

ROS Workshop

Lorenz Mösenlechner
Technische Universität München

July 18th, 2012



Outline

1. Overview
2. ROS Communication Layer
3. ROS Build System
4. Programming with ROS
5. The TF Library

Outline

1. Overview
2. ROS Communication Layer
3. ROS Build System
4. Programming with ROS
5. The TF Library

What is ROS?

More than just a middleware



- ▶ A “meta” operating system for robots
- ▶ A collection of packaging, software building tools
- ▶ An architecture for distributed inter-process/inter-machine communication and configuration
- ▶ Development tools for system runtime and data analysis
- ▶ A language-independent architecture (c++, python, lisp, java, *and more*)

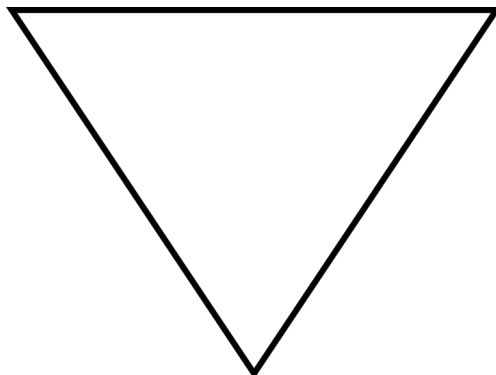
What is ROS not?

No confusion

- ▶ An *actual* operating system
- ▶ A programming language
- ▶ A programming environment / IDE
- ▶ A hard real-time architecture

What does ROS get you?

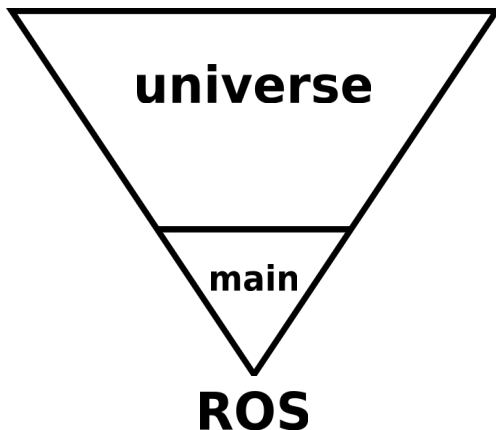
All levels of development



ROS

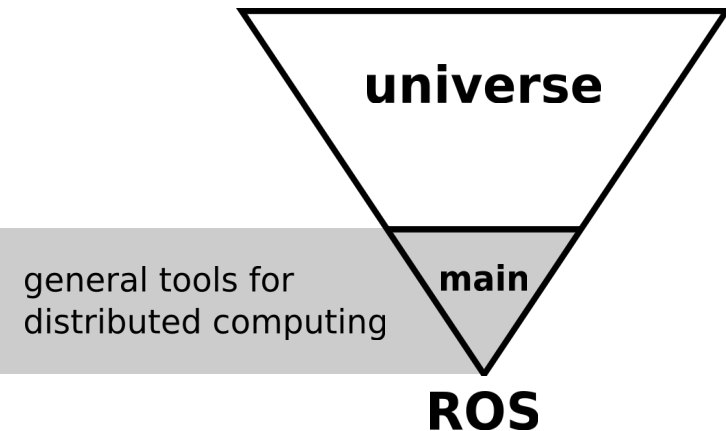
What does ROS get you?

All levels of development



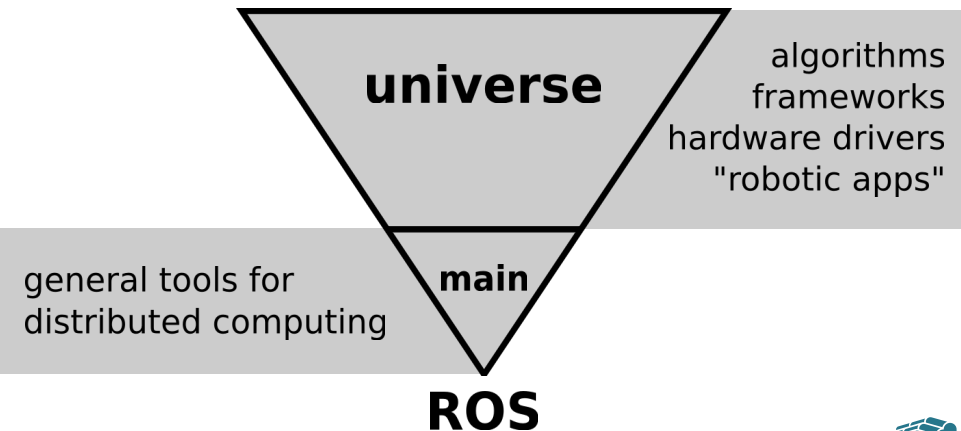
What does ROS get you?

All levels of development



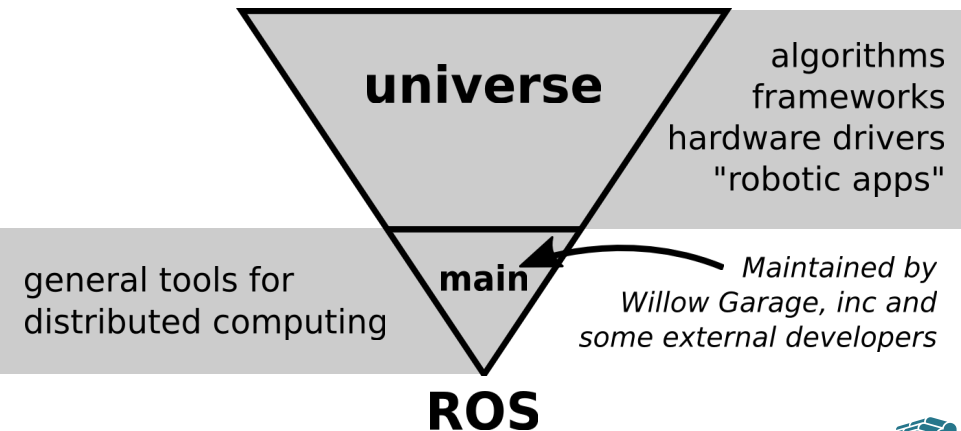
What does ROS get you?

All levels of development



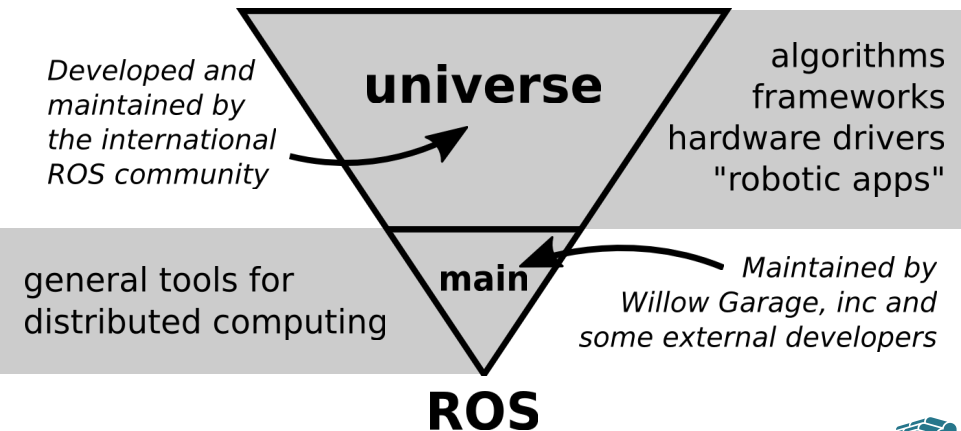
What does ROS get you?

All levels of development



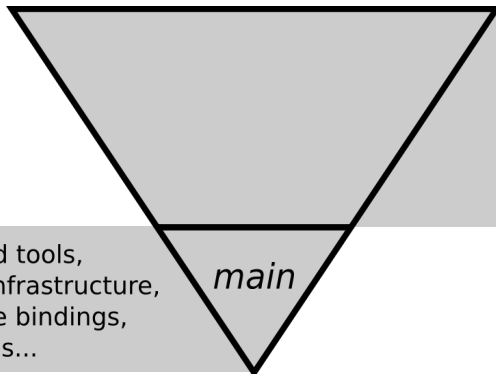
What does ROS get you?

All levels of development



What does ROS get you?

All levels of development

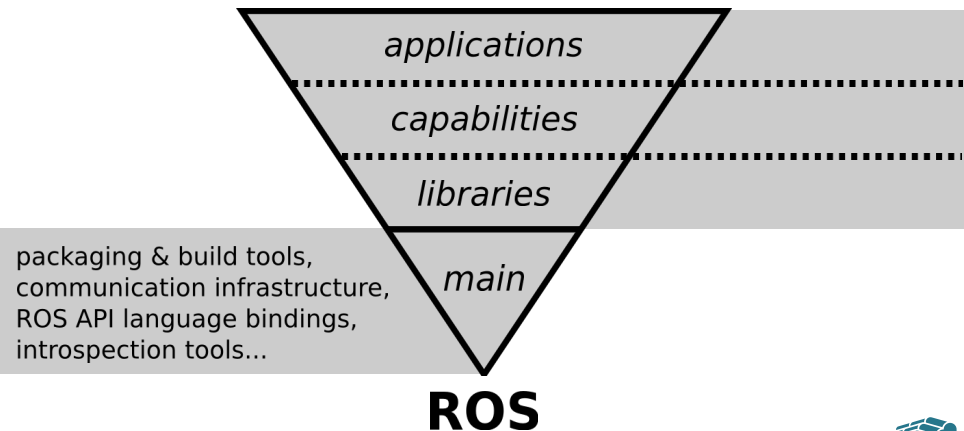


packaging & build tools,
communication infrastructure,
ROS API language bindings,
introspection tools...

ROS

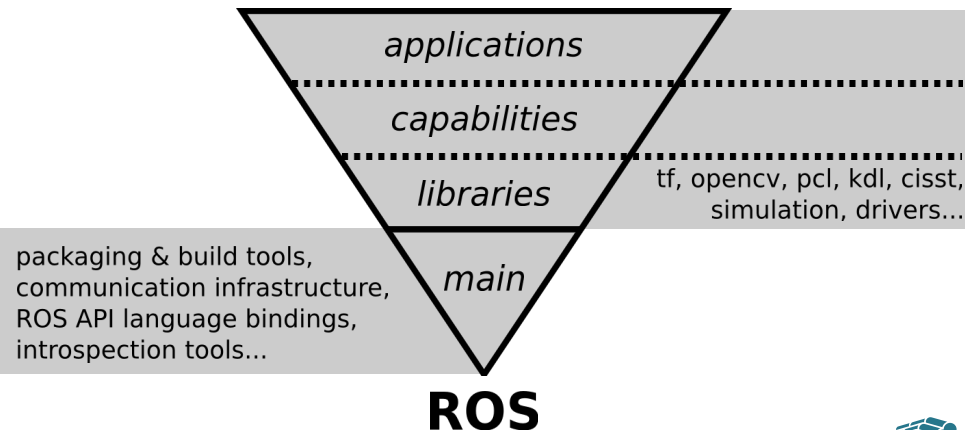
What does ROS get you?

All levels of development



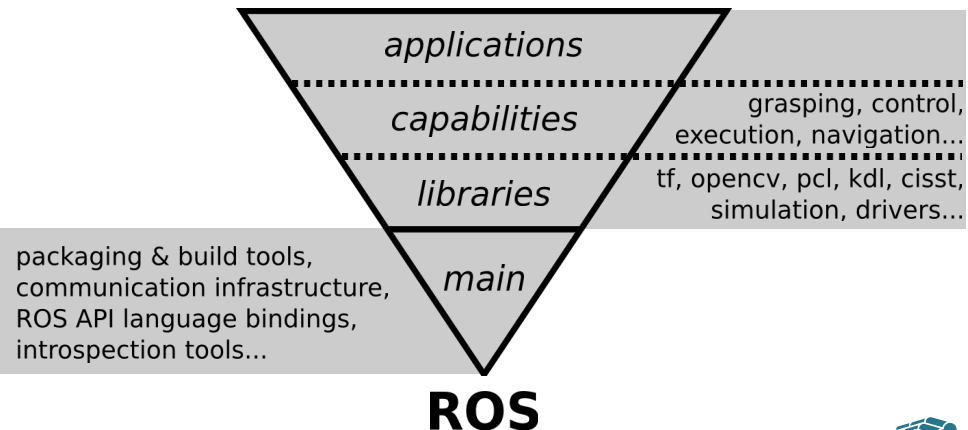
What does ROS get you?

All levels of development



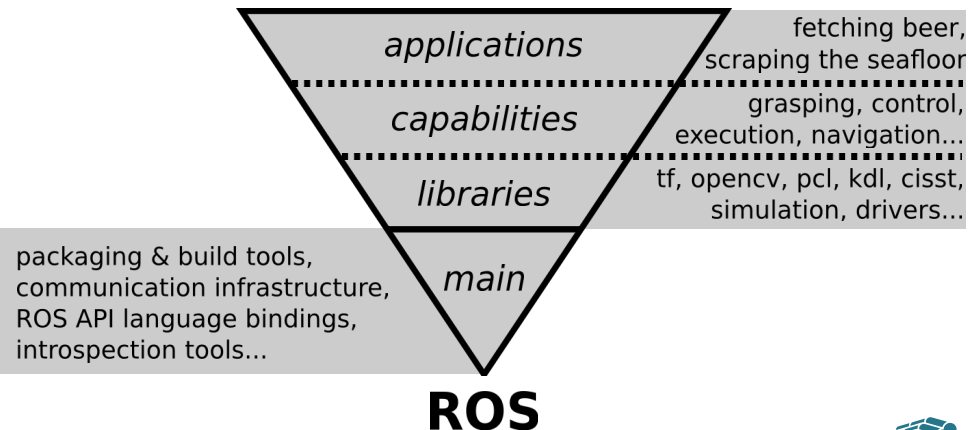
What does ROS get you?

All levels of development



What does ROS get you?

All levels of development



www.ros.org - The ROS Hub

A centralized location for ROS users and developers

[About](#) | [Support](#) | [answers.ros.org](#)Search: [Documentation](#)[Browse Software](#)[News](#)[Download](#)

Documentation

ROS (Robot Operating System) provides libraries and tools to help software developers create robot applications. It provides hardware abstraction, device drivers, libraries, visualizers, message-passing, package management, and more.

ROS:

[Install](#)

Install ROS on your machine.

[Getting Started](#)

[Tutorials](#), technical overview, and links to [getting help](#). Also, check out the [ROS Cheat Sheet PDF](#).

[Contribute](#)

How to contribute to the ROS community, such as submitting your own [repository](#).

Software:

[Core Libraries](#)

APIs by language and topic.

[Common Tools](#)

Common tools for developing and debugging ROS software.

◆ [Search Software](#)

Search the 2000+ libraries available for ROS.

Robot hardware:

[Robots](#)

Robots that you can use with ROS.

[Sensors](#)

Sensor drivers for ROS.

[Driver Tutorials](#)

Tutorials for supported hardware.

Publications, Courses, and Events:

[Papers](#)

Published papers with open source implementations available.

[Courses](#)

Courses using or teaching ROS.

[Events](#)

Past events and materials based on ROS.

Except where otherwise noted, the ROS wiki is licensed under [Creative Commons Attribution 3.0](#).

Wiki: Documentation (last edited 2011-06-17 23:49:49 by [MeloneeWise](#))

Wiki

[ROS](#)[RoadList](#)[RecentChanges](#)[rosaceReviews](#)[Documentation](#)

Page

[Edit \(Text\)](#)[Info](#)[Subscribe](#)[Add Link](#)[Attachments](#)[More Actions:](#) ▼

User

[JonathanBrennen](#)[Settings](#)[Logout](#)

answers.ros.org - ROS Questions & Answers

Community-supported help for ROS users

The screenshot shows the ROS Answers website interface. At the top, there are navigation links: **ANSWERS**, **questions**, **tags**, **people**, **badges**, and **ask a question**. A search bar is located on the right. Below the navigation, there's a section for **1,370 questions** with a search tip. The main content area lists several questions with their titles, tags, and status (e.g., 'Using a python node in parallel mode', 'Using the "Player" package', 'How much horsepower is needed to run the kinect stack'). To the right of the questions is a 'Contributors' section displaying a grid of avatars of users who have contributed answers. At the bottom right, there are sections for 'Interesting tags' and 'Ignored tags'.

Outline

1. Overview
2. ROS Communication Layer
3. ROS Build System
4. Programming with ROS
5. The TF Library

ROS Core

Where it all comes together

- ▶ **ROS Master**

- ▶ A centralized XML-RPC server
- ▶ Negotiates communication connections
- ▶ Registers and looks up names for ROS graph resources

- ▶ **Parameter Server**

Stores persistent configuration parameters and other arbitrary data

- ▶ **rosout**

Essentially a network-based stdout for human-readable messages

ROS “Graph” Abstraction

Named network resources

ROS graph resources:

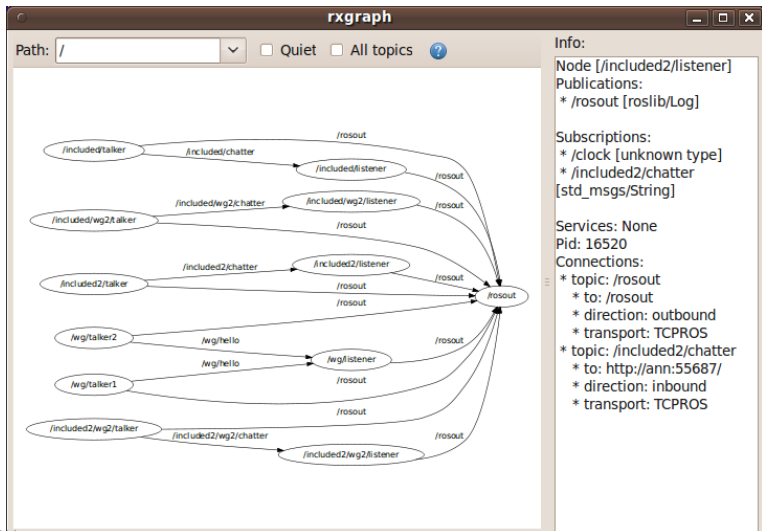
- ▶ **nodes**
 - ▶ processes
 - ▶ produce and consume data
- ▶ **parameters**
 - ▶ persistent data storage
 - ▶ configuration, initialization settings
 - ▶ stored on parameter server
- ▶ **topics**

Asynchronous many-to-many communication streams.
- ▶ **services**

Synchronous one-to-many network-based functions.

ROS “Graph”

rxgraph: communication network visualization



Creating and Running ROS Nodes

Distributing computation with ROS

Launch files

- ▶ XML files for launching nodes
- ▶ associate a set of parameters and nodes with a single file
- ▶ hierarchically compose collections of other launch files
- ▶ automatically re-spawn nodes if they crash
- ▶ change node names, namespaces, topics, and other resource names *without* recompiling
- ▶ easily distribute nodes across multiple machines

ROS Communication Protocols

Connecting nodes over the network

► ROS Topics

- Asynchronous “stream-like” communication
- Strongly-typed (ROS .msg spec)
- Can have one or more [publishers](#)
- Can have one or more [subscribers](#)

► ROS Services

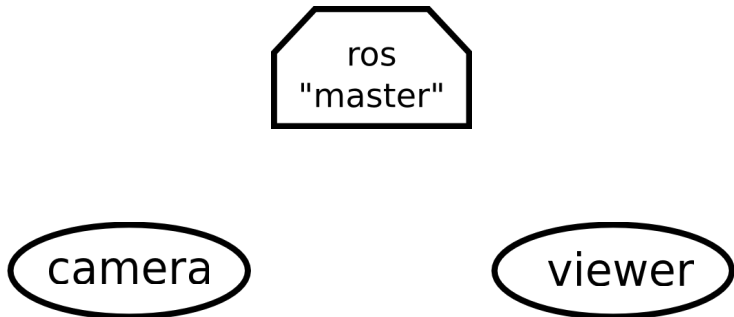
- Synchronous “function-call-like” communication
- Strongly-typed (ROS .srv spec)
- Can have only one [server](#)
- Can have one or more [clients](#)

► Actions

- Built on top of topics
- Long running processes
- Cancellation

Asynchronous Distributed Communication

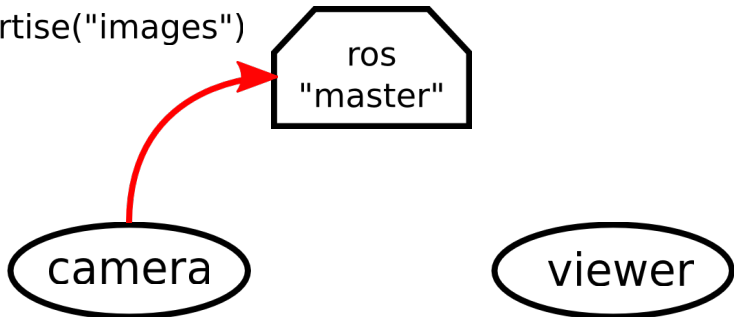
ROS TCP Topics



Asynchronous Distributed Communication

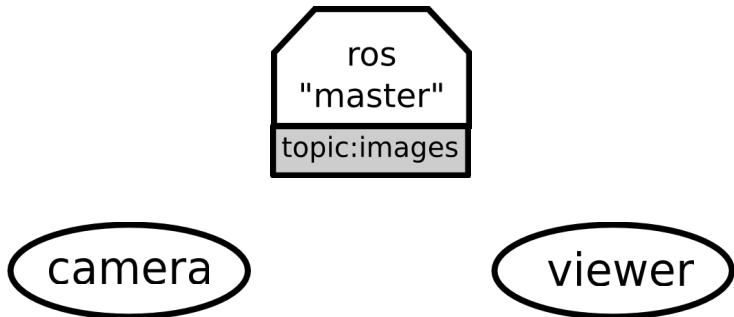
ROS TCP Topics

`advertise("images")`



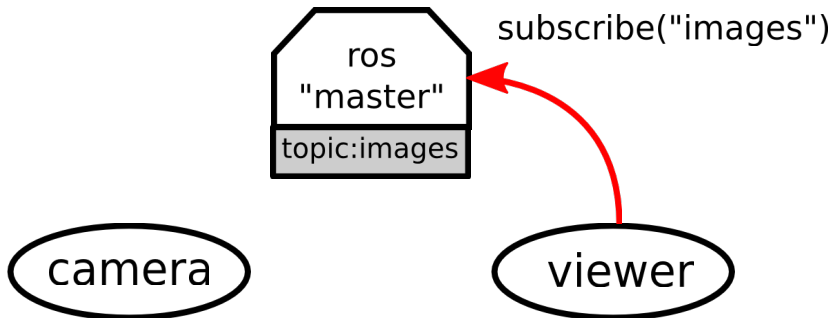
Asynchronous Distributed Communication

ROS TCP Topics



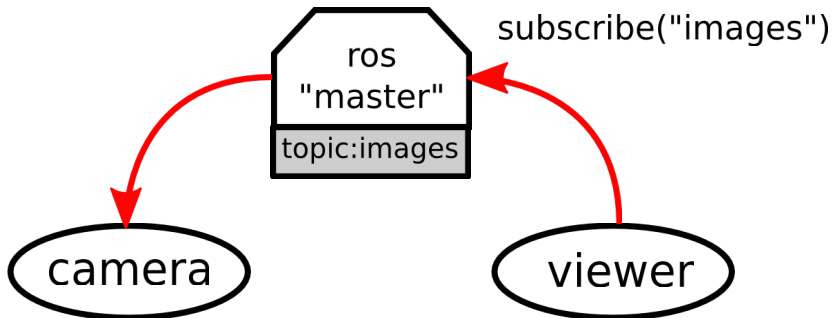
Asynchronous Distributed Communication

ROS TCP Topics



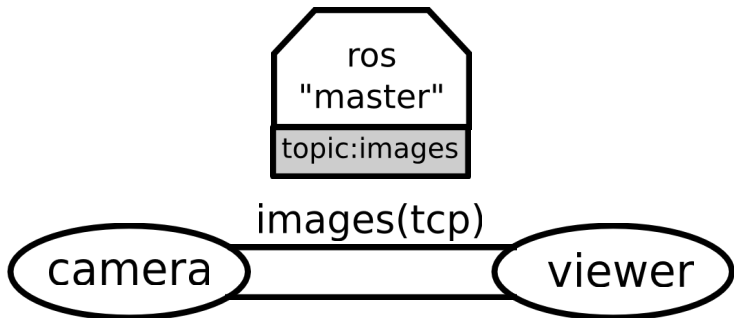
Asynchronous Distributed Communication

ROS TCP Topics



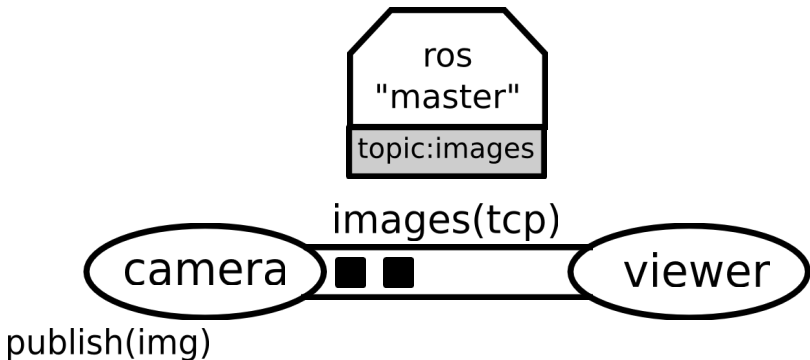
Asynchronous Distributed Communication

ROS TCP Topics



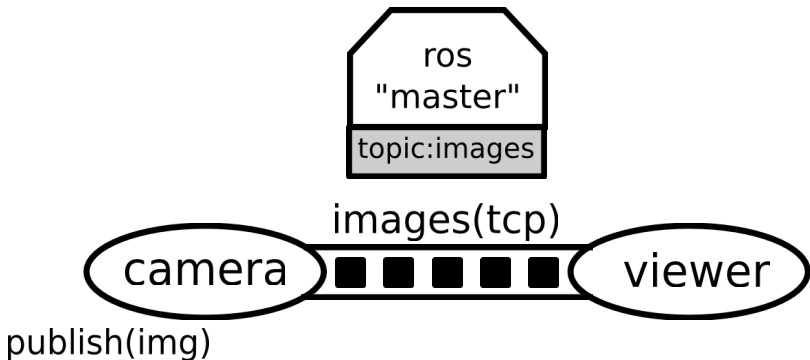
Asynchronous Distributed Communication

ROS TCP Topics



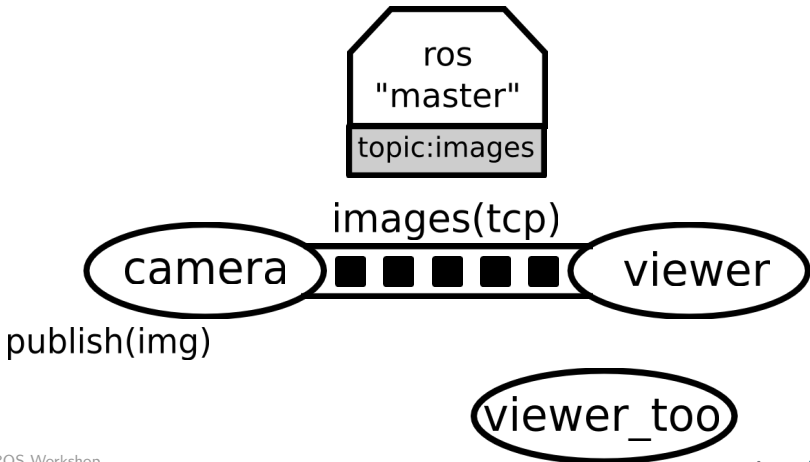
Asynchronous Distributed Communication

ROS TCP Topics



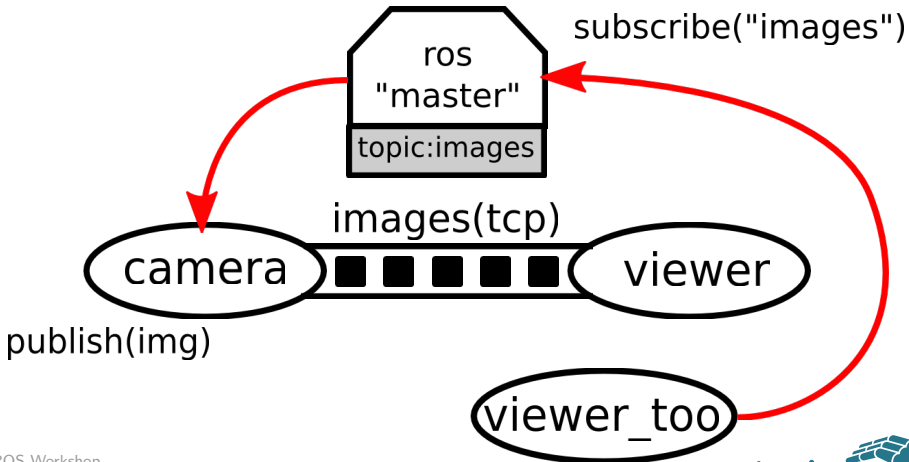
Asynchronous Distributed Communication

ROS TCP Topics



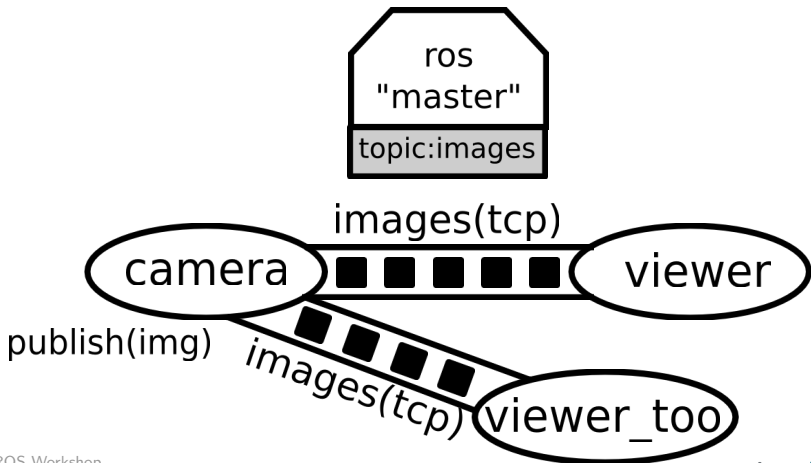
Asynchronous Distributed Communication

ROS TCP Topics



Asynchronous Distributed Communication

ROS TCP Topics



Debugging

rosout

ROS provides mechanisms in all languages for specifying different *levels* of human-readable log messages.

The five default levels are:

- ▶ fatal
- ▶ error
- ▶ warn
- ▶ info
- ▶ debug

Corresponding logging commands (C++):

- ▶ `ROS_FATAL(...)`
- ▶ `ROS_ERROR(...)`
- ▶ `ROS_WARN(...)`
- ▶ `ROS_INFO(...)`
- ▶ `ROS_DEBUG(...)`

ROS Graph Introspection

No more wireshark

ROS provides several tools for analyzing the data flowing over ROS communication resources:

- ▶ `rostopic`

Gives a user information about a node: publications, subscriptions, etc

- ▶ `rostopic`

Gives datarate, actual data, publishers, subscribes

- ▶ `rosservice`

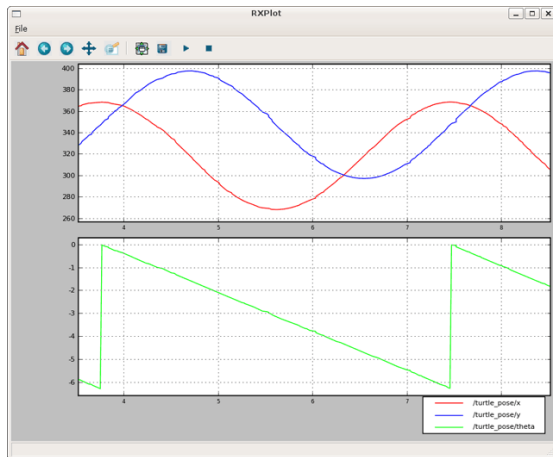
Enables a user to call a ROS Service from the command line

- ▶ `roswtf` (wire trouble finder)

Diagnoses problems with a ROS network

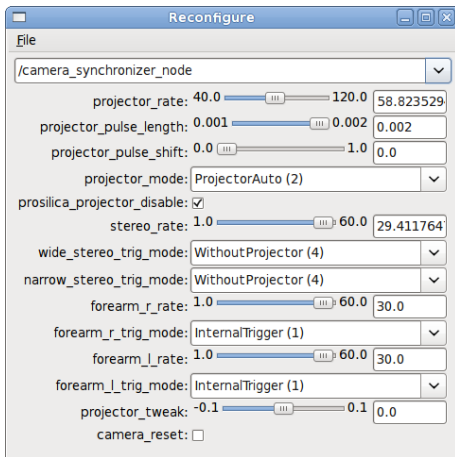
ROS GUI Tools

There are lots...



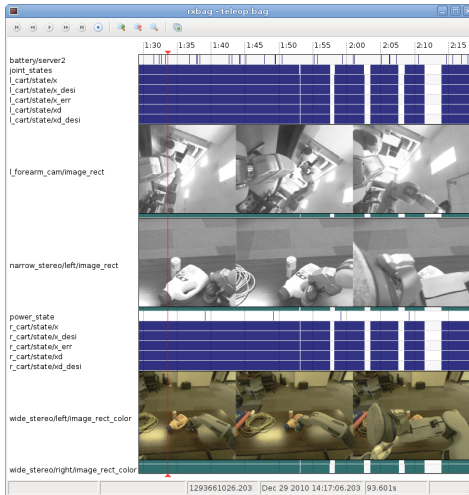
ROS GUI Tools

There are lots...



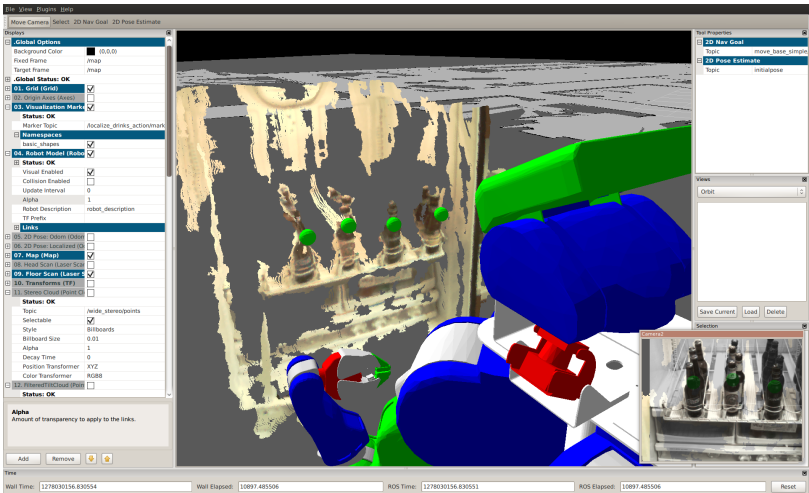
ROS GUI Tools

There are lots...



rviz - 3D Visualization

Modular state and sensor visualization



ROS Distributions

Delivering ROS packages to the masses

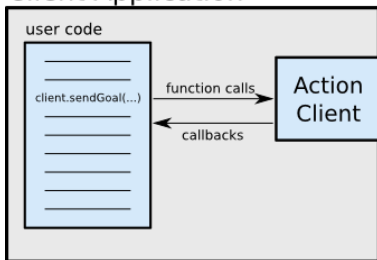
- source code
- header declarations
- scripts
- message definitions
- service definitions
- configuration files
- launch files
- metadata
- ...

Actions

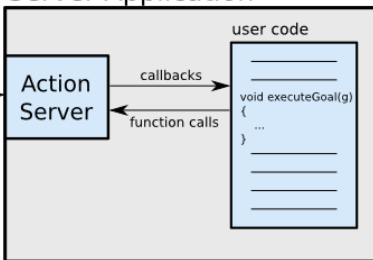
Using function calls and callbacks

- ▶ request goals (client side)
- ▶ execute goals (server side)

Client Application



Server Application

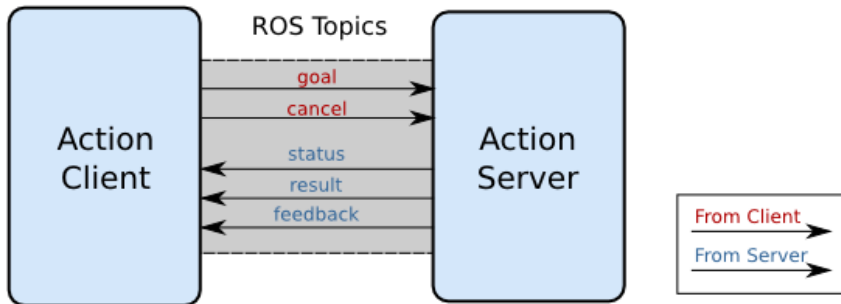


ROS

Actions

- ▶ **action protocol** relies on ROS topics to transport messages

Action Interface



Action Definitions

- ▶ Similar to messages and services.
- ▶ Definition: Request + result + feedback
- ▶ Defined in `ros-package/action/*.action`
- ▶ Generated by CMake macro `genaction()`.
- ▶ Example: `actionlib_tutorials/Fibonacci.action`

```
#goal definition
```

```
int32 order
```

```
---
```

```
#result definition
```

```
int32[] sequence
```

```
---
```

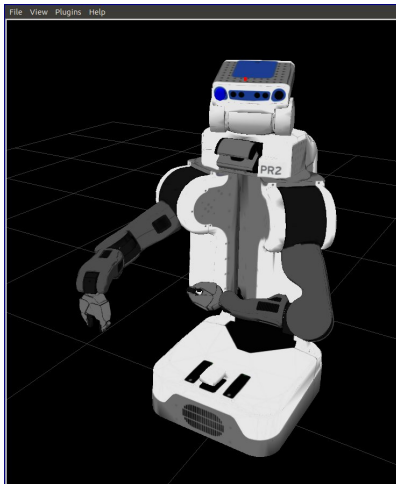
```
#feedback
```

```
int32[] sequence
```

Outline

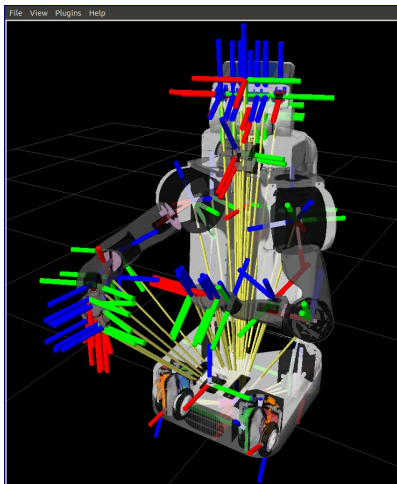
1. Overview
2. ROS Communication Layer
3. ROS Build System
4. Programming with ROS
5. The TF Library

Coordinate frames



- ▶ robots consist of many **links**
- ▶ every link describes its own **coordinate system**
- ▶ sensor measurements are local to the corresponding link
- ▶ links change their position over time

Coordinate frames



- ▶ robots consist of many **links**
- ▶ every link describes its own **coordinate system**
- ▶ sensor measurements are local to the corresponding link
- ▶ links change their position over time

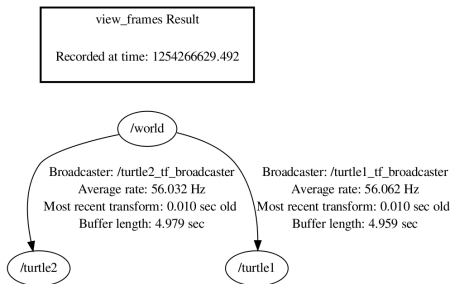
Transforms are distributed

- ▶ Transforms are produced by different nodes:
 - ▶ Localization in map (AMCL, gmapping)
 - ▶ Odometry (base controller)
 - ▶ Joint positions (robot controllers and robot_state_publisher)
- ▶ Many publishers, many consumers
- ▶ Distributed system, redundancy issues, ...

What is TF?

- ▶ decentralized: many publishers, many subscribers
- ▶ A coordinate frame tracking system
 - ▶ standardized protocol for publishing transforms
 - ▶ Classes and methods for looking up, calculating and sending transforms
 - ▶ transforms are published on the /tf topic
- ▶ No central instance managing the tree of transforms
- ▶ Command line tools

The transform tree



- ▶ Consists of frames (links) and the transforms between them.
- ▶ Each link is cached (10 secs default caching time)
- ▶ Works with multiple disconnected trees
- ▶ Transforms must form a proper tree (no cycles)

TF data types

- ▶ Transforms and poses
- ▶ **stamped** data types (via ROS header)
- ▶ Header contains time stamp and frame names
- ▶ **StampedTransform** and **PoseStamped**
- ▶ StampedTransform: frame name and child frame name

TF and time

- ▶ TF buffers transforms for 10 seconds
- ▶ query transforms in the past
- ▶ TF interpolates frames
- ▶ fixed frame: frame that doesn't move (reference)