ROS – Robotic Operating System

…a one-shot learning experience…

…covering ROS1 and a little of ROS2…

Armando Sousa asousa@fe.up.pt

& Rafael Arrais (INESC TEC)

& Luís Paulo Reis (LIACC)

Attendance

Before Start…

https://spectrum.ieee.org/how-kaist-drc-hubo-won-darpa-robotics-challenge
http://blog.a-care.biz/2015/06/south-koreas-drc-hubo-robot-won-darpa.html

https://www.theguardian.com/technology/2015/jun/07/darpa-robotics-challenge-south-korea-wins

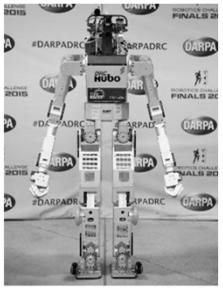


Photo: DARPA
This robot won the DARPA Robotics Challenge 2015.

https://youtu.be/7oCe0aLye-U

Driverless taxis take to the streets of San Francisco – BBC News
https://youtu.be/Y8qfHlpe31k

Lexicon

What is an Operating System

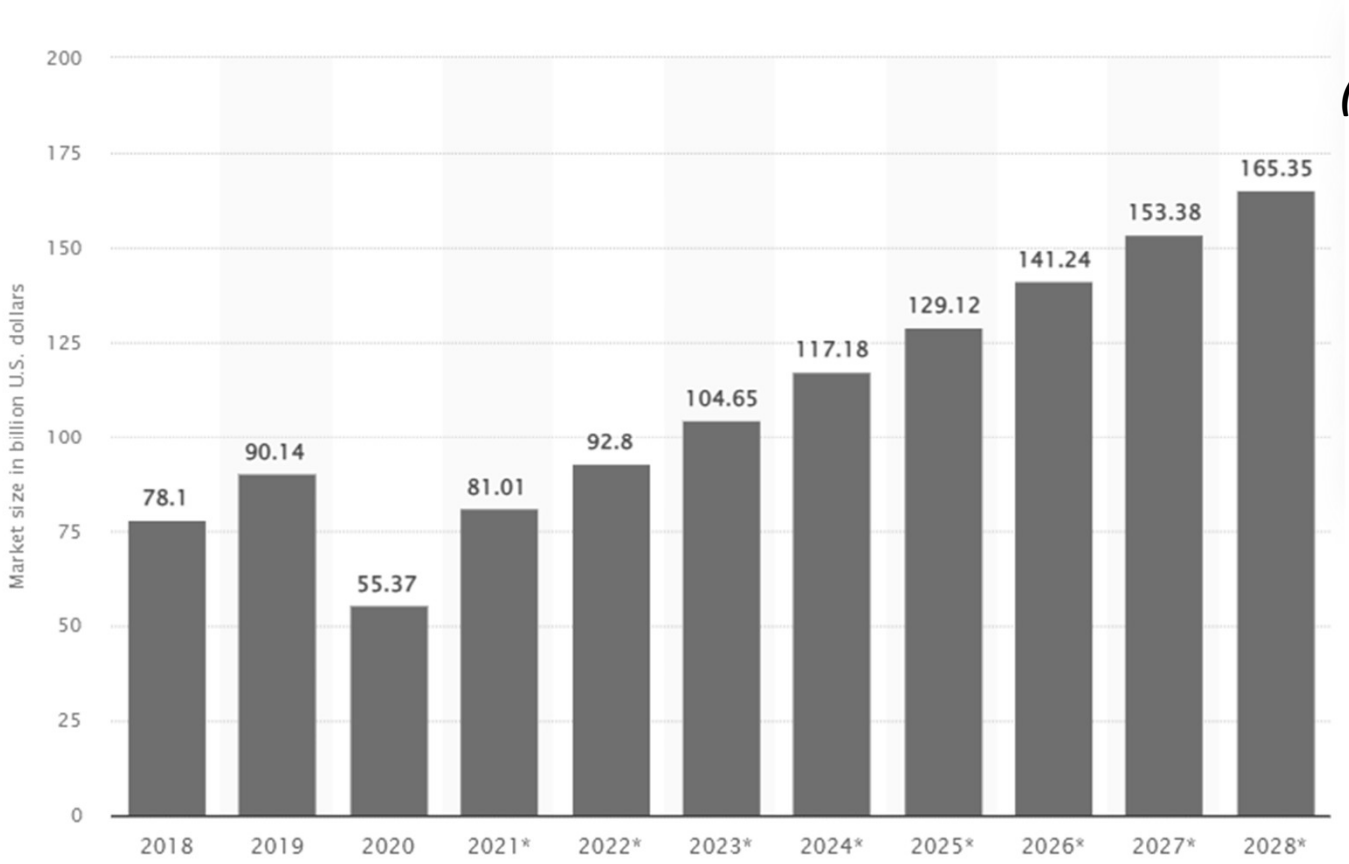
What is a robot?

Motivation

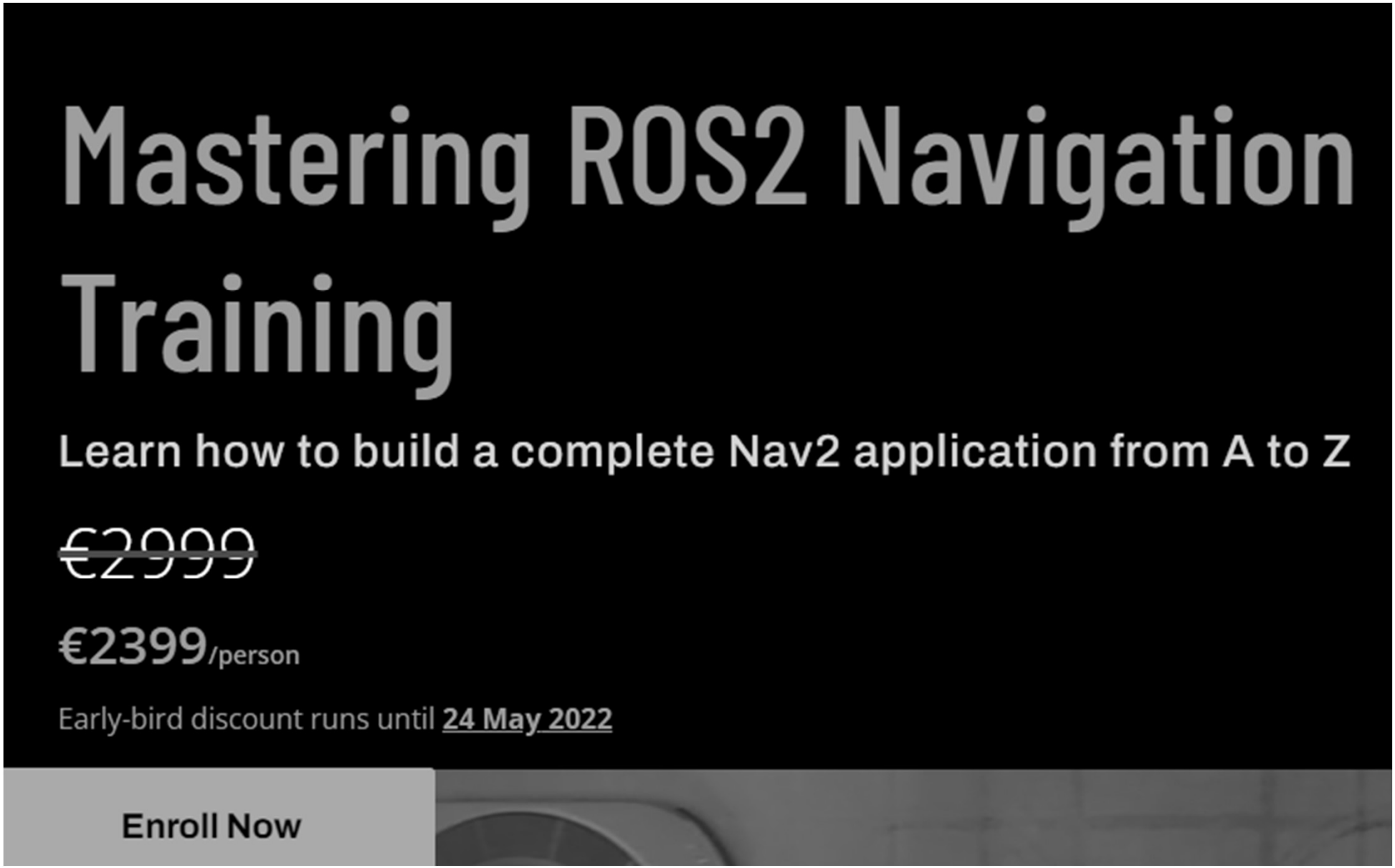
https://www.statista.com/statistics/728530/industrial-robot-market-size-worldwide/

Size of the market for industrial robots worldwide from 2018 to 2020, with a forecast through 2028

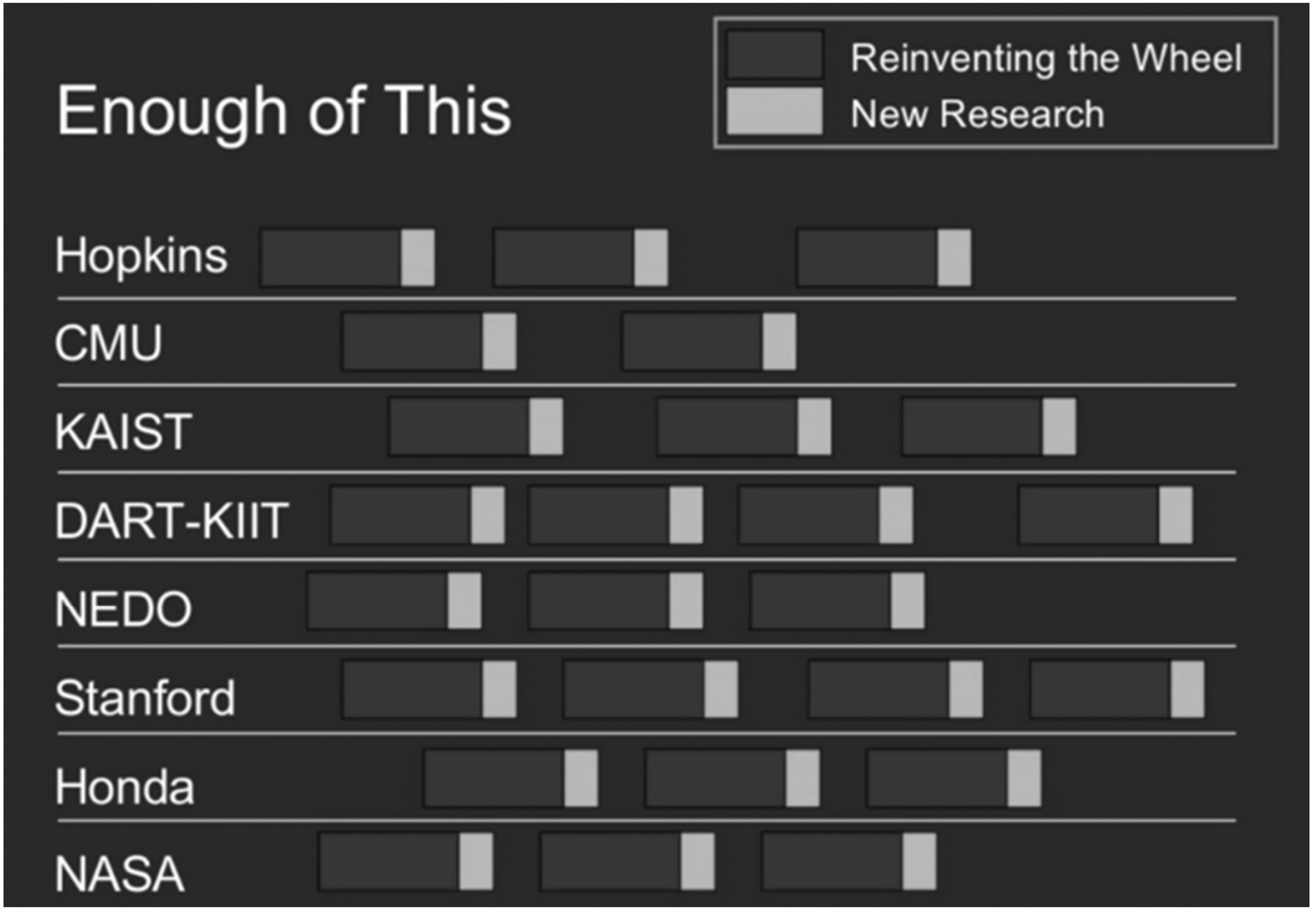
(in billion U.S. dollars)



https://www.theconstructsim.com/ros2-navigation-training/?utm_source=direct&utm_medium=blogfooter&utm_campaign=nav2workshop_0422



https://www.theconstructsim.com/history-ros/

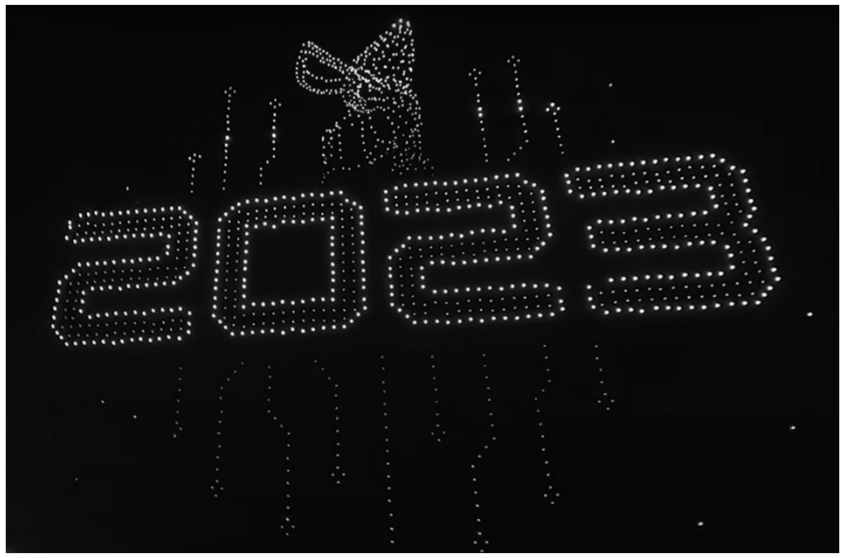


Totally Different Videos…

https://youtu.be/9PIuRFD443I?t=354

https://youtu.be/44KvHwRHb3A

https://youtu.be/01G1tei6OGg?t=293



Why?

https://www.ros.org/blog/why-ros/

http://wiki.ros.org/Industrial

ROS one-shot learning

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files

ROS one-shot learning

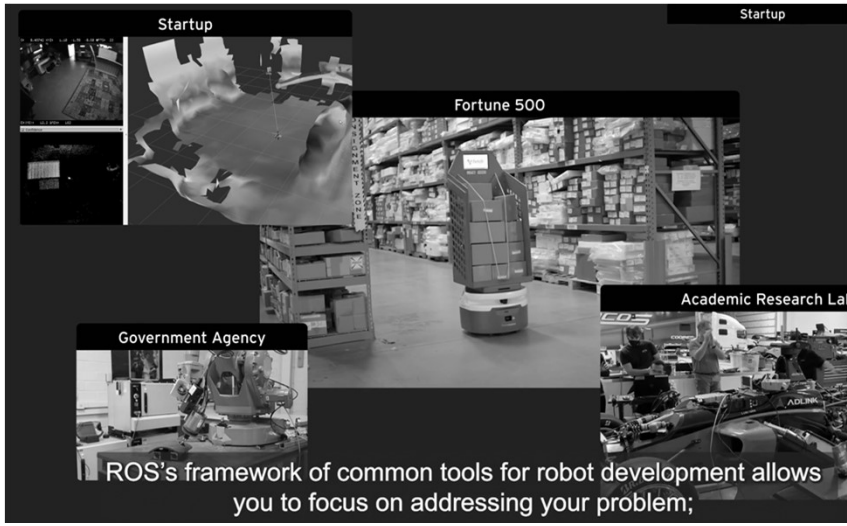
1. **What is ROS**
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files

What?

Videos from ROS.ORG

https://vimeo.com/639236696

(https://vimeo.com/637995822)
(…)



ROS: Powering the World's Robots

Not really an Operating System!

*"The **Robot ~~Operating System~~** (ROS) is a flexible **framework** for writing robot software.*

*It is a **collection** of **tools**, **libraries**, and **conventions** that aim to **simplify the task of creating complex and robust robot behavior** across a **wide variety of robotic platforms**."*

(Adapted from http://www.ros.org/about-ros/)

What is it?

A 'Meta' OS. Open Source!
- Sits on top of Linux (preferably Ubuntu)
- ROS1 “Kind of” Works in Windows SubSystem Linux (WSL)
- ROS2 with windows support

- Agent based
- Message passing
  - Publish / Subscribe
  - Service (remote operation) invocation

- Package Management
- Name and Parameter Services
- Programming Language Support
  - C++
  - Python

  (…)

http://wiki.ros.org/Distributions
https://docs.ros.org/en/rolling/Releases.html

Distro	Release date	Poster	*Tuturtle*, turtle in tutorial	EOL date
ROS Noetic Ninjemys (**Recommended**)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			May, 2023 (Bionic EOL)

Distro	Release date	Logo	EOL date
Humble Hawksbill	May 23rd, 2022		May 2027
Galactic Geochelone	May 23rd, 2021		November 2022
Foxy Fitzroy	June 5th, 2020		May 2023

Legend
- light yellow: future release
- green: supported release
- grey: unsupported release (End of Life)

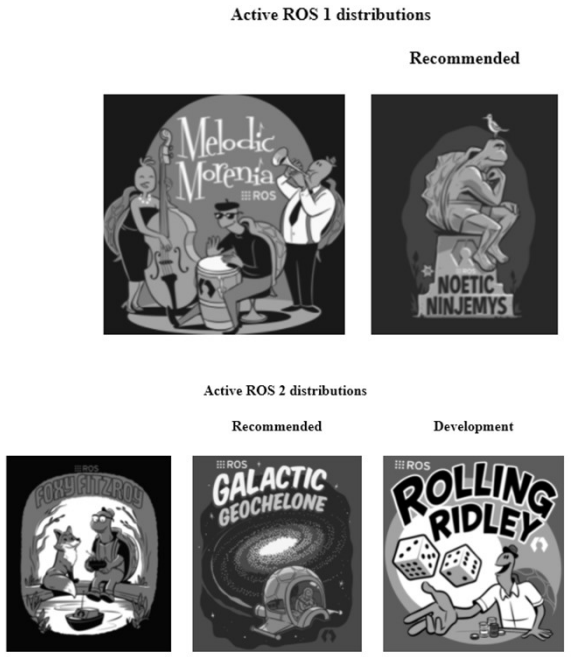
What is it?

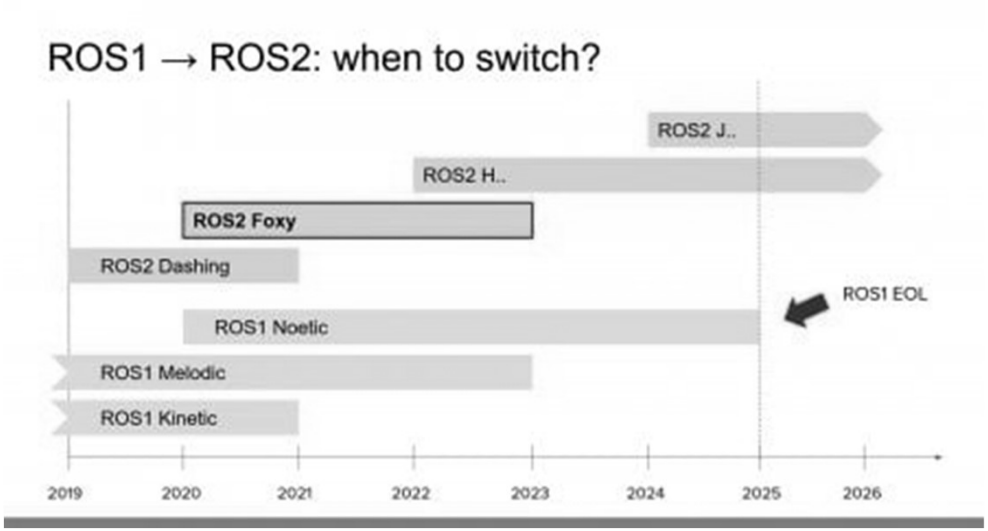
A 'Meta' OS. Open Source!
- Sits on top of Linux (preferably Ubuntu)
- ROS1 “Kind of” Works in Windows SubSystem Linux (WSL)
- ROS2 with windows support

- Agent based
- Message passing
  - Publish / Subscribe
  - Service (remote operation) invocation

- Package Management
- Name and Parameter Services
- Programming Language Support
  - C++
  - Python
  (…)

https://docs.ros.org/





ROS: Powering the World's Robots

*"**Why?** Because creating truly **robust**, **general-purpose robot software** is **hard**.*

*From the robot's perspective, problems that seem trivial to humans often **vary wildly** between **instances of tasks and environments**. **Dealing with these variations is so hard** that **no single individual**, **laboratory**, or **institution** can **hope to do it on their own**."*

(Adapted from http://www.ros.org/about-ros/)

ROS: Powering the World's Robots

*"As a result, **ROS** was built from the ground up to **encourage collaborative robotics software development**.*

*For example, one laboratory might have experts in **mapping indoor environments**, and could contribute a world-class system for producing maps. Another group might have experts at **using maps to navigate**, and yet another group might have discovered a **computer vision approach** that works well for recognizing small objects in clutter.*

ROS was designed specifically for groups like these to collaborate and build upon each other's work (…)"

(Adapted from http://www.ros.org/about-ros/)

De facto standard

https://blog.51cto.com/u_12369060/5171821

https://www.ros.org/blog/why-ros/

Why?

- Global Community
- Proven in Use
- Shorten Time to Market
- Multi-domain
- Multi-platform
- 100% Open-source
- Commercial Friendly
- Industry Support

Robotics Middleware
Hardware abstraction
Pub-Sub
ROS2 is Real Time aware

Also:
https://micro.ros.org/

Willow Garage

Open Source Robotics Foundation

https://spectrum.ieee.org/open-robotics

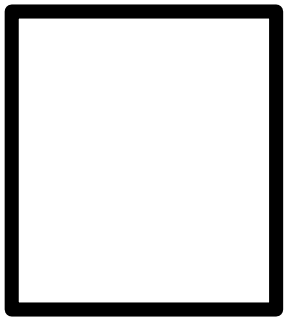
the Open Source Robotics Foundation announced that it was spinning out of Willow Garage

https://spectrum.ieee.org/open-source-robotics-foundation-officially-announced

What is ROS?

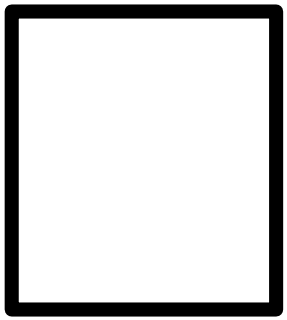
And it’s all open source ;)

What is ROS? ROS is… Plumbing!



(Adapted from Willow Garage's "What is ROS?" Presentation)

What is ROS? ROS is… Plumbing!

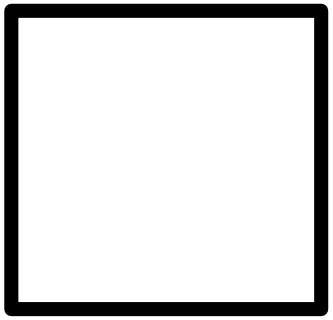


ROS Drivers Examples

- Perception Systems (2D/3D Cameras)
- Laser Scanners
- Robot Actuators
- Inertial Units
- Audio
- GPS
- Joysticks
- …

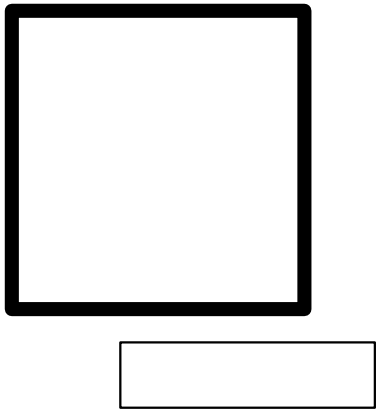
(Adapted from Morgan Quigley's "ROS: An Open-Source Framework for Modern Robotics" presentation)

What is ROS? ROS is… Tools!



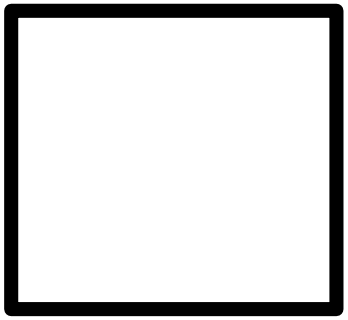
(Adapted from Willow Garage's "What is ROS?" Presentation)

What is ROS? ROS is… Capabilities!



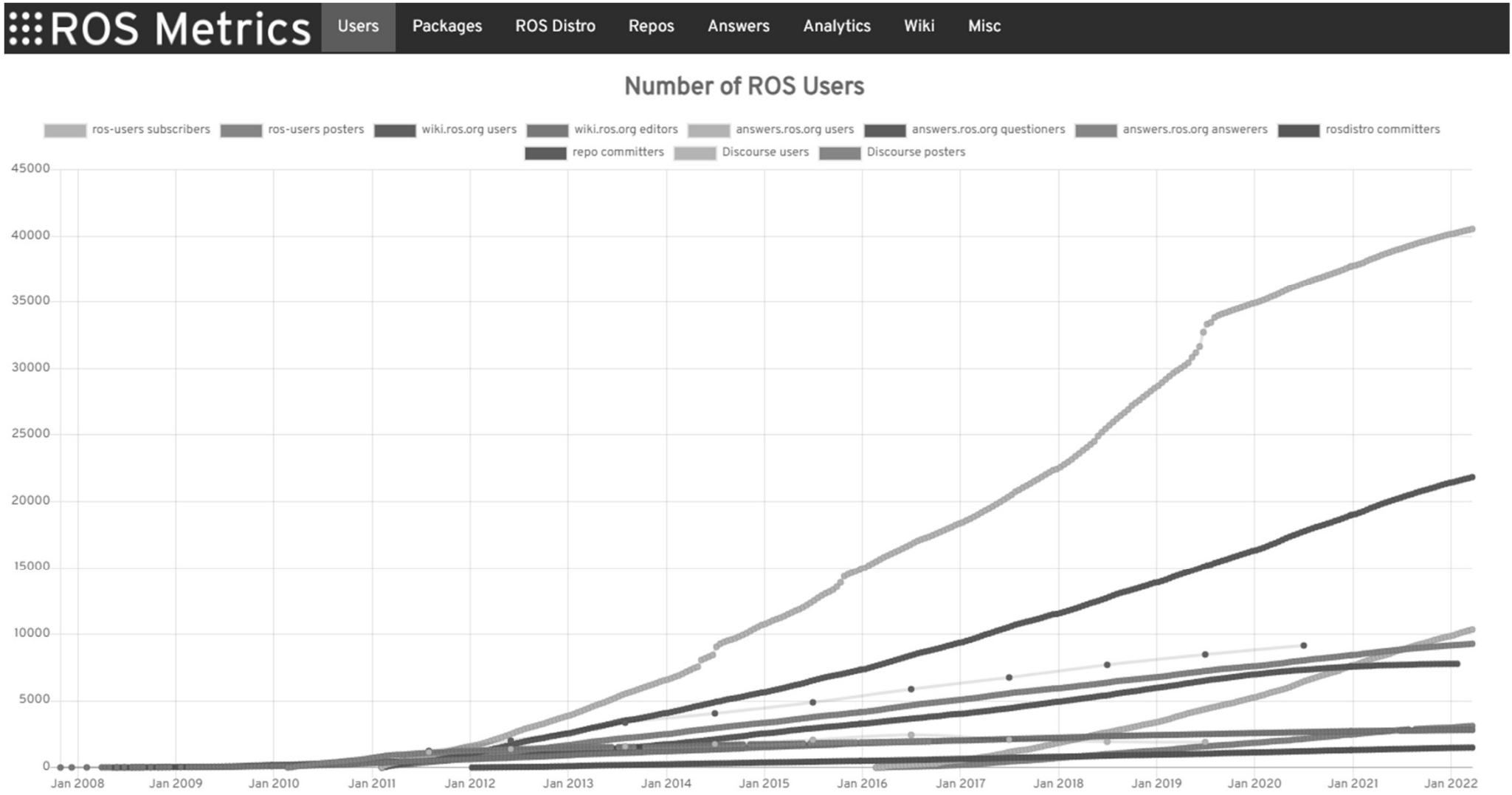
(Adapted from Willow Garage's "What is ROS?" Presentation)

What is ROS? ROS is… an Ecosystem!



(Adapted from ROS ROS-I Basic Developers Training)

De facto…



A collection of different metrics for measuring the number of users in the ROS community.

https://metrics.ros.org/

ROS is also… Growing (Scientifically)!

Total Number of Papers Citing "ROS: an open-source Robot Operating System" *(Quigley et al., 2009)*:

4652

(as of September 2018)

ROS is also… Growing (Commercially)!

Number of Different
Types of Robots
Available to the
community with
ROS drivers
(at robots.ros.org)
❼

(Adapted from ROS Wiki July 2017 Metrics: *wiki.ros.org/Metrics*)

ROS is also… International!

wiki.ros.org Visitor Locations:

(Adapted from ROS Wiki July 2017 Metrics: *wiki.ros.org/Metrics*)

ROS is also… a set of Repositories & Packages!

(Adapted from Morgan Quigley's "ROS: An Open-Source Framework for Modern Robotics" presentation)

ROS is also… a set of Repositories & Packages!

(Adapted from ROS Wiki July 2017 Metrics: *wiki.ros.org/Metrics*)

Celebrate Open Robotics ☺

ROS is also… Celebrating 10 years!

https://vimeo.com/245826128 https://youtu.be/mDwZ21Zia8s

(Adapted from ROS Wiki July 2017 Metrics: https://wiki.ros.org/Metrics)

ROS one-shot learning

1. What is ROS
2. **ROS Distros, Installation, Programming Languages**
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files

ROS 1 Versions: Annual Releases (“distributions”)

LTS

LTS

LTS

LTS

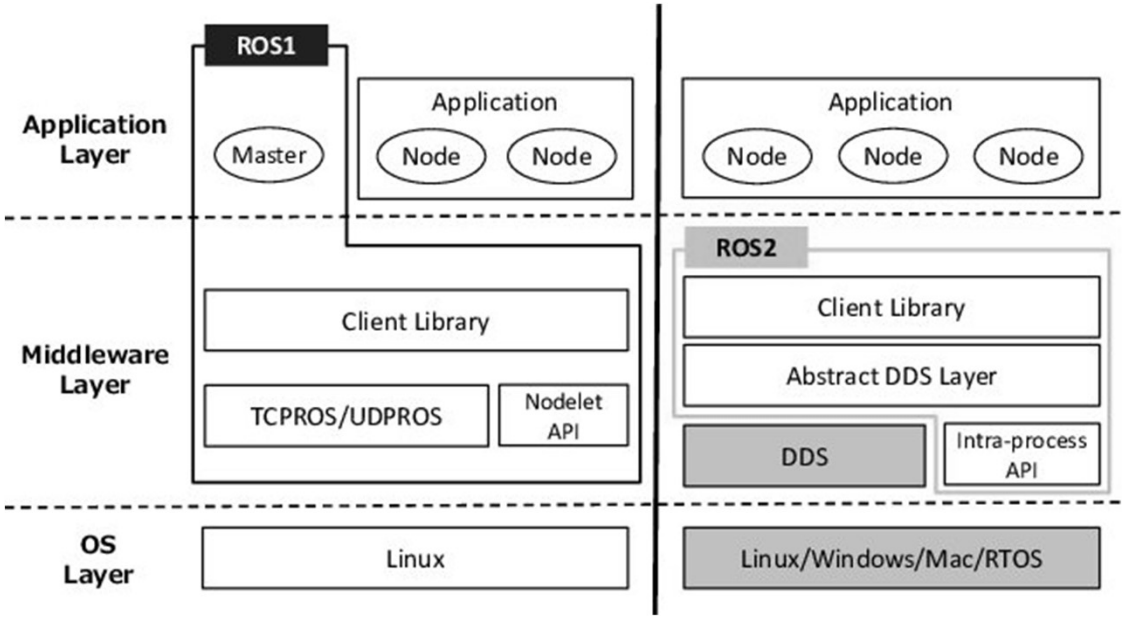
LTS

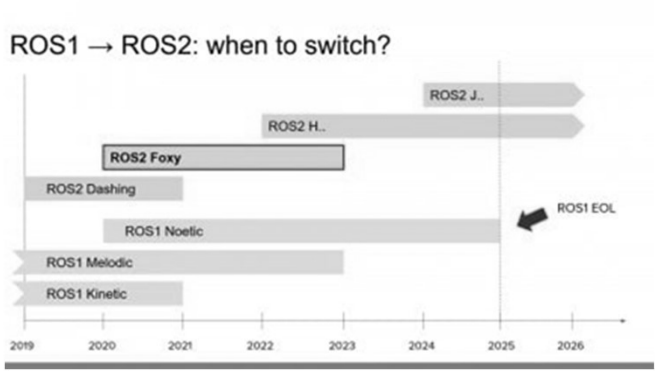
2020 - 2025

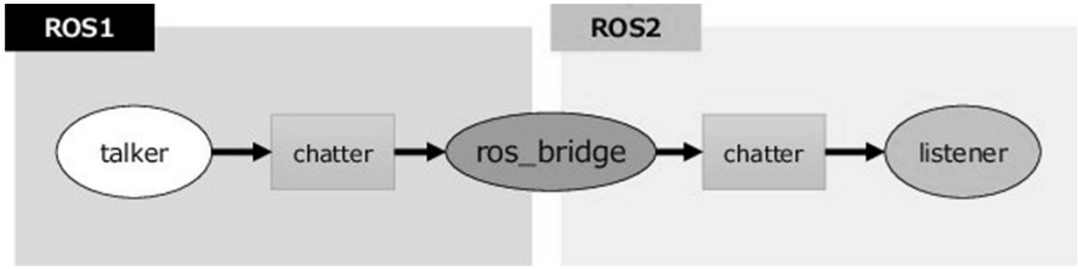
(Adapted from ROS ROS-I Basic Developers Training)

ROS 1 or 2?

https://www.researchgate.net/publication/309128426_Exploring_the_performance_of_ROS2/figures?lo=1

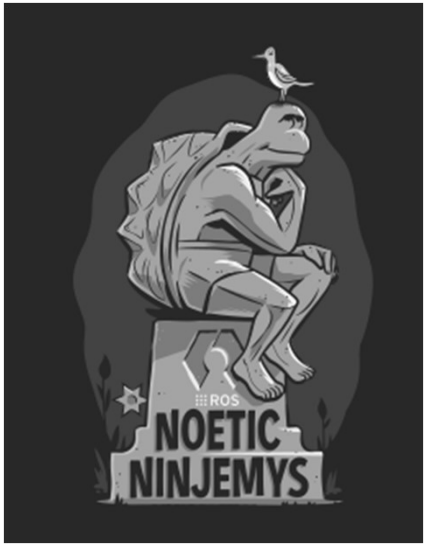






More information:
https://roboticsbackend.com/ros1-vs-ros2-practical-overview/
https://geekgasteiz.wordpress.com/2018/11/01/ros2-vs-ros-1-migramos/

Installing ROS 1



- ROS Wiki provides a detailed guide on installing and configuring ROS 1:

http://wiki.ros.org/noetic/Installation

http://wiki.ros.org/noetic/Installation/Source

[ROS Noetic Ninjemys is primarily targeted at the Ubuntu 20.04 (Focal) release]

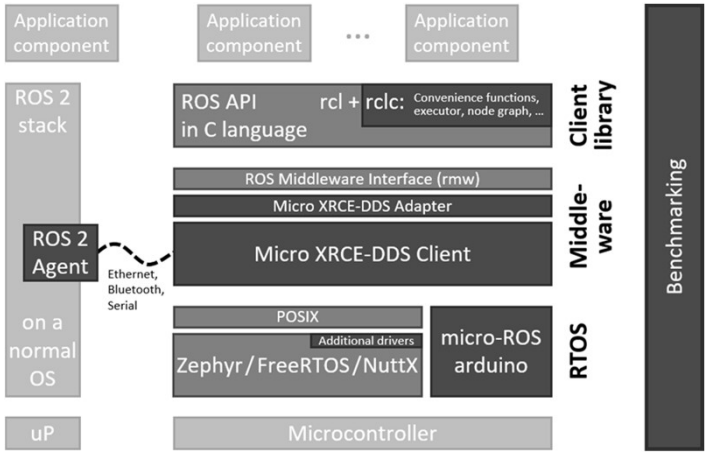
Ubuntu	Focal	amd64	armhf	arm64			
Debian	Buster	amd64	arm64				
Windows	10	amd64					
Arch Linux	Any	amd64	i686	arm	armv6h	armv7h	aarch64

Installing ROS 2

- ROS Wiki provides a detailed guide on installing and configuring ROS 2:

https://docs.ros.org/en/galactic/Installation.html

Ubuntu Linux - Focal Fossa (20.04) 64-bit
Ubuntu 20.04 (Focal): amd64 and arm64
Debian Linux - Bullseye (11) 64-bit
Debian Bullseye (11): amd64, arm64 and arm32
Windows 10 (Visual Studio 2019): amd64
Mac macOS 10.14 (Mojave): amd64





Running ROS 2 nodes in Docker

[community-contributed]
https://docs.ros.org/en/galactic/How-To-Guides/Run-2-nodes-in-single-or-separate-docker-containers.htm

ROS 1 vs 2 overview

https://roboticsbackend.com/ros1-vs-ros2-practical-overview/

Check out how to write a minimal ROS2 Python node, and a ROS2 Cpp node, with OOP.

Check out how to write a ROS2 launch file.

When creating a multi-machine ROS2 application, you won't have to define one machine as the "master".

Check out how to handle Parameters in your code: rclcpp params and rclpy params. And how to create Parameter callbacks: rclpp parameter callback, and rclpy parameter callback.

ROS2 introduces QoS, or Quality of Service.

ROS1 vs ROS2: Packages, workspace and environment

Using ROS1 and ROS2 together with the ros1_bridge package

https://www.udemy.com/course/ros2-for-beginners/?couponCode=SEP_23 (paid…)

ROS Programming

- ROS uses **platform-agnostic** methods for most communication:
  - TCP/IP Sockets, XML, etc.

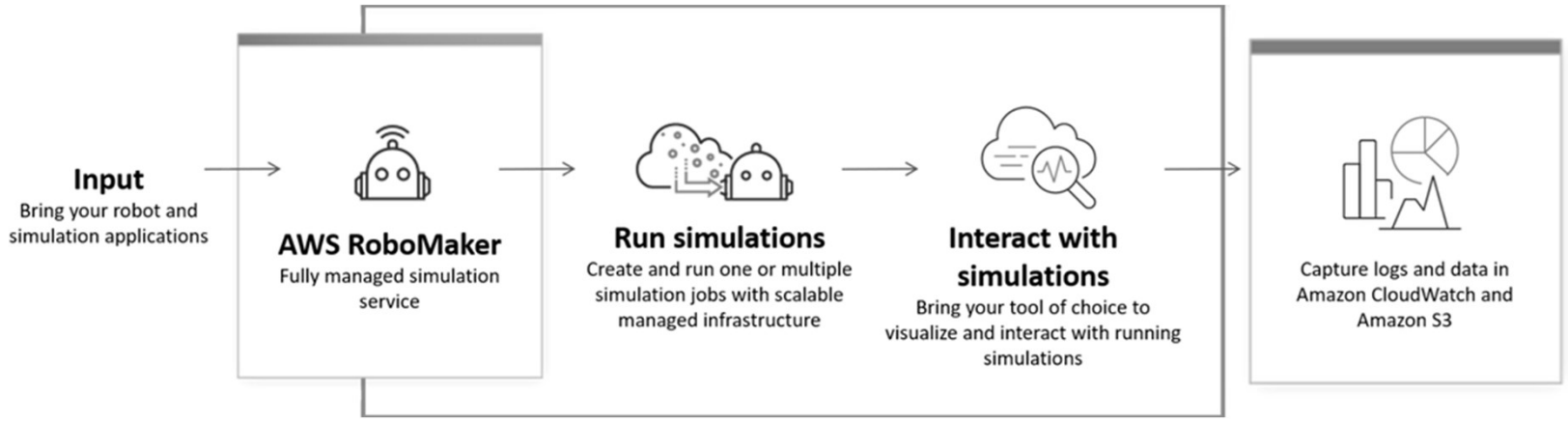
- Can intermix **Programming Languages**:
  - Primary: **C++**, **Python**, ~~Lisp~~
  - Also: ~~C#~~, ~~Java~~, ~~Matlab~~, ~~JavaScript~~, ~~etc.~~

Programming Languages other than C++ and Python are not really used a lot…

(Adapted from ROS ROS-I Basic Developers Training)

ROS in the cloud

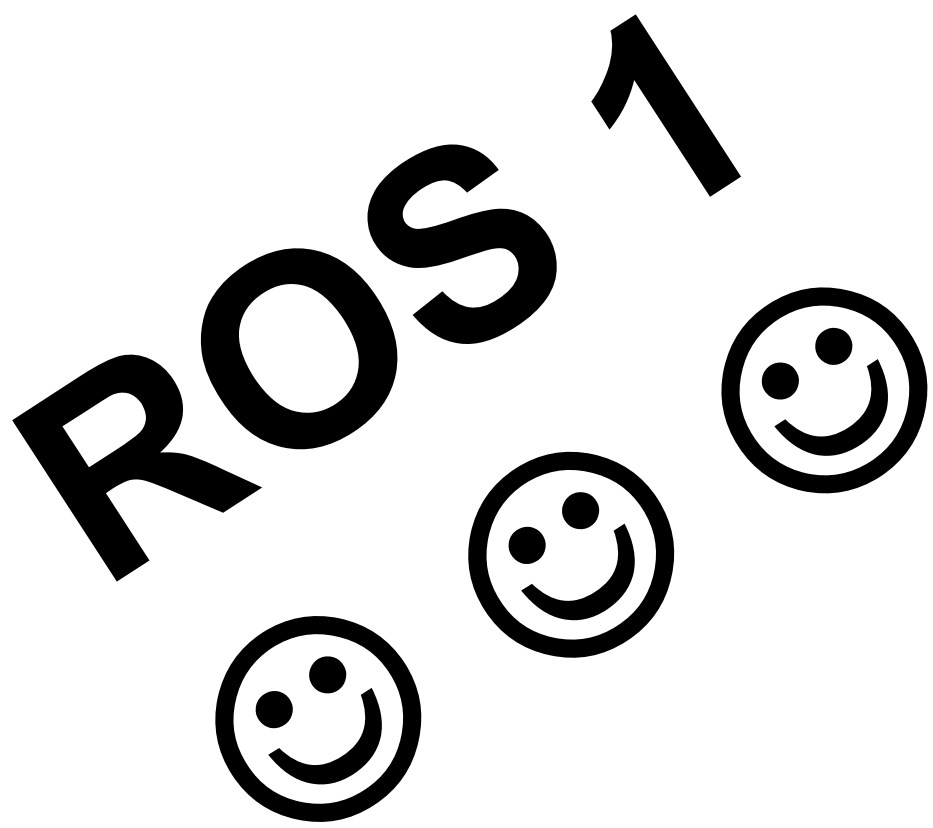
AWS RoboMaker



1 Simulation Unit, or SU (1 vCPU and 2 GB of memory)	$0.45 per hour
1 GPU Unit, or GU (1 GPU device)	$1.688 per hour
World Generation	$1.688 per world
World Export	$5.625 per world

Free tier: AWS RoboMaker is part of the AWS Free Tier program.
Upon first use of RoboMaker Simulation within your payer account and linked accounts, customers get to use 25 SU-hours without incurring any charges during the first month.
If your application uses the 25 SU-hours or the free tier expires after one month, you simply pay standard, pay-as-you-go rates.

https://aws.amazon.com/robomaker/pricing/


ROS 1

ROS one-shot learning

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. **ROS Architecture**
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files

ROS: High Level Architectural Overview

- **Robots** are **Cyber-Physical Systems**:
  - ROS acts as a software framework for enabling Sensors and Actuators to interact with the Physical Environment.

(Adapted from Morgan Quigley's "ROS: An Open-Source Framework for Modern Robotics" presentation)

ROS: High Level Architectural Overview

Software

Simulation

Hardware

- Break Complex Software into Smaller Pieces
- Provide a framework, tools, and interfaces for distributed development
- Encourage re-use of software pieces
- Easy transition between simulation and hardware

(Adapted from Morgan Quigley's "ROS: An Open-Source Framework for Modern Robotics" presentation)

ROS 1 Architecture: Nodes

- A **Node** is a single ROS-enabled program:
  - Most communication happens **between** nodes.
  - Nodes can run on many different **devices.**
- One **Master** per system.

(Adapted from ROS ROS-I Basic Developers Training)

ROS Architecture: Packages

- ROS **Packages** are groups of related nodes/data
  - Many ROS commands are **package-oriented**

(Adapted from ROS ROS-I Basic Developers Training)

ROS Architecture: MetaPkg

- **MetaPackages** are groups of related packages
  - Mostly for convenient install/deployment

(Adapted from ROS ROS-I Basic Developers Training)

ROS Build System: Catkin

- ROS uses the **catkin** build system:
  - based on CMAKE

Only in theory… Practical use is linux (Ubuntu) based.

  - cross-platform (Ubuntu, Windows, embedded...)
  - replaces older *rosbuild* system:
    - different build commands, directory structure, etc.
    - most packages have already been upgraded to *Catkin*
    - *rosbuild: manifest.xml, catkin: package.xml*

Today there are very few ROS packages that still used *rosbuild*. The change from *rosbuild* to *Catkin* occurred around **5 years ago**.

(Adapted from ROS ROS-I Basic Developers Training)

ROS Build System: Catkin Workspace

- **Catkin** uses a specific directory structure:
  - Each “project” typically gets its own **catkin workspace**
  - All packages/source files go in the **src** directory
  - Temporary build-files are created in **build**
  - Results are placed in **devel**

(Adapted from ROS ROS-I Basic Developers Training)

ROS Build System: Catkin Build Process

- **Setup (one-time):**
  1. Create a *catkin* workspace somewhere:
     - **catkin_ws**
     - **src** sub-directory must be created manually
     - **build**, **devel** directories created automatically
  2. Run **catkin init** from workspace root
  3. Download/create **packages** in **src** subdir

- **Compile-Time:**
  1. Run **catkin build** anywhere in the workspace
  2. Run **source devel/setup.bash** to make workspace visible to ROS
     - Must re-execute in each new terminal window
     - Can add to ~/.bashrc to automate this process

(Adapted from ROS ROS-I Basic Developers Training)

Adding 3rd-Party Packages: Install Options

- Two interchangeable options for installing 3rd Party ("Resources") Packages:
  - **Debian Packages:**
    - Nearly "automatic"
    - Recommended for end-users
    - Stable
    - Easy
  - **Source Repositories:**
    - Access "latest" code
    - Most at Github.com
    - More effort to setup
    - (Can be) Unstable

ROS Website (*ros.org/browse*) can be used to browse/search for known packages

The ROS Community is also very active on *Github.com*

(Adapted from ROS ROS-I Basic Developers Training)

Adding 3rd-Party Packages: Install using Debian Packages

- **Fully automatic install:**
  1. Download `.deb` package from central ROS repository
  2. The process automatically copies files to standard locations (`/opt/ros/kinetic/...`)
  3. The process also installs any other required dependencies

- `sudo apt-get remove ros-distro-package`
  - Removes software (but not dependencies!)

(Adapted from ROS ROS-I Basic Developers Training)

Adding 3rd-Party Packages: Install from Source

- **Somewhat manual process:**

1. **Find** GitHub repository.
2. **Clone** repository into your workspace `src` directory:

3. **Build** your catkin workspace:

4. Now the package and its resources are available to you.

(Adapted from ROS ROS-I Basic Developers Training)

Extra

WORKSPACES

Create Workspace

```
mkdir catkin_ws && cd catkin_ws
wstool init src
catkin_make
source devel/setup.bash
```

Add Repo to Workspace

```
roscd; cd ../src
wstool set repo_name \
--git http://github.com/org/repo_name.git \
--version=melodic-devel
wstool up
```

Resolve Dependencies in Workspace

```
sudo rosdep init  # only once
rosdep update
rosdep install --from-paths src --ignore-src \
--rosdistro=${ROS_DISTRO} -y
```

PACKAGES

Create a Package

```
catkin_create_pkg package_name [dependencies ...]
```

Package Folders

`include/package_name`	C++ header files
`src`	Source files. Python libraries in subdirectories
`scripts`	Python nodes and scripts
`msg, srv, action`	Message, Service, and Action definitions

Release Repo Packages

```
catkin_generate_changelog
# review & commit changelogs
catkin_prepare_release
bloom-release --track melodic --ros-distro melodic repo_name
```

Reminders

- Testable logic
- Publish diagnostics
- Desktop dependencies in a separate package

CMakeLists.txt

Skeleton

```
cmake_minimum_required(VERSION 2.8.3)
project(package_name)
find_package(catkin REQUIRED)
catkin_package()
```

Package Dependencies

To use headers or libraries in a package, or to use a package's exported CMake macros, express a build-time dependency:

```
find_package(catkin REQUIRED COMPONENTS roscpp)
```

Tell dependent packages what headers or libraries to pull in when your package is declared as a catkin component:

```
catkin_package(
  INCLUDE_DIRS include
  LIBRARIES ${PROJECT_NAME}
  CATKIN_DEPENDS roscpp)
```

Note that any packages listed as `CATKIN_DEPENDS` dependencies must also be declared as a `<run_depend>` in `package.xml`.

Messages, Services

These go after find_package(), but before catkin_package().
Example:

```
find_package(catkin REQUIRED COMPONENTS message_generation
std_msgs)
add_message_files(FILES MyMessage.msg)
add_service_files(FILES MyService.msg)
generate_messages(DEPENDENCIES std_msgs)
catkin_package(CATKIN_DEPENDS message_runtime std_msgs)ww
```

Build Libraries, Executables

Goes after the `catkin_package()` call.

```
add_library(${PROJECT_NAME} src/main)
add_executable(${PROJECT_NAME}_node src/main)
target_link_libraries(
  ${PROJECT_NAME}_node ${catkin_LIBRARIES})
```

Installation

```
install(TARGETS ${PROJECT_NAME}
  DESTINATION ${CATKIN_PACKAGE_LIB_DESTINATION})
install(TARGETS ${PROJECT_NAME}_node
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
install(PROGRAMS scripts/myscript
  DESTINATION ${CATKIN_PACKAGE_BIN_DESTINATION})
install(DIRECTORY launch
  DESTINATION ${CATKIN_PACKAGE_SHARE_DESTINATION})
```

RUNNING SYSTEM

Run ROS using plain:

```
roscore
```

Alternatively, roslaunch will run its own roscore automatically if it can't find one:

```
roslaunch my_package package_launchfile.launch
```

Suppress this behaviour with the `--wait` flag.

Nodes, Topics, Messages

```
rosnode list
rostopic list
rostopic echo cmd_vel
rostopic hz cmd_vel
rostopic info cmd_vel
rosmsg show geometry_msgs/Twist
```

Remote Connection

Master's ROS environment:

- `ROS_IP` or `ROS_HOSTNAME` set to this machine's network address.
- `ROS_MASTER_URI` set to URI containing that IP or hostname.

Your environment:

- `ROS_IP` or `ROS_HOSTNAME` set to your machine's network address.
- `ROS_MASTER_URI` set to the URI from the master.

To debug, check ping from each side to the other, run roswtf on each side.

ROS Console

Adjust using rqt_logger_level and monitor via rqt_console. To enable debug output across sessions, edit the `$HOME/.ros/config/rosconsole.config` and add a line for your package:

```
log4j.logger.ros.package_name=DEBUG
```

And then add the following to your session:

```
export ROSCONSOLE_CONFIG_FILE=$HOME/.ros/config/rosconsole.config
```

Use the roslaunch --screen flag to force all node output to the screen, as if each declared `<node>` had the `output="screen"` attribute.

CLEARPATH ROBOTICS™

www.clearpathrobotics.com/ros-cheat-sheet

ROS.org

https://github.com/YueErro/cheatsheets/blob/master/cheat_sheets/ROS2/ROS2.md

- Filesystem management
  - ros2 pkg
- Process launch
- Running system
  - ros2 node
  - ros2 topic
  - ros2 service
  - ros2 action
  - ros2 param
  - ros2 msg and ros2 srv
  - ros2 bag
- URDF and xacro

https://docs.ros.org/en/foxy/Releases.html

Distro	Release date		EOL date
Iron Irwini	May 23rd, 2023		November 2024
Humble Hawksbill	May 23rd, 2022		May 2027

ROS one-shot learning

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. **ROS Packages**
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files

ROS Packages: Contents

- ROS components are organized into **packages**

- Packages contain several **required files**:
  - package.xml
    - **metadata** for ROS: *package name, description, dependencies, ...*
  - CMakeLists.txt
    - **build rules** for *catkin*

Can also require a **setup.py** file for **Python Packages**

(Adapted from ROS ROS-I Basic Developers Training)

ROS Packages: package.xml

- **Metadata**: *name, description, author, license ...*

The package manifest is an XML file called package.xml that must be included with any catkin-compliant package's root folder. This file defines properties about the package such as the package name, version numbers, authors, maintainers, and dependencies on other catkin packages.

(Adapted from ROS ROS-I Basic Developers Training)

ROS Packages: package.xml

- **Metadata**: *name, description, author, license ...*
- **Dependencies:**
  - <buildtool_depend>: Needed to **build** itself. (Typically *catkin*)
  - <depend>: Needed to **build**, **export**, and **execution** dependency. *(format "2" only)*
  - <build_depend>: Needed to **build** this package.
  - <build_export_depend>: Needed to **build against** this package.
  - <exec_depend>: Needed to **run** code in this package.
  - <test_depend>: Only additional dependencies for unit **tests**.
  - <doc_depend>: Needed to generate **documentation**.

Format 2 is the recommended format for new packages.

<build_depend> and **<exec_depend>** are the most commonly used dependency tags within package.xml

(Adapted from ROS ROS-I Basic Developers Training)

ROS Packages: package.xml

- **Realistic example:**

(Adapted from ROS Wiki: *wiki.ros.org/catkin/package.xml*)

ROS Packages: CMakeList.txt

- The file **CMakeLists.txt** is the **input** to the **CMake build system** for **building software packages**.

- Any CMake-compliant package contains **one or more** CMakeLists.txt file that describe **how to build the code** and **where to install it to**.

- The CMakeLists.txt file used for a ROS-based *catkin* project is a **standard vanilla** CMakeLists.txt file with a **few additional constraints**.

(Adapted from ROS Wiki: *wiki.ros.org/catkin/CMakeLists.txt*)

ROS Packages: CMakeList.txt

- Provides **rules** for **building software**
  - The template file contains many examples

- **include_directories**(include ${catkin_INCLUDE_DIRS})
  - Adds directories to CMAKE include rules

- **add_executable**(myNode src/myNode.cpp src/widget.cpp)
  - Builds program myNode, from myNode.cpp and widget.cpp

- **target_link_libraries**(myNode ${catkin_LIBRARIES})
  - Links node myNode to dependency libraries

(Adapted from ROS ROS-I Basic Developers Training)

ROS Packages: CMakeList.txt

- **Realistic Example:**

(Adapted from ROS Wiki: *wiki.ros.org/catkin/package.xml*)

ROS Packages: setup.py

- If your ROS package contains **Python modules** and **scripts** to **install**, you need to define the **installation process** and a way to make the **scripts accessible** in the develspace.

- The **setup.py** file uses Python to describe the Python content of the stack Catkin allows you to specify the **installation** of your **Python files** in this `setup.py` and reuse some of the information in your `CMakeLists.txt`.

(Adapted from ROS Docs: *docs.ros.org/api/catkin/html/user_guide/setup_dot_py.html*)

ROS Packages: Create New Packages

- Easiest way to start a **New Package**:
  - Create directory, required files
  - **mypkg**: name of package to be created
  - **dep 1/2**: dependency package names
    - Automatically added to `CMakeList.txt` and `package.xml`
    - Can manually add additional dependencies later

(Adapted from ROS ROS-I Basic Developers Training)

ROS Packages: Other Useful Commands

- **roscd package_name**
  - Change to package directory

- **rospack**
  - **rospack find package_name**
    - Find directory of package_name

  - **rospack list**
    - List all ROS packages installed

  - **rospack depends package_name**
    - List all dependencies of package_name

(Adapted from ROS ROS-I Basic Developers Training)

ROS one-shot learning

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. **ROS Nodes**
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files

ROS Nodes: Overview

- All running nodes have a **graph resource name** that **uniquely identifies** them to the rest of the system.
  - *For example:* /hokuyo_node *could be the name of a Hokuyo driver broadcasting laser scans.*

- Nodes also have a **node type**, that simplifies the process of referring to a node executable on the filesystem.
  - These node types are **package resource names** with the name of the node's package and the name of the node executable file.
  - In order to resolve a node type, ROS searches for all executables in the package with the specified name and chooses the first that it finds.

- A ROS node is written with the use of a ROS client library, such as **roscpp** or **rospy**.

(Adapted from ROS Wiki: *wiki.ros.org/Nodes*)

ROS Nodes: A Simple C++ ROS1 Node

(Adapted from ROS ROS-I Basic Developers Training)

ROS1 Concepts: roscore

- **roscore** is a collection of nodes and programs that are pre-requisites of a ROS-based system

- You must have a **roscore** running in order for ROS nodes to communicate. It is launched using the `roscore` command.

- **roscore** will start up:
  - a ROS Master;
  - a ROS Parameter Server;
  - a *rosout* logging node.

- ***NOTE***: If you use *roslaunch*, it will automatically start *roscore* if it detects that it is not already running.

(Adapted from ROS Wiki: *wiki.ros.org/roscore*)

ROS Nodes1: Useful Commands

- **rosrun package_name node_name**
  - Execute ROS node

- **rosnode**
  - **rosnode list**
    - View running nodes

  - **rosnode info node_name**
    - View node details *(publishers, subscribers, services, etc.)*

  - **rosnode kill node_name**
    - Kill running node
    - Good for remote machines
    - *Ctrl+C is usually easier*

(Adapted from ROS ROS-I Basic Developers Training)

ROS Nodes: Example Graph

(Adapted from ROS ROS-I Basic Developers Training)

ROS one-shot learning

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. **ROS Topics & Messages**
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files

ROS Topics: Overview

- **Topics** are **named buses** over which **nodes exchange messages**.

- Topics have **anonymous publish/subscribe semantics**, which **decouples the production of information from its consumption**.
  - In general, **nodes are not aware of who they are communicating with**.
  - Instead, nodes that are interested in data **subscribe** to the relevant topic;
  - Nodes that generate data **publish** to the relevant topic.
  - There can be **multiple publishers and subscribers** to a topic.

- Topics are intended for **unidirectional, streaming communication**.
  - Nodes that need to perform remote procedure calls, *i.e. receive a response to a request, should use* **services** *instead*.
  - There is also the **Parameter Server** for maintaining small amounts of *(static)* state.

Services and **Parameters** will be addressed later on in this tutorial!

(Adapted from ROS Wiki: *wiki.ros.org/Topics*)

ROS Topics: Types

- **ROS Topics Types:**
  - Each topic is strongly typed by the **ROS message type** used to publish to it and nodes can only receive messages with a matching type.
  - The **Master does not enforce type consistency** among the publishers, but subscribers will not establish message transport unless the types match.
  - Furthermore, all ROS clients check to make sure that an MD5 computed from the msg files match.
    - This check ensures that the ROS Nodes were compiled from consistent code bases.

(Adapted from ROS Wiki: *wiki.ros.org/Topics*)

ROS Topics: Transports Systems

- **ROS Topics Transport Systems:**

This will be a **major** change in **ROS2**!

ROS currently supports TCP/IP-based and UDP-based message transport:
The TCP/IP-based transport is known as TCPROS and streams message data over persistent TCP/IP connections. TCPROS is the default transport used in ROS and is the only transport that client libraries are required to support.
The UDP-based transport, which is known as UDPROS and is currently only supported in roscpp, separates messages into UDP packets. UDPROS is a low-latency, lossy transport, so is best suited for tasks like teleoperation.

ROS nodes negotiate the desired transport at runtime.
For example:
if a node prefers UDPROS transport but the other Node does not support it,
it can fallback on TCPROS transport.
This negotiation model enables new transports to be added over time as compelling use cases arise.

(Adapted from ROS Wiki: *wiki.ros.org/Topics*)

ROS Topics: Details

- Each **Topic** is a stream of **Messages**:
  - Sent by **publisher(s)**, received by **subscriber(s)**

- Messages are **asynchronous**
  - Publishers don't know if anyone's listening
  - Messages may be dropped
  - Subscribers are event-triggered *(by incoming messages)*

- Typical Uses:
  - Sensor Readings: *camera images, distance, I/O*
  - Feedback: *robot status/position*
  - Open-Loop Commands: *desired position*

(Adapted from ROS ROS-I Basic Developers Training)

ROS Messages: Overview

- Nodes communicate with each other by **publishing messages** to **topics**.

- A **message** is a **simple data structure**, comprising **typed fields**.
  - Standard **primitive types** *(integer, floating point, boolean, etc.)* are supported, as are arrays of primitive types.
  - Messages can include **arbitrarily nested structures and arrays** *(much like C structs)*.

- Nodes can also exchange a request and response message as part of a **ROS service call**. These request and response messages are defined in **`srv`** files.

Services will be addressed later on in this tutorial!

(Adapted from ROS Wiki: *wiki.ros.org/Messages*)

ROS Messages: Types

- **Similar to C structures**

- **Standard data primitives**:
  - **Boolean**: bool
  - **Integer**: int8,int16,int32,int64
  - **Unsigned Integer**: uint8,uint16,uint32,uint64
  - **Floating Point**: float32, float64
  - **String**: string
  - **Fixed length arrays**: bool[16]
  - **Variable length arrays**: int32[]

- Other:
  - Nest message types for more complex data structure

(Adapted from ROS ROS-I Basic Developers Training)

ROS Messages: Message Description File

- All Messages are defined by a **.msg** file

(Adapted from ROS ROS-I Basic Developers Training)

ROS Messages: Custom ROS Messages

- **Custom message types** are defined in **msg** subfolder of packages

- **Modify** CMakeLists.txt & package.xml to enable **message generation**.

CMakeList.txt Lines needed to **generate custom message types:**

```
find_package(catkin REQUIRED COMPONENTS
message_generation)  add_message_files(custom.msg ...)
generate_messages(DEPENDENCIES ...)
catkin_package(CATKIN_DEPENDS roscpp message_runtime)
```

package.xml lines needed to **generate custom message types**:

```
<build_depend>message_generation</build_depend>
<build_export_depend>message_runtime</build_export_depend>
<run_depend>message_runtime</run_depend>
```

(Adapted from ROS ROS-I Basic Developers Training)

ROS Topics / Messages

(Adapted from ROS ROS-I Basic Developers Training)

ROS Topics / Messages: Topics vs. Messages

- **Topics** are **channels**, **Messages** are **data types**.
  - Different topics can use the same Message type

(Adapted from ROS ROS-I Basic Developers Training)

ROS Topics / Messages: Practical Example

(Adapted from ROS ROS-I Basic Developers Training)

ROS Topics / Messages: Multiple Publishers / Subscribers

- Many nodes can **Publish** or **Subscribe** to the same Topic
  - Communications are direct **node-to-node**

(Adapted from ROS ROS-I Basic Developers Training)

ROS Topics / Messages: ROS Topics Syntax

- **ROS Topic Publisher**:
  - Advertises available topic *(Name, Data Type)*
  - Populates message data
  - Periodically publishes new data

(Adapted from ROS ROS-I Basic Developers Training)

ROS Topics / Messages: ROS Topics Syntax

- **ROS Topic Subscriber**:
  - Defines callback function
  - Listens for available topic *(Name, Data Type)*

(Adapted from ROS ROS-I Basic Developers Training)

ROS Topics / Messages: ROS Messages Commands

- **rosmsg list**
  - Show all ROS topics currently installed on the system
- **rosmsg package <package>**
  - Show all ROS message types in package <package>
- **rosmsg show <package>/<message_type>**
  - Show the structure of the given message type

(Adapted from ROS ROS-I Basic Developers Training)

ROS Topics / Messages: ROS Topics Commands

- **rostopic list**
  - List all topics currently subscribed to and/or publishing
- **rostopic type <topic>**
  - Show the message type of the topic
- **rostopic info <topic>**
  - Show topic message type, subscribers, publishers, etc.
- **rostopic echo <topic>**
  - Echo messages published to the topic to the terminal
- **rostopic find <message_type>**
  - Find topics of the given message type

(Adapted from ROS ROS-I Basic Developers Training)

More information

https://people.eng.unimelb.edu.au/pbeuchat/asclinic/software/ros.html

- Writing a Simple Publisher and Subscriber (C++)
- Writing a Simple Publisher and Subscriber (Python)
- Examining the Simple Publisher and Subscriber

```
#include "ros/ros.h"
#include "std_msgs/String.h"
#include <sstream>

int main(int argc, char **argv)
{
  ros::init(argc, argv, "talker");
  ros::NodeHandle n;
  ros::Publisher chatter_pub = n.advertise<std_msgs::String>("chatter", 1000);
  ros::Rate loop_rate(10);
  int count = 0;
  while (ros::ok())
  {
    std_msgs::String msg;
    std::stringstream ss;
    ss << "hello world " << count;
    msg.data = ss.str();
    ROS_INFO("%s", msg.data.c_str());
    chatter_pub.publish(msg);
    ros::spinOnce();
    loop_rate.sleep();
    ++count;
  }
  return 0;
}
```

```
#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import String

def talker():
    pub = rospy.Publisher('chatter', String, queue_size=10)
    rospy.init_node('talker', anonymous=True)
    rate = rospy.Rate(10) # 10hz
    while not rospy.is_shutdown():
        hello_str = "hello world %s" % rospy.get_time()
        rospy.loginfo(hello_str)
        pub.publish(hello_str)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

Minimal ROS2 Nodes

https://roboticsbackend.com/write-minimal-ros2-python-node/

```
import rclpy
from rclpy.node import Node
def main(args=None):
rclpy.init(args=args)
node = Node('my_node_name’)
rclpy.spin(node)
rclpy.shutdown()
if __name__ == '__main__’:
      main()
```

https://roboticsbackend.com/write-minimal-ros2-cpp-node/

```
#include "rclcpp/rclcpp.hpp"
int main(int argc, char **argv)
{
     rclcpp::init(argc, argv);
     auto node = std::make_shared<rclcpp::Node>("my_node_name");
     rclcpp::spin(node);
     rclcpp::shutdown();
     return 0;
}
```

ROS one-shot learning

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. **ROS Services**
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files

ROS Services: Overview

- The **publish / subscribe model** is a **very flexible communication** paradigm, but its **many-to-many one-way transport is not appropriate for RPC request / reply interactions**, which are often required in a distributed system.

- **Request / reply** is done via a **ROS Service**, which is defined by a **pair of messages**:
  - One message for the **request** and one message for the **reply**.

- A providing ROS node offers a **service** under a **string name**, and a **client calls the service** by sending the **request message** and **awaiting the reply**.

- **Services** are defined using **`.srv`** files, which are compiled into source code by a ROS client library.

(Adapted from ROS Wiki: *wiki.ros.org/Services*)

ROS Services: Types

- Like topics, **services** have an associated service type that is the package resource name of the **`.srv`** file.

- As with other ROS filesystem-based types, the **service type** is the **package name + the name of the .srv file**.
  - For example: `my_srvs/srv/PolledImage.srv` has the service type `my_srvs/PolledImage`.

- In addition to the service type, **services are versioned by an MD5 sum** of the **`.srv`** file.
  - Nodes can only make service calls if both the service type and MD5 sum match.
  - This ensures that the client and server code were built from a consistent codebase.

(Adapted from ROS Wiki: *wiki.ros.org/Services*)

ROS Services

(Adapted from ROS ROS-I Basic Developers Training)

ROS Services: Details

- Each **Service** is made up of 2 components:
  - **Request** : sent by **client**, received by **server**
  - **Response** : generated by **server**, sent to **client**

- Call to service **blocks** in client
  - Code will wait for service call to complete
  - Separate connection for each service call

- Typical Uses:
  - Algorithms: *kinematics, perception*
  - Closed-Loop Commands: *move-to-position, open gripper*

(Adapted from ROS ROS-I Basic Developers Training)

ROS Services: Syntax

- **Service Definition**:
  - Defines **Request** and **Response** data types
    - Either/both data type(s) may be **empty**. Always receive “completed” handshake.
  - Auto-generates C++ Class files *(.h/.cpp)*, Python, etc.

(Adapted from ROS ROS-I Basic Developers Training)

ROS Services: Syntax

- **Service Server**:
  - Defines associated **Callback Function**
  - Advertises available service *(Name, Data Type)*

(Adapted from ROS ROS-I Basic Developers Training)

ROS Services: Syntax

- **Service Client**:
  - Connects to specific Service *(Name / Data Type)*
  - Fills in Request data
  - Calls Service

(Adapted from ROS ROS-I Basic Developers Training)

ROS one-shot learning

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. **ROS Actions**
9. ROS Parameters
10. ROS Launch Files

ROS Actions: Overview

- In any large ROS based system, there are cases when someone would like to **send a request** to a node to perform some task, and also **receive a reply to the request**. This can currently be achieved via **ROS services**.

- In some cases, however, **if the service takes a long time to execute**, the user might want the ability to **cancel the request during execution** or get **periodic feedback** about how the request is progressing.

- ROS Actions can create servers that execute **long-running goals** that can be **preempted**. It also provides a **client interface** in order to **send requests** to the server.

(Adapted from ROS Wiki: *wiki.ros.org/actionlib*)

ROS Actions: Overview

(Adapted from ROS ROS-I Basic Developers Training)

ROS Actions: Details

- Each **ROS Action** is made up of 3 components:
  - **Goal**, sent by client, received by server
  - **Result**, generated by server, sent to client
  - **Feedback**, generated by server

- **Non-blocking in client**
  - Can monitor feedback or cancel before completion

- Typical Uses:
  - **“Long” Tasks**: *Robot Motion, Path Planning*
  - **Complex Sequences**: *Pick Up Box, Sort Widgets*

(Adapted from ROS ROS-I Basic Developers Training)

ROS Actions: Syntax

- **Action Definition:**
  - Defines **Goal**, **Feedback** and **Result** data types
    - Any data type(s) may be **empty**. Always receive handshakes.
  - Auto-generates C++ Class files *(.h/.cpp)*, Python, etc.

(Adapted from ROS ROS-I Basic Developers Training)

ROS Actions: Syntax

- **Action Server:**
  - Defines **Execute Callback**
  - Periodically **Publish Feedback**
  - Advertises available action *(Name, Data Type).*

(Adapted from ROS ROS-I Basic Developers Training)

ROS Actions: Syntax

- **Action Client:**
  - Connects to specific **Action** *(Name / Data Type)*
  - Fills in **Goal** data
  - Initiate Action / Waits for Result

(Adapted from ROS ROS-I Basic Developers Training)

ROS Messages vs. ROS Services vs. ROS Actions

(Adapted from ROS ROS-I Basic Developers Training)

ROS one-shot learning

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. **ROS Parameters**
10. ROS Launch Files

ROS Parameters: Overview

(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: Overview

- Typically **configuration-type** values:
  - *Robot kinematics*
  - *Workcell description*
  - *Algorithm limits / tuning*

- Accessed through **the Parameter Server.**
  - Typically handled by **roscore**

(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: Ways to Setup

- Can set from:
  1. YAML Files
  2. Command Line
  3. Programs

(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: Datatypes

- **Native Types**
  - *int, real, boolean, string*
- **Lists (vectors)**
  - *can be mixed type: [1, str, 3.14159]*
  - *but typically of single type: [1.1, 1.2, 1.3]*
- **Dictionaries (structures)**
  - *translated to “folder” hierarchy on server*

(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: Namespaces

- Folder Hierarchy allows Separation:
  - *Separate nodes can co-exist, in different “namespaces”*
  - *relative vs. absolute name references*

(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: Namespaces

- **rosparam**
  - rosparam set <key> <value>
    - Set parameters
  - rosparam get <key>
    - Get parameters
  - rosparam delete <key>
    - Delete parameters
  - rosparam list
    - List all parameters currently set
  - rosparam load <filename> [<namespace>]
    - Load parameters from file

(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: Namespaces

- **rosparam**
  - rosparam set <key> <value>
    - Set parameters
  - rosparam get <key>
    - Get parameters
  - rosparam delete <key>
    - Delete parameters
  - rosparam list
    - List all parameters currently set
  - rosparam load <filename> [<namespace>]
    - Load parameters from file

(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: C++ API

- Accessed through **ros::NodeHandle object**
  - Also sets default **Namespace** for access
    - Relative Namespaces:
    - Fixed Namespaces:
    - Private Namespaces:

(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: C++ API

- NodeHandle object methods
  - **nh.hasParam(key)**
    - Returns true if parameter exists
  - **nh.getParam(key, &value)**
    - Gets value, returns T/F if exists.
  - **nh.param(key, &value, default)**
    - Get value (or default, if doesn't exist)
  - **nh.setParam(key, value)**
    - Sets value
  - **nh.deleteParam(key)**
    - Deletes parameter

(Adapted from ROS ROS-I Basic Developers Training)

ROS Parameters: C++ API

- Parameters must be read explicitly by nodes
  - *no on-the-fly updating*
  - *typically read only when node first started*

- ROS package **dynamic_reconfigure** can help
  - *nodes can register callbacks to trigger on change*
  - *outside the scope of this class, but useful*

(Adapted from ROS ROS-I Basic Developers Training)

ROS one-shot learning

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. **ROS Launch Files**

ROS Launch Files: Motivation

- ROS is a **Distributed System**:
  - Often **dozens** (if not hundreds) of **nodes**, plus **configuration data**
  - It would be *(VERY!)* painful to start each node “manually”

(Adapted from ROS ROS-I Basic Developers Training)

ROS Launch Files: Overview

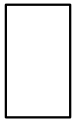
- **ROS Launch** is a **tool** for **easily launching multiple ROS nodes locally and remotely** via SSH, as well as **setting parameters on the Parameter Server**.

- It includes options to **automatically respawn processes** that have already died.

- **roslaunch** takes in one or more **XML configuration files** (with the .launch extension) that specify the **parameters** to set and **nodes** to launch, as well as the machines that they should be run on.

(Adapted from ROS Wiki: *wiki.ros.org/roslaunch*)

ROS Launch Files: Overview



(Adapted from ROS ROS-I Basic Developers Training)

ROS Launch Files: Overview

- **Launch files automate** system startup
- **XML** formatted script for running nodes and setting parameters
- Ability to **pull information from other packages**
- Will automatically start/stop **roscore**

(Adapted from ROS ROS-I Basic Developers Training)

ROS Launch Files: Notes

- Can launch ***other*** launch files
- **Executed in order**, without pause or wait
  - *Exception: Parameters set to parameter server before nodes are launched*
- Can accept **arguments**
- Can perform **simple IF-THEN** operations
- Supported **parameter types**:
  - *Bool, string, int, double, text file, binary file*

(Adapted from ROS ROS-I Basic Developers Training)

ROS Launch Files: Basic Syntax

- **<launch>** – Required outer tag
- **<rosparam>** or **<param>** – Set parameter values
  - Including load from file (YAML)
- **<node>** – Start running a new node
- **<include>** – Import another launch file

(Adapted from ROS ROS-I Basic Developers Training)

ROS Launch Files: More Syntax

- **<arg>** – Pass a value into a launch file
- **if=** or **unless=** – Conditional branching
  - Extremely limited. True/False only (no comparisons).
- **<group>** – group commands, for if/unless or namespace
- **<remap>** – rename topics/services/etc.

(Adapted from ROS ROS-I Basic Developers Training)

ROS1 Launch Files: Example

- motoman_sia20d_moveit_cfg
  - moveit_planning_exec.launch

(Adapted from ROS ROS-I Basic Developers Training)

ROS2 Launch Files:

https://docs.ros.org/en/foxy/Tutorials/Intermediate/Launch/Creating-Launch-Files.html

Copy and paste the complete code into the launch/turtlesim_mimic_launch.py file:

```
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='turtlesim',
            namespace='turtlesim1',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='turtlesim',
            namespace='turtlesim2',
            executable='turtlesim_node',
            name='sim'
        ),
        Node(
            package='turtlesim',
            executable='mimic',
            name='mimic',
            remappings=[
                ('/input/pose', '/turtlesim1/turtle1/pose'),
                ('/output/cmd_vel', '/turtlesim2/turtle1/cmd_vel'),
            ]
        )
    ])
```

ROS one-shot learning

1. What is ROS
2. ROS Distros, Installation, Programming Languages
3. ROS Architecture
4. ROS Packages
5. ROS Nodes
6. ROS Topics & Messages
7. ROS Services
8. ROS Actions
9. ROS Parameters
10. ROS Launch Files
11. ROS bags

☺ Demo's and tools

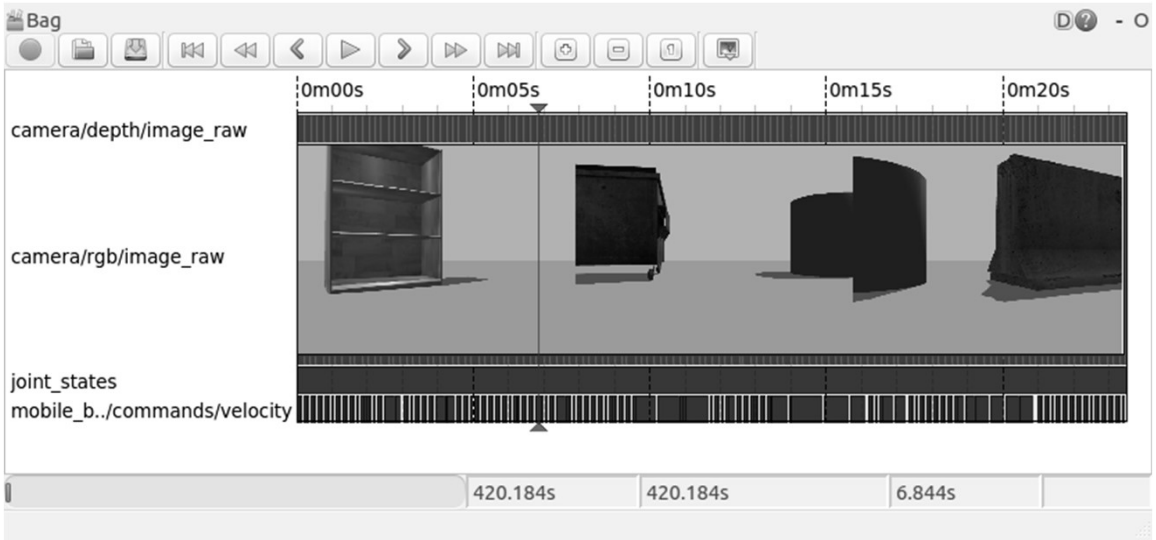
ROS Bags

1
3
5

ROS Bags

http://wiki.ros.org/Bags

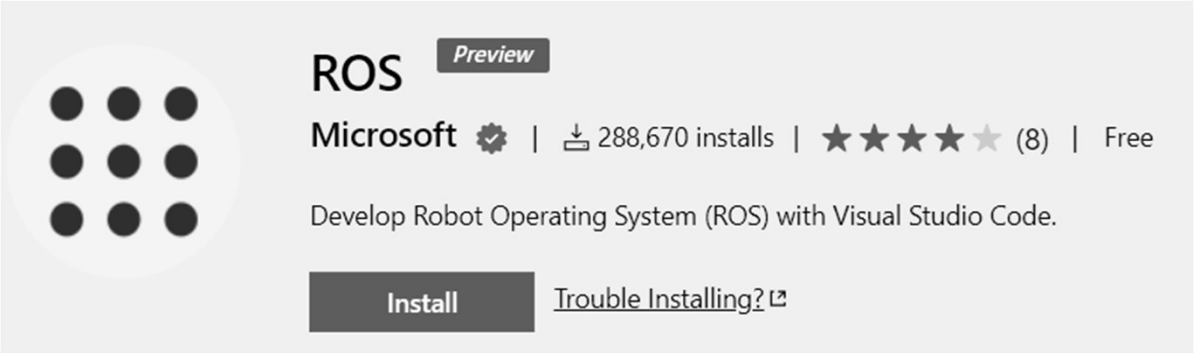
- Simulated clock
- No real time => easy debug
- Step by Step replay of problematic data
- Dataset friendly



•rosbag: unified console tool for recording, playback, and other operations.
•rqt_bag: graphical tool for visualizing bag file data.
•rostopic: the echo and list commands are compatible with bag files.
•Foxglove Studio: A browser and desktop tool to play, analyze, and visualize bag files.
•Webviz: A browser-based tool to look at data and ROS bag files.

Tools

My tool?
VSCode +
ROS extension



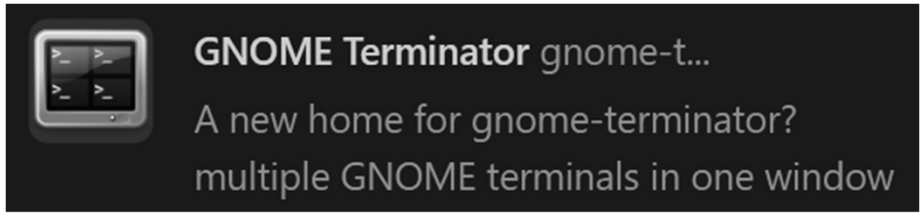
The Visual Studio Code Extension for ROS[^1] provides support for (ROS) development for ROS1 and ROS2 on Windows and Linux.

Features
- Automatic ROS environment configuration.
- Allows starting, stopping and viewing the ROS core status.
- Automatically create catkin_make or catkin build build tasks.
- Create catkin packages using catkin_create_pkg script or catkin create pkg.
- Run rosrun or roslaunch
- Resolve dependencies with rosdep shortcut
- Syntax highlighting for .msg, .urdf and other ROS files.
- Automatically add the ROS C++ include and Python import paths.
- Format C++ using the ROS clang-format style.
- Preview URDF and Xacro files.
- Debug a single ROS node (C++ or Python) by attaching to the process.
- Debug ROS nodes (C++ or Python) launched from a .launch file.

- *Careful:*
  - *multi-OS*
  - *multi "program" debugging…*
  - *communications*

https://marketplace.visualstudio.com/items?itemName=ms-iot.vscode-ros

Terminator



https://github.com/gnome-terminator/terminator

https://marketplace.visualstudio.com/items?itemName=ms-iot.vscode-ros


More?

More Information

- https://www.ros.org/
- http://docs.ros.org/
- http://docs.ros.org/en/rolling/
- https://rosindustrial.org/
- https://www.youtube.com/hashtag/rostutorials
- https://www.youtube.com/hashtag/ros
- https://www.youtube.com/hashtag/ros2

Demo...

ROS – Robotic Operating System

…a one-shot learning experience…

…covering ROS1 and a little of ROS2…

Rafael Arrais **&** **Armando Sousa**

asousa@fe.up.pt

& Luís Paulo Reis