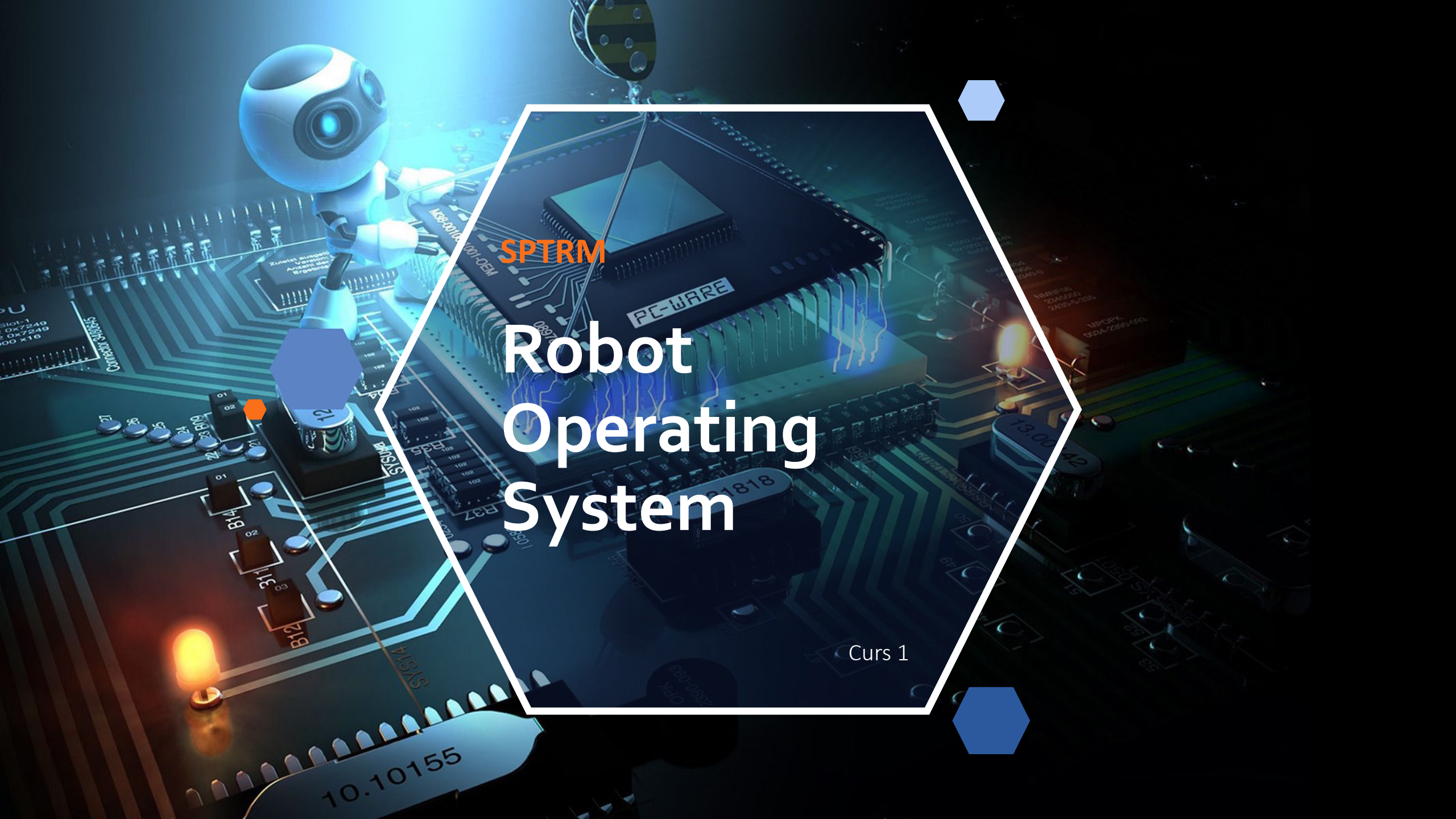
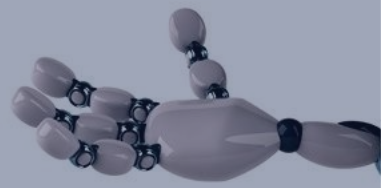


SPTRM

Robot Operating System

Curs 1

Introduction in ROS



Topics. Services. Actions.
Messages. Parameter Server

What is ROS

”ROS (Robot Operating System) is an open-source, meta-operating system for your robot.

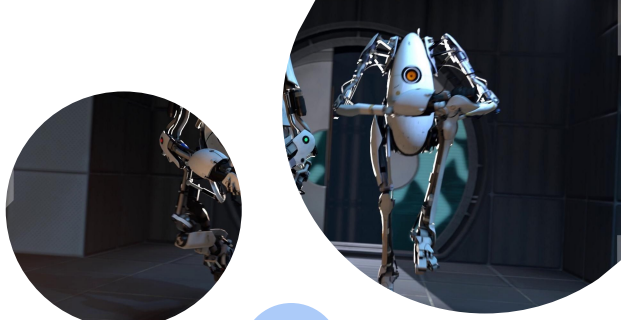
It provides the services you would expect from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management.

It also provides tools and libraries for obtaining, building, writing, and running code across multiple computers.” (official description)

What is ROS

- **Plumbing** (Process management, IPC, Device drivers)
- **Tools** (Simulation, Visualisation, GUI, Data logging)
- **Capabilities** (Control, Planning, Perception, Mapping, Manipulation)
- **Ecosystem** (Package management, Software distribution, Documentation, Tutorials) - http://wiki.ros.org/

ROS details



What brings ROS

- Distributed computation
- Software reuse
- Rapid testing

What is not ROS

- ROS is not a programming language
- ROS is not (only) a library
- ROS is not an integrated development environment

History

- 2007 - Stanford Artificial Intelligence Laboratory -> Willow Garage
- 2013 - maintained by OSRF (Open Source Robotic Foundation)
- Used by many robots (commercial / research)
- De facto standard for robot programming

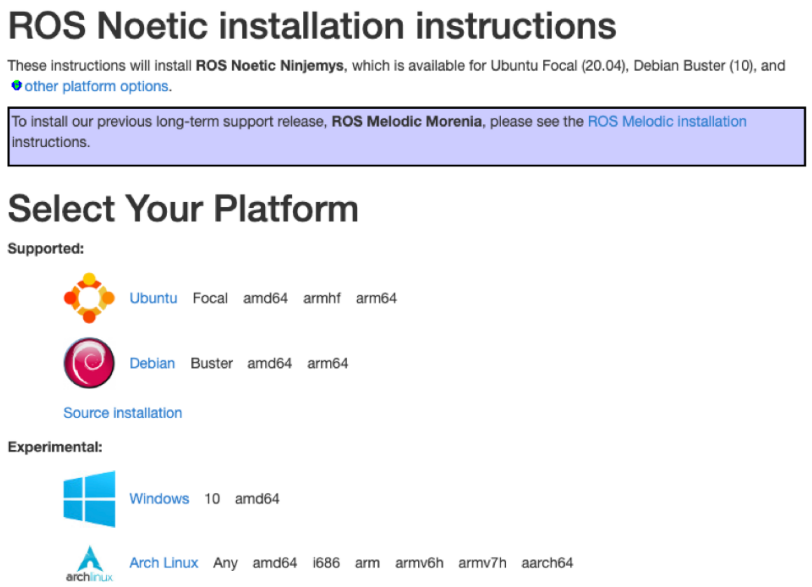
Versions

Version	Launch Date	EOL Date
Box Turtle	March 2, 2010	
C Turtle	August 2, 2010	
Diamondback	March 2, 2011	
Electric Emys	August 30, 2011	
Fuerte Turtle	April 23, 2012	
Groovy Galapagos	December 31, 2012	July, 2014
Hydro Medusa	September 4th, 2013	May, 2015
Indigo Igloo	July 22nd, 2014	April, 2019
Jade Turtle	May 23rd, 2015	May, 2017
Kinetic Kame	May 23rd, 2016	April, 2021
Lunar Loggerhead	May 23rd, 2017	May, 2019
Melodic Morenia	May, 2018	May, 2023
Noetic Ninjemys	May, 2020	May, 2025

Supported OS

- Supported OS: Ubuntu + Debian
- Experimental: OSX, Gentoo, OpenEmbedded

- http://wiki.ros.org/ROS/Installation



ROS Noetic installation instructions

These instructions will install **ROS Noetic Ninjemys**, which is available for Ubuntu Focal (20.04), Debian Buster (10), and other platform options.

To install our previous long-term support release, **ROS Melodic Morenia**, please see the ROS Melodic installation instructions.

Select Your Platform

Supported:

Ubuntu Focal amd64 armhf arm64

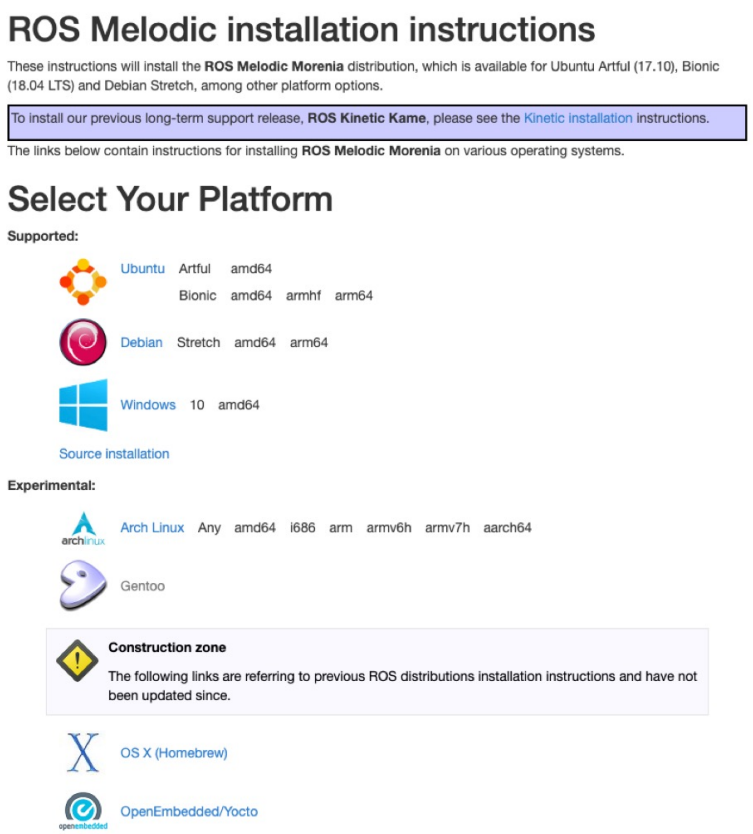
Debian Buster amd64 arm64

Source installation

Experimental:

Windows 10 amd64

Arch Linux Any amd64 i686 arm armv6h armv7h aarch64



ROS Melodic installation instructions

These instructions will install the **ROS Melodic Morenia** distribution, which is available for Ubuntu Artful (17.10), Bionic (18.04 LTS) and Debian Stretch, among other platform options.

To install our previous long-term support release, **ROS Kinetic Kame**, please see the Kinetic installation instructions.

The links below contain instructions for installing **ROS Melodic Morenia** on various operating systems.

Select Your Platform

Supported:

Ubuntu Artful amd64

Bionic amd64 armhf arm64

Debian Stretch amd64 arm64

Windows 10 amd64

Source installation

Experimental:

Arch Linux Any amd64 i686 arm armv6h armv7h aarch64

Gentoo

Construction zone

The following links are referring to previous ROS distributions installation instructions and have not been updated since.

OS X (Homebrew)

OpenEmbedded/Yocto

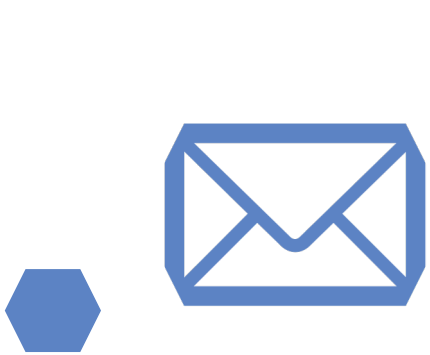
Philosophy

- **Peer to peer** (IPC over defined API: ROS messages, topics, services, etc.)
- **Distributed** (multiple computers connected in LAN)
- **Multi-langual** (client library: C++, Python, Matlab, Java, etc.)
- **Lightweight** (standalone libraries wrapped around a thin ROS layer)
- **Free and open-source**

ROS core components

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

ROS Core Components

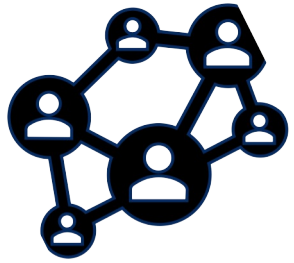


Communication

Message Passing

Bags

Recording and Playback of Messages



Network

Remote Procedure Calls

Integrated

Distributed Parameter System

Message Passing

- built-in and well-tested messaging system
- manages the details of communication between distributed nodes via the anonymous publish/subscribe mechanism
- forces the developer to implement clear interfaces between the nodes in your system
- the structure of these message interfaces is defined in the **message IDL** (Interface Description Language).

Message Passing

- Manages communication between nodes
- Every node registers to a master
- Start master:

roscore

ROS Master

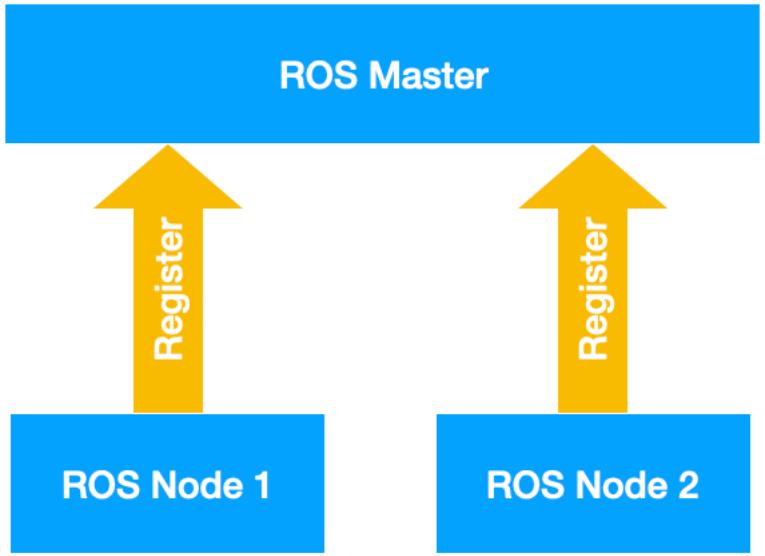
ROS Nodes

- Single-purpose, executable program
- Individually compiled, executed and managed
- Organised in packages
- Run a node:

  rosrun <package> <node>
- See active nodes:

  rosnode list
- Get info about a node:

  rosnode info <node>



ROS Topics

- Node (IPC) communication over topics
- Publisher vs Subscriber
- Topic = name for a stream of messages
- List topics:

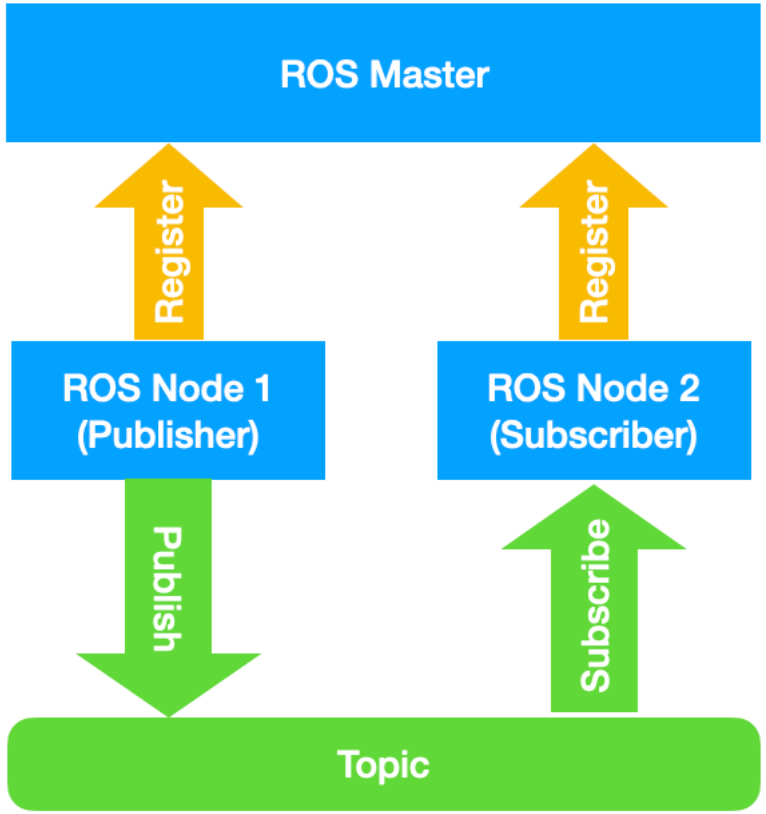
  rostopic list

- Listen to a topic:

  rostopic echo /topic

- Get info about a topic:

  rostopic info /topic



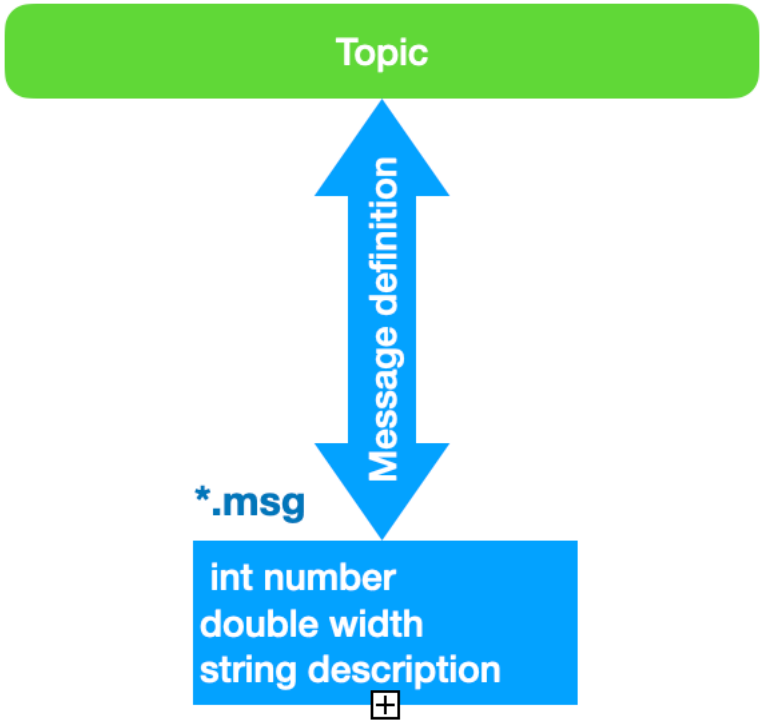
ROS Message

- Data structure defining the type of a topic
- Nested structure
- Simple data (integers, floats, strings, etc.)
- Complex types (objects)
- Supports Arrays
- Defined in a *.msg file

  rostopic type /topic
- Publish a message on a topic

  rostopic pub /topic type args
- Listen to a topic

  rostopic echo /topic



ROS Message Example

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
```

Recording and Playback of Messages

- the publish/subscribe system is anonymous and asynchronous
- data can be easily captured and replayed without any changes to code
- powerful design pattern that can significantly reduce your development effort and promote flexibility and modularity

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.

Preemptable Remote Procedure Calls

- **Actions** are like services except they can report progress before returning the final response, and they can be preempted by the caller.

- So, for example, you can instruct your robot to navigate to some location, monitor its progress as it attempts to get there, stop or redirect it along the way, and be told when it has succeeded (or failed).

Distributed Parameter System

- The ROS middleware also provides a way for tasks to share configuration information through a global key-value store.

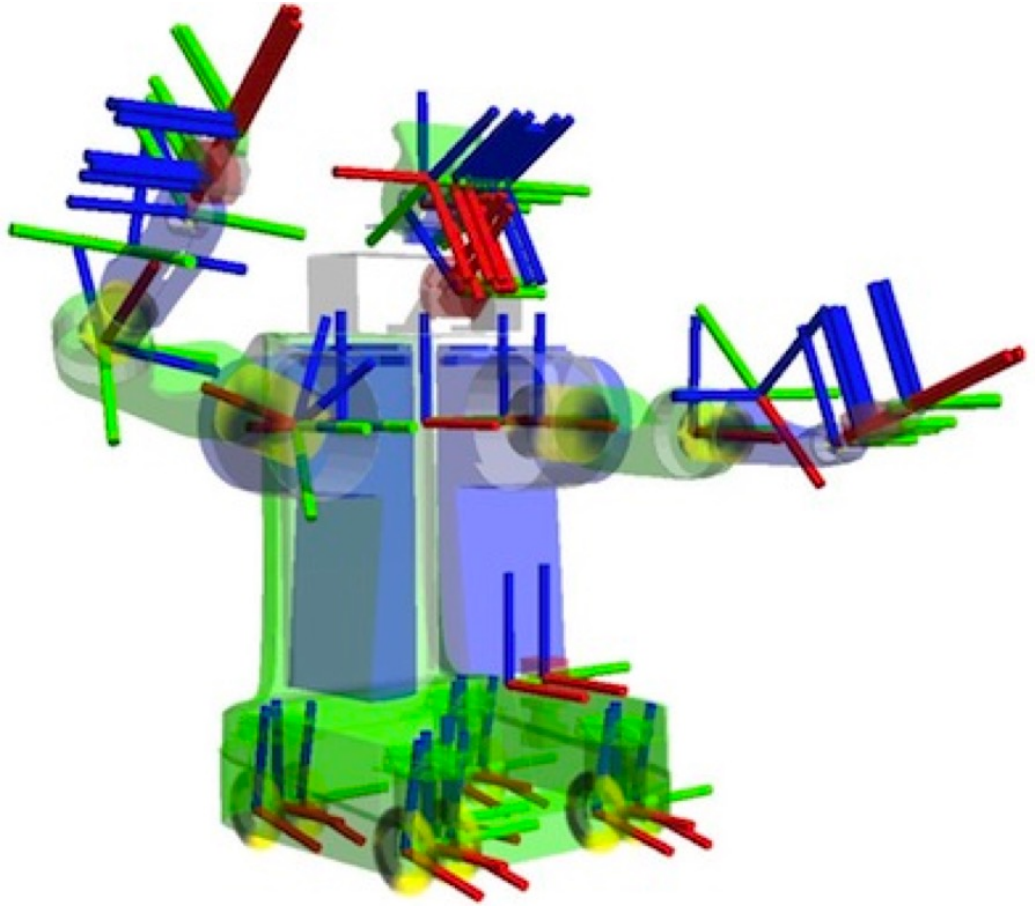
- This system allows you to easily modify your task settings, and even allows tasks to change the configuration of other tasks.

Robot-specific features

- Standard Robot Messages
  - for geometric concepts like poses, transforms, and vectors;
  - for sensors like cameras, IMUs and lasers;
  - for navigation data like odometry, paths, and maps; among many others

Robot-specific features

- Robot Geometry Library
  - keeps track of where different parts of the robot
  - tf (transform) library, which will keep track of where everything is in your robot system.
  - tf library can manage coordinate transform data for robots with more than one hundred degrees of freedom and update rates of hundreds of Hertz



Robot-specific features

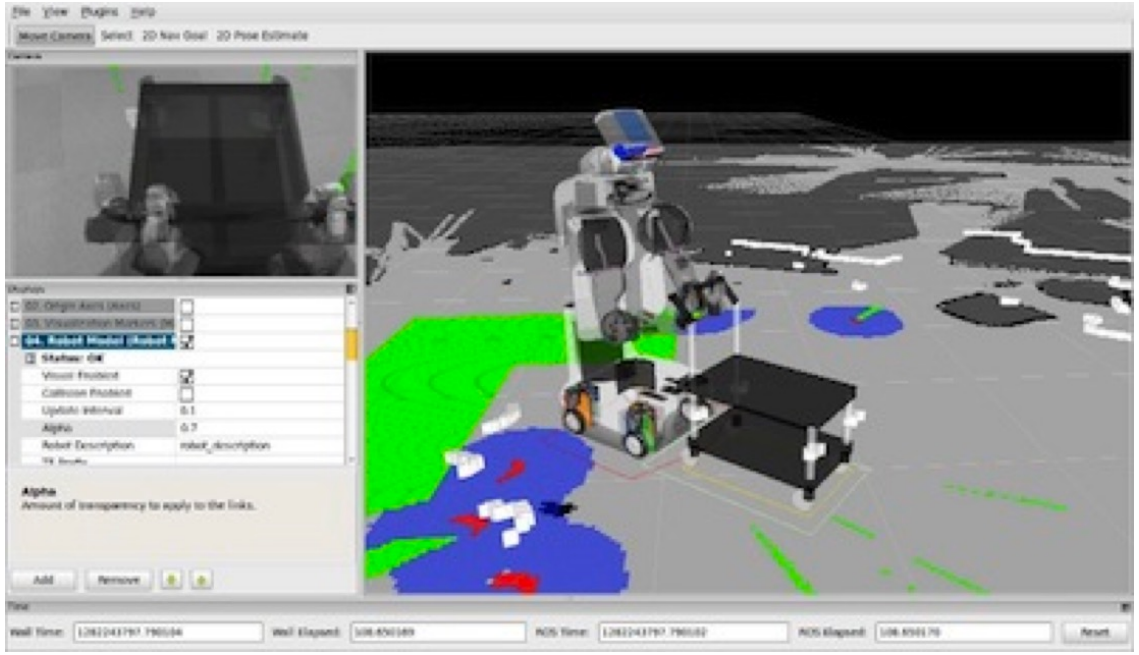
- Robot Description Language
  - how to describe your robot in a machine-readable way
  - URDF (Unified Robot Description Format)
  - integrated tools tf, robot_state_publisher, and rviz.

Tools

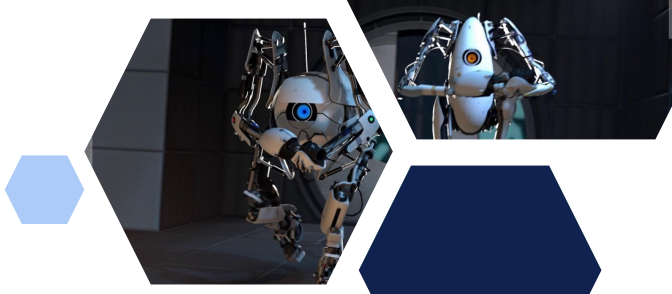
- more than 45 command line tools
  - launching multiple nodes
  - check topics, services and actions
  - recording and playing data
- graphical tools, rviz and rqt provide similar (and extended) functionality

rviz

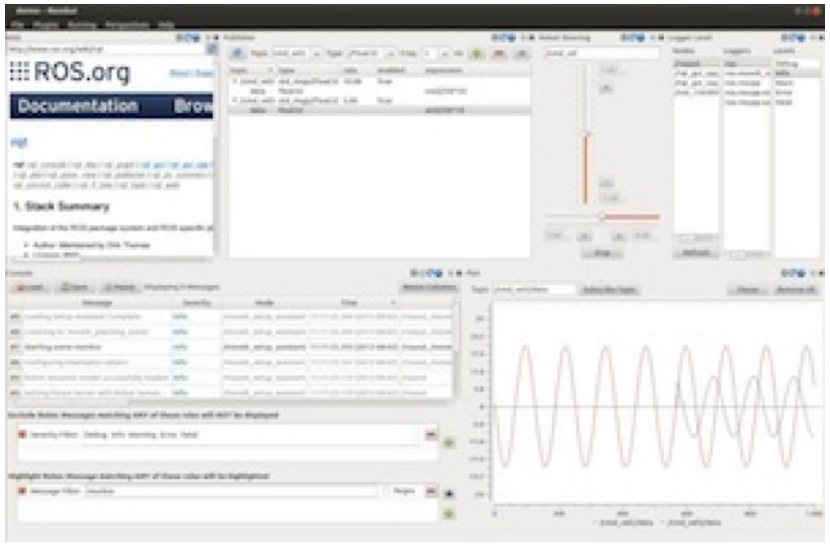
- most well-known tool in ROS
- provides general purpose, three-dimensional visualization of many sensor data types and any URDF-described robot.



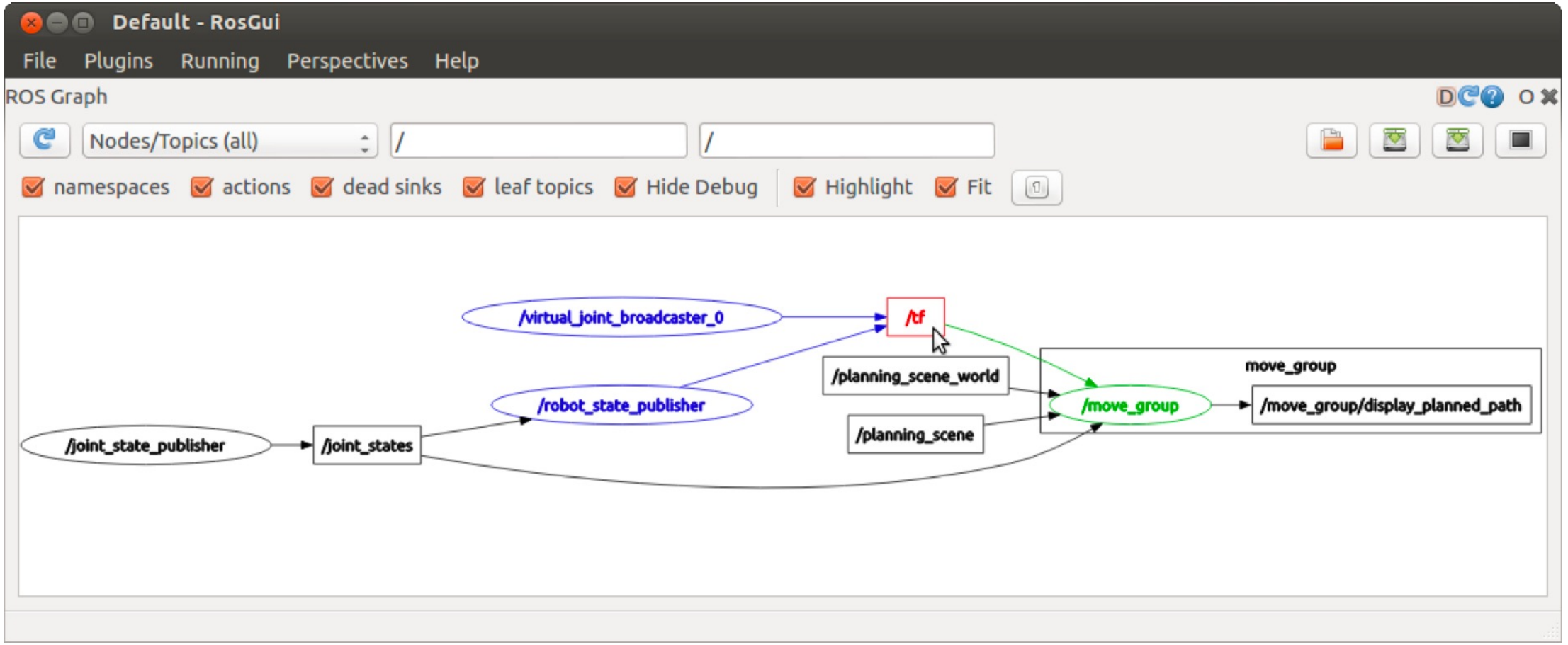
rqt



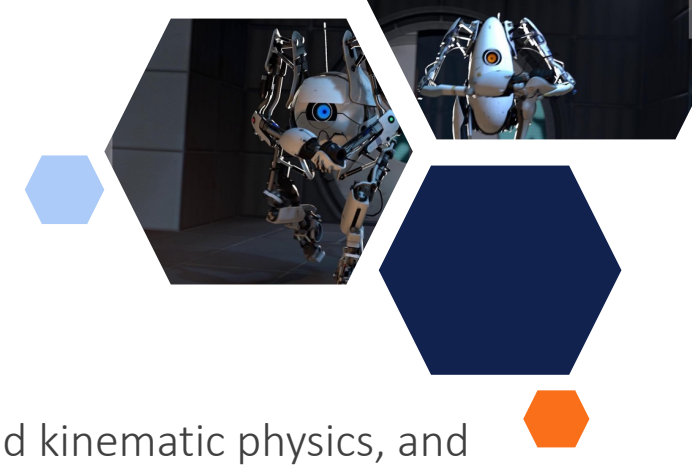
- a Qt-based framework for developing graphical interfaces
- create custom interfaces by composing and configuring the extensive library of built-in rqt plugins into tabbed, split-screen, and other layout



rqt_graph


Default - RosGui
File
Plugins
Running
Perspectives
Help
ROS Graph
Nodes/Topics (all)
/
/
namespaces
actions
dead sinks
leaf topics
Hide Debug
Highlight
Fit
/virtual_joint_broadcaster_0
/tf
/planning_scene_world
move_group
/robot_state_publisher
/move_group
/move_group/display_planned_path
/joint_state_publisher
/joint_states
/planning_scene

Integrations



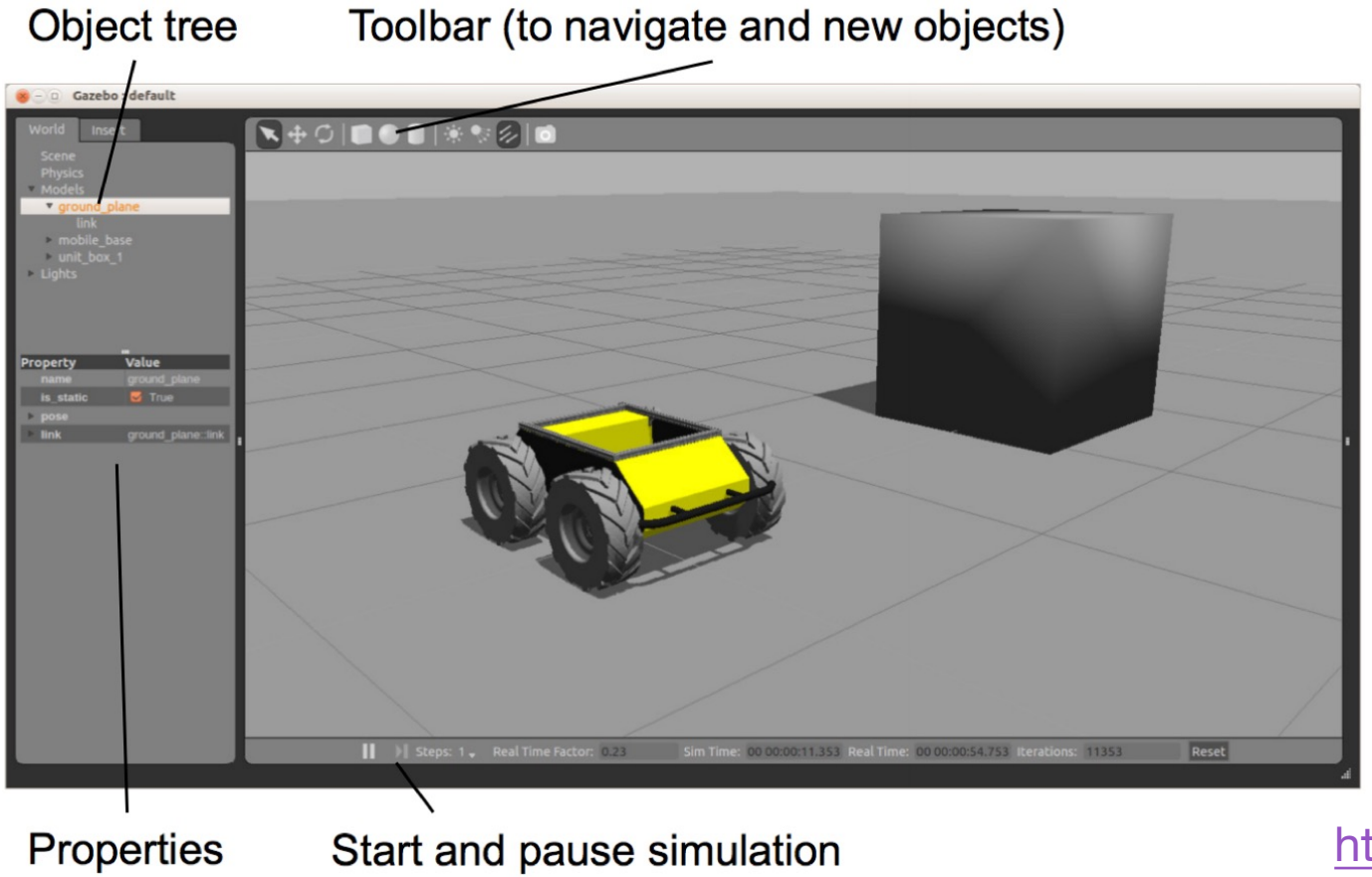
- **Gazebo** is a 3D indoor and outdoor multi-robot simulator, complete with dynamic and kinematic physics, and a pluggable physics engine.
- **OpenCV** is the premier computer vision library, used in academia and in products around the world. OpenCV provides many common computer vision algorithms and utilities that you can use and build upon
- **PCL**, the Point Cloud Library, is a perception library focused on the manipulation and processing of three-dimensional data and depth images
- **MoveIt** is a motion planning library that offers efficient, well-tested implementations of state of the art planning algorithms that have been used on a wide variety of robots, from simple wheeled platforms to walking humanoid
- **ROS-Industrial** is an open-source project that extends the advanced capabilities of ROS to manufacturing automation and robotics

Gazebo Simulator

- Simulate 3D rigid-body dynamics
- Simulate a variety of sensors (including noise)
- 3D visualisation and user interaction
- Database with objects (robots, objects, etc.)
- Provides a ROS Interface
- Extensible via plugins

rosrun gazebo_ros gazebo

Gazebo Simulator



http://gazebosim.org

Workspace environment

- Defines the context for the current workspace
- Default workspace loaded:

  source /opt/ros/indigo/setup.bash

- Overlay using catkin (http://docs.ros.org/api/catkin/html/)
- Check workspace:

  echo $ROS_PACKAGE_PATH

Starting a roscore

Start a roscore with

```
> roscore
```

```
student@ubuntu:~/catkin_ws$ roscore
... logging to /home/student/.ros/log/6c1852aa-e961-11e6-8543-000c297bd368/ros
launch-ubuntu-6696.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://ubuntu:34089/
ros_comm version 1.11.20


SUMMARY
========

PARAMETERS
 * /rosdistro: indigo
 * /rosversion: 1.11.20

NODES

auto-starting new master
process[master]: started with pid [6708]
ROS_MASTER_URI=http://ubuntu:11311/

setting /run_id to 6c1852aa-e961-11e6-8543-000c297bd368
process[rosout-1]: started with pid [6721]
started core service [/rosout]
```

Analyze the talker node

See the list of active nodes

```
> rosnode list
```

```
student@ubuntu:~/catkin_ws$ rosnode list
/rosout
/talker
```

Show information about the *talker* node

```
> rosnode info /talker
```

```
student@ubuntu:~/catkin_ws$ rosnode info /talker
--------------------------------------------------------------------------------
Node [/talker]
Publications:
 * /chatter [std_msgs/String]
 * /rosout [rosgraph_msgs/Log]

Subscriptions: None

Services:
 * /talker/get_loggers
 * /talker/set_logger_level
```

See information about the *chatter* topic

```
> rostopic info /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic info /chatter
Type: std_msgs/String

Publishers:
 * /talker (http://ubuntu:39173/)

Subscribers: None
```

Analyze chatter topic

Check the type of the *chatter* topic

```
> rostopic type /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic type /chatter
std_msgs/String
```

Show the message contents of the topic

```
> rostopic echo /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic echo /chatter
data: hello world 11874
---
data: hello world 11875
---
data: hello world 11876
```

Analyze the frequency

```
> rostopic hz /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic hz /chatter
subscribed to [/chatter]
average rate: 9.991
        min: 0.099s max: 0.101s std dev: 0.00076s window: 10
average rate: 9.996
        min: 0.099s max: 0.101s std dev: 0.00069s window: 20
```

Start a listener node

See the new *listener* node with

```
> rosnode list
```

```
student@ubuntu:~/catkin_ws$ rosnode list
/listener
/rosout
/talker
```

Show the connection of the nodes over the chatter topic with

```
> rostopic info /chatter
```

```
student@ubuntu:~/catkin_ws$ rostopic info /chatter
Type: std_msgs/String

Publishers:
 * /talker (http://ubuntu:39173/)

Subscribers:
 * /listener (http://ubuntu:34664/)
```

ROS Console Commands

- roscore
- rostopic
- rosnode
- rosrun
- roslaunch

http://wiki.ros.org/ROS/CommandLineTools

catkin Build System

- catkin => default ROS build system
- uses workspaces
- Catkin Command Line Tools (extension)
- Three spaces: **src**, build and devel

http://wiki.ros.org/catkin

ROS Launch

- launch is a tool for starting multiple nodes (together with their parameters)
- Launch files (XML) => *.launch

roslaunch [<package>] <name>.launch

ROS Launch file structure

- launch: Root element
- node: Each node specifies the node to be launched
  - name = name of the node
  - pkg = package
  - type = name of the executable
  - output = where should the output be displayed (screen, file)

http://wiki.ros.org/roslaunch/XML

Example

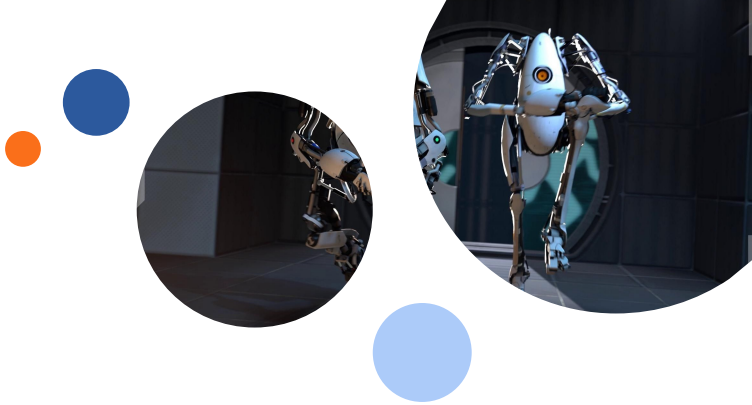
```
<launch>

    <node name="talker" pkg="rospy_tutorials" type="talker" />

    <node name=“listener” pkg="rospy_tutorials" type="listener" />

</launch>
```

ROS Launch



- Arguments: command line argument
- Include other launch files
- Pass down to arguments

```
<?xml version="1.0"?>
<launch>
  <arg name="use_sim_time" default="true"/>
  <arg name="world" default="gazebo_ros_range"/>
  <arg name="debug" default="false"/>
  <arg name="physics" default="ode"/>

  <group if="$(arg use_sim_time)">
    <param name="/use_sim_time" value="true" />
  </group>

  <include file="$(find gazebo_ros)
                                /launch/empty_world.launch">
    <arg name="world_name" value="$(find gazebo_plugins)/
                      test/test_worlds/$(arg world).world"/>
    <arg name="debug" value="$(arg debug)"/>
    <arg name="physics" value="$(arg physics)"/>
  </include>
</launch>
```

Further references

- ROS Wiki (http://wiki.ros.org)
- ROS Installation (http://wiki.ros.org/ROS/Installation)
- ROS Tutorials (http://wiki.ros.org/ROS/Tutorials)
- Available packages (http://ros.org/browse)
- ROS Cheat Sheet (https://github.com/ros/cheatsheet/releases/download/0.0.1/ROScheatsheet_catkin.pdf)
- ROS Best Practices (https://github.com/ethz-asl/ros_best_practices/wiki)
- ROS Package Template (https://github.com/ethz-asl/ros_best_practices/tree/master/ros_package_template)
- Courses (http://wiki.ros.org/Courses)

Thank you

Question Time