter, ETH Zurich

ROS Topics

- Nodes communicate over topics
  - Nodes can publish or subscribe to a topic
  - Typically, 1 publisher and n subscribers
- Topic is name for stream of messages



See active topics with

```
$ rostopic list
```

Subscribe and print the contents of a topic with

```
$ rostopic echo /topic
```

More info:
http://wiki.ros.org/rostopic

Slide Credit: Marco Hutter, ETH Zurich

ROS Messages

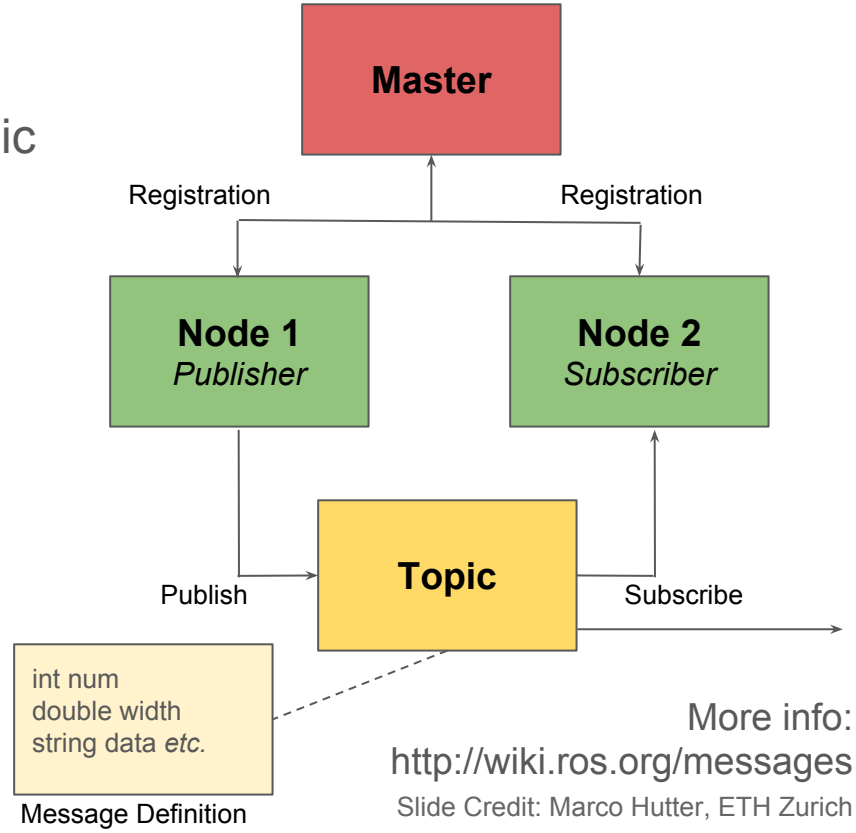
- Data structure defining the type of a topic
  - Comprised of a nested structure of integers, floats, strings etc. and arrays of objects
- Defined in *.msg files

See the type of a topic

```
$ rostopic type /topic
```

Publish a message to a topic

```
$ rostopic pub /topic type args
```



More info:
http://wiki.ros.org/messages

Slide Credit: Marco Hutter, ETH Zurich

ROS Messages

geometry_msgs/Point.msg

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

sensor_msgs/Image.msg

```
std_msgs/Header header
  uint32 seq
  time stamp
  string frame_id
uint32 height
uint32 width
string encoding
uint8 is_bigendian
uint32 step
uint8[] data
```

geometry_msgs/PoseStamped.msg

```
std_msgs/Header header
 uint32 seq
 time stamp
 string frame_id
geometry_msgs/Pose pose
  geometry_msgs/Point position
     float64 x
     float64 y
     float64 z
   geometry_msgs/Quaternion
orientation
     float64 x
     float64 y
     float64 z
     float64 w
```

More info:
http://wiki.ros.org/std_msgs

Slide Credit: Marco Hutter, ETH Zurich

End of Live Demo

- **What does ‘asynchronous’ communication mean?**
  - In our demo if the images are published @30fps
  - What if the image_view node takes more than 1/30th of a second to render and display the image on a slow computer?
  - ROS needs a general mechanism to transfer data between nodes working at different rates

Asynchronous Communication: Implementation

- Solution 1 - Queueing
  - If the subscribing node is continuously slow, then the queue might grow very large over time
  - Image display will lag far behind real time
- Solution 2 - Instantly dropping missed messages (processing only the latest)
  - If the subscribing node slows down only temporarily, then dropping messages can cause problems that could be avoided with queueing
  - For instance, in case of incremental/sequential data, missing one message in between can cause problems in processing further data

Asynchronous Communication: Implementation

- Better Solution - Middleground
  - ROS maintains a fixed length circular queue for messages
    - Messages stored until queue is full, then oldest one is dropped
  - Length of queue chosen by developer
    - Larger queue - less likely to drop messages
    - Shorter queue - less likely to show outdated/old messages
  - If dropping messages in between is not an issue, choosing a queue size of 1/2/3 is perfectly fine

More info:
http://wiki.ros.org/rospy/Overview/Publishers%20and%20Subscribers#Choosing_a_good_queue_size

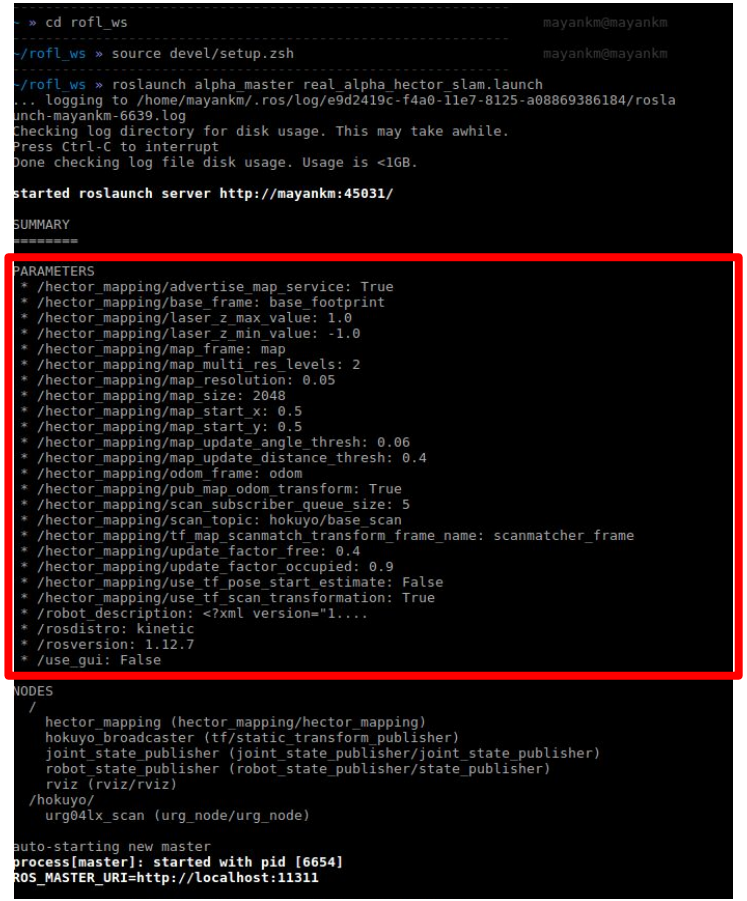
ROS Parameter Server

- Nodes use the parameter server to store and retrieve parameters at runtime
- Best used for static data such as configuration parameters
- Parameters can be defined in launch files or separate YAML files

List all parameters with

```
$ rosparam list
```

More info:
http://wiki.ros.org/rosparam



ROS Launch

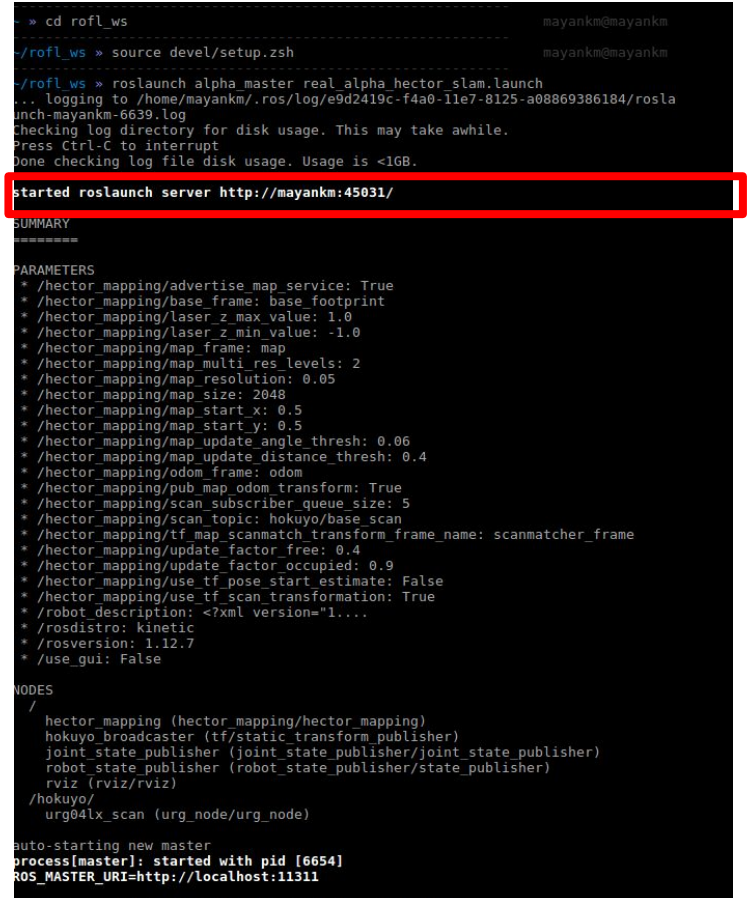
- launch is a tool for launching multiple nodes (as well as setting parameters)
- Are written in XML as *.launch files
- If not yet running, launch automatically starts a roscore

Start a launch file from a package with

```
$ roslaunch package_name file_name.launch
```

More info:
http://wiki.ros.org/roslaunch

Slide Credit: Marco Hutter, ETH Zurich

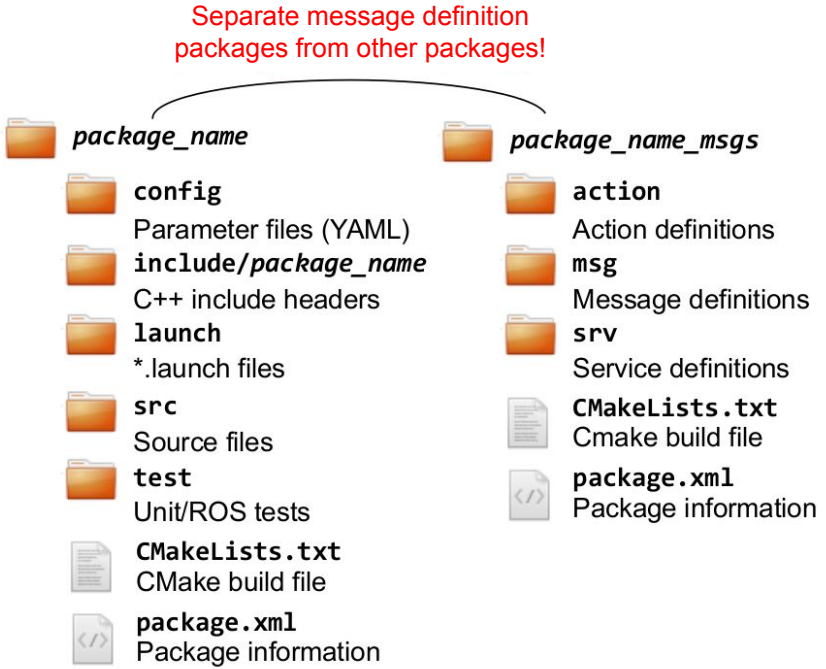


ROS Packages

- ROS software is organized into packages, which can contain source code, launch files,configuration files, message definitions, data, and documentation
- A package that builds up on/requires other packages (e.g. message definitions), declares these as dependencies

To create a new package, use:

```
$ catkin_create_pkg package_name {dependencies}
```



More info:
http://wiki.ros.org/Packages

Slide Credit: Marco Hutter, ETH Zurich

catkin Build System

- *catkin* is the ROS build system to generate executables, libraries, and interfaces
- The *catkin* command line tools are pre-installed in the provided installation.

Navigate to your catkin workspace with

```
$ cd ~/catkin_ws
```

Build a package with

```
$ catkin_make --package package_name
```

Whenever you build a new package, update your environment

```
$ source devel/setup.bash
```

Slide Credit: Lorenz Mösenlechner, TU Munich

catkin Build System

The catkin workspace contains the following spaces

Work here

The source space contains the source code. This is where you can clone, create, and edit source code for the packages you want to build.

Don't touch

The build space is where CMake is invoked to build the packages in the source space. Cache information and other intermediate files are kept here.

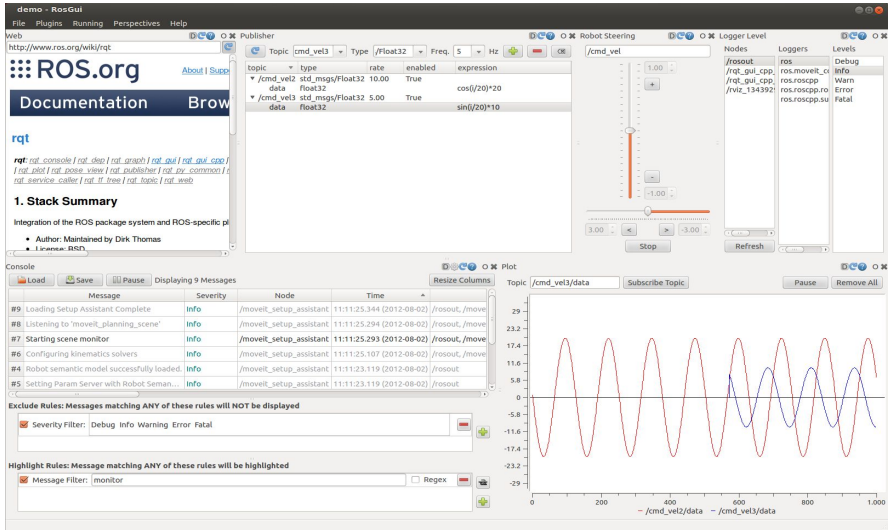
Don't touch

The development (devel) space is where built targets are placed (prior to being installed).

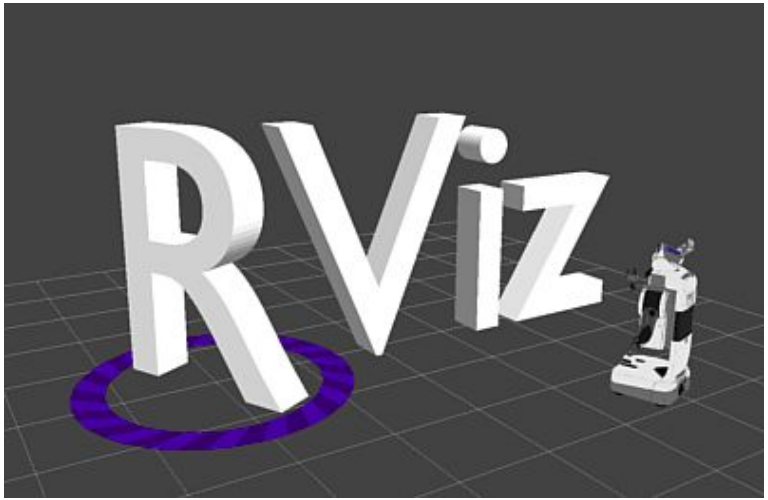
Slide Credit: Marco Hutter, ETH Zurich

ROS GUI Tools

rqt : A QT based GUI developed for ROS

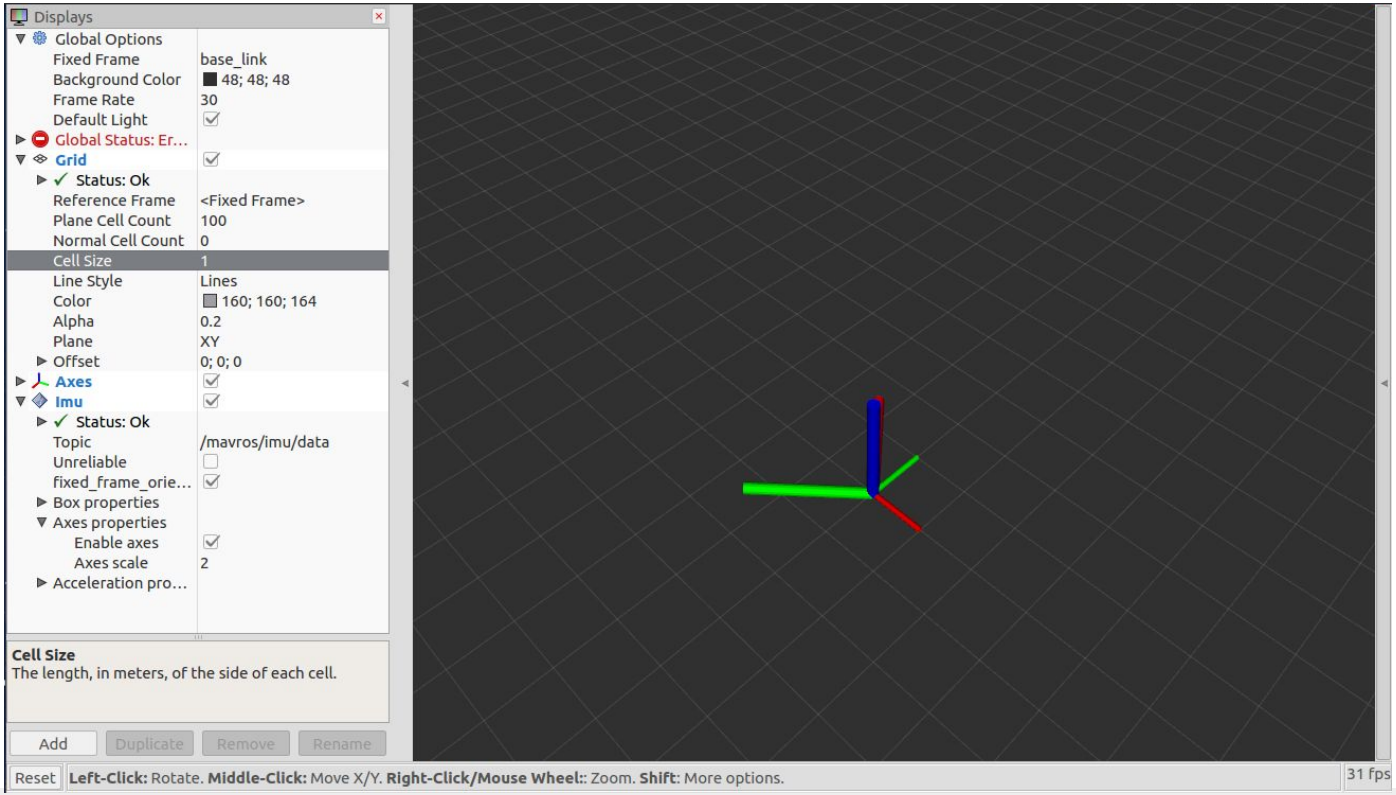


rviz : Powerful tool for 3D Visualization

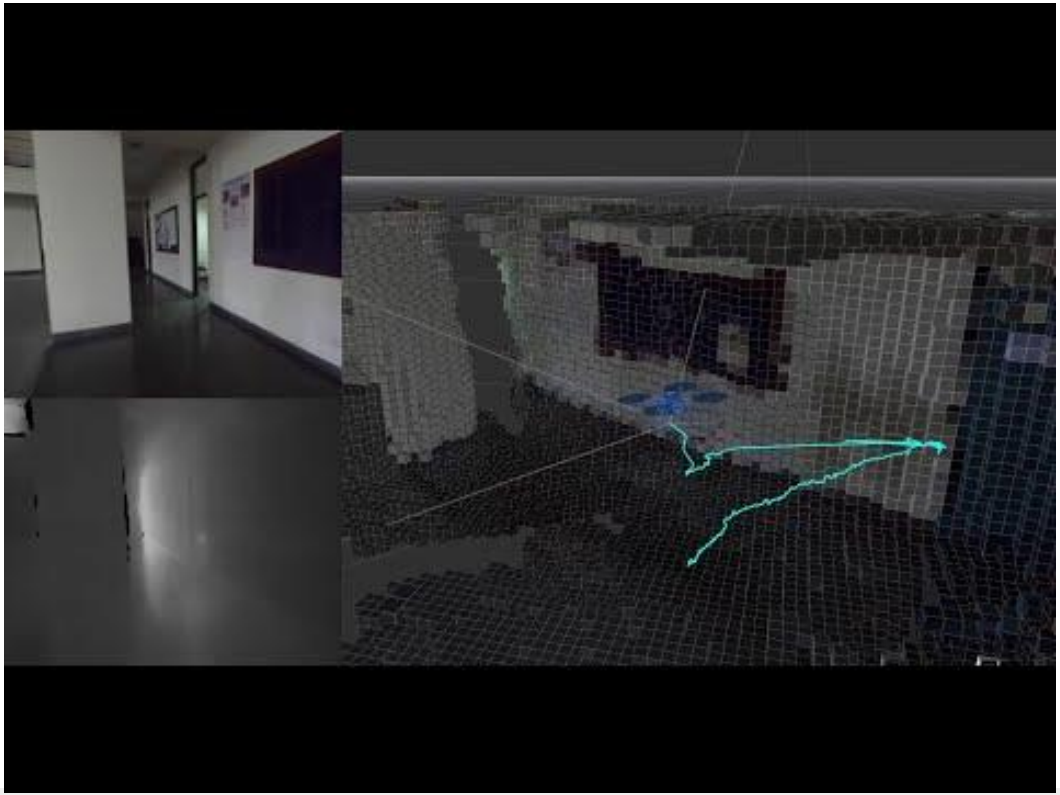


More info:
http://wiki.ros.org/rqt

Live Demo 2: IMU data on RViz



Video: RViz Capabilities



Video Link

ROS Bags

- A bag is a format for storing message data
- Binary format with file extension *.bag
- Suited for logging and recording datasets for later visualization and analysis

Record all topics in a bag

```
$ rosbag record --all
```

Record given topics

```
$ rosbag record topic_1 topic_2 topic_3
```

Show information about a bag

```
$ rosbag info bag_name.bag
```

Record given topics

```
$ rosbag play [options] bag_name.bag
```

--rate=factor	Publish rate factor
--clock	Publish the clock time (set param use_sim_time to true)
--loop	Loop playback

More info:
http://wiki.ros.org/Clock

Slide Credit: Marco Hutter, ETH Zurich

Live Demo 3: Recording Image Data (and Playing it Back)

Libraries/Tools available with ROS

Image Courtesy: Open Source Robotics Foundation

Homework

- Install ROS Kinetic on your laptop (Ubuntu 16.04LTS)
  - Instructions: http://wiki.ros.org/kinetic/Installation/Ubuntu
  - Alternate Option:
    - Download Shell Script (available here)
    - Run on terminal: `./install_ROS kinetic`
- Complete all of the ROS beginner tutorials. (1-8, 11, 13, 17 are crucial)

References

- Slides from lectures on ‘Programming for Robotics’ by ETH Zurich
- A Gentle Introduction to ROS, Jason M. O'Kane. Oct 2013 (available online)
- Berger, E., Conley, K., Faust, J., Foote, T., Gerkey, B.P., Leibs, J., Ng, A.Y., Quigley, M., & Wheeler, R. (2009). “**ROS: an open-source Robot Operating System”.**
- ROS Wiki