ROS INTRODUCTION

Paloma de la Puente

INDUSTRIALES
ETSII | UPM

Content

- ROS Overview
- Ubuntu and basic terminal commands
- ROS2 Installation and Configuration
- ROS2 Computation Graph
- ROS2 command line tools
- ROS2 topics, services and actions

ROS OVERVIEW

What is ROS?

- Meta “operating system” for robots: middleware
- Collection of packages and software building tools
- Architecture for distributed* inter-process /inter-machine communication and configuration
- ROS is language-independent (C++, python, android, matlab, java...)
- ROS is open-source under permissive BSD licenses

What is ROS?

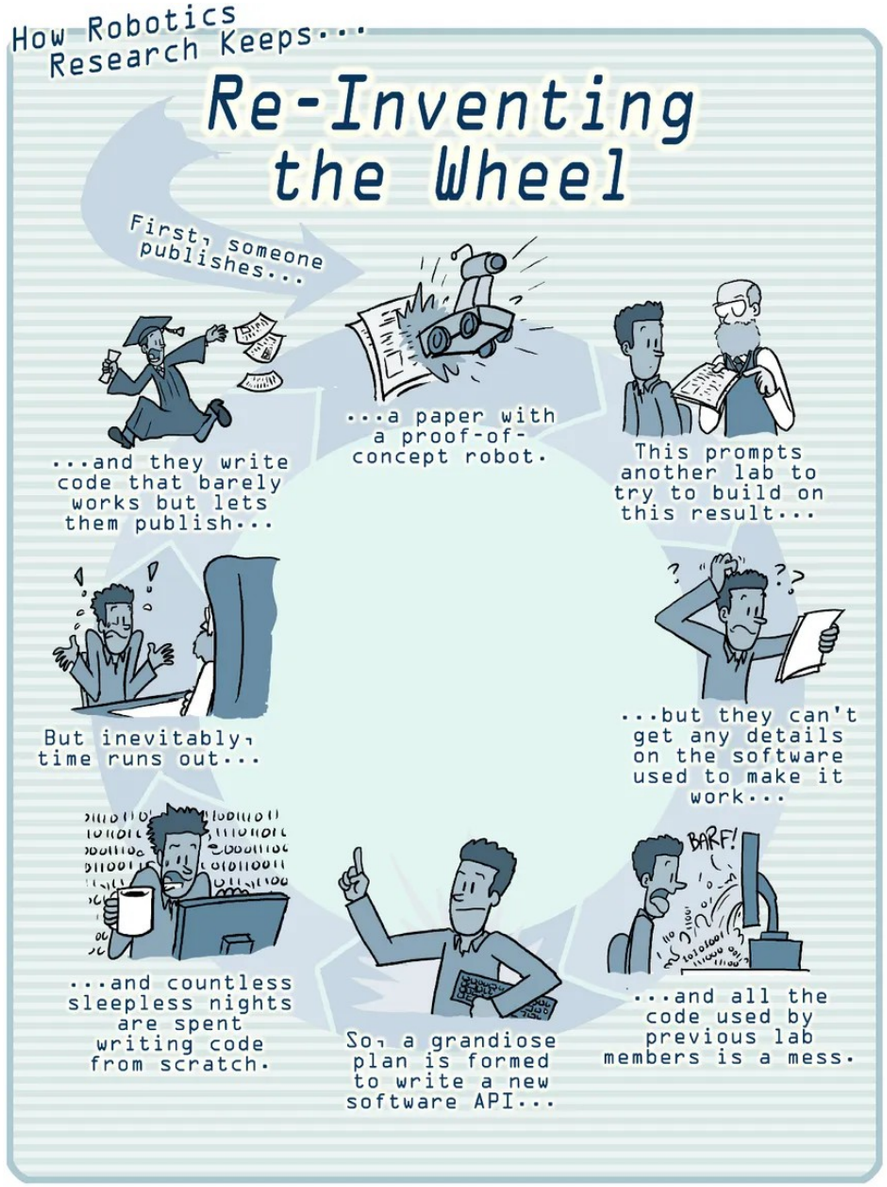
- Plumbing
- Tools
- Capabilities
- Ecosystem

What is ROS?

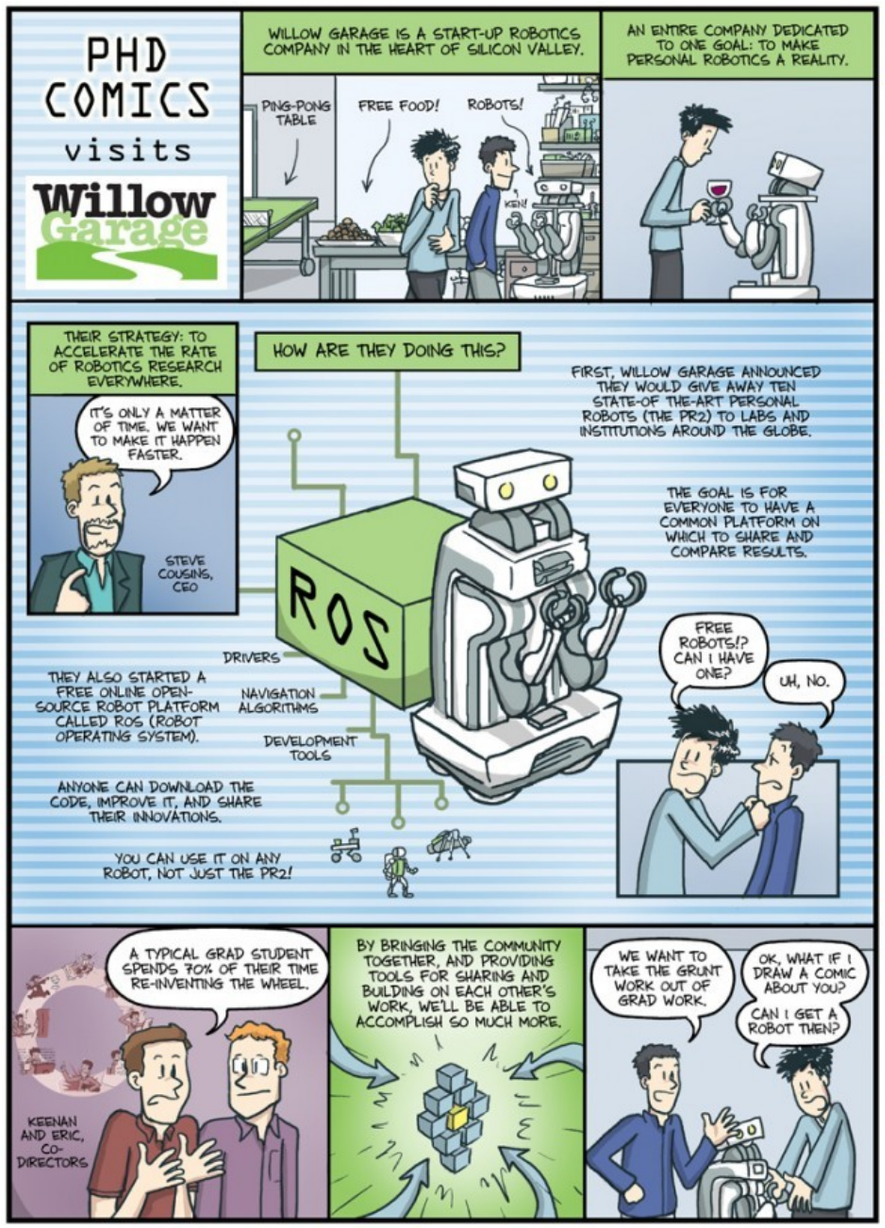
- “The Robot Operating System (ROS) is a set of software libraries and tools that help you build robot applications. From drivers to state-of-the-art algorithms, and with powerful developer tools, ROS has what you need for your next robotics project. And it's all open source.”

https://www.ros.org/

Why ROS?

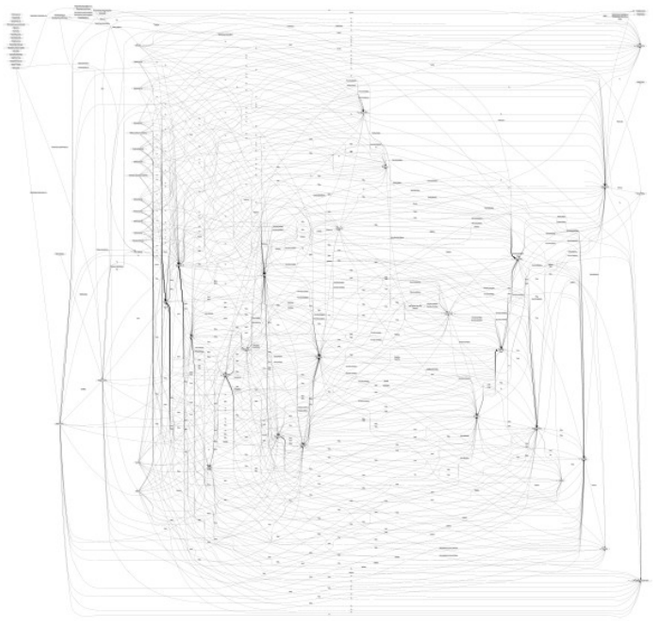

How Robotics Research Keeps...
Re-Inventing the Wheel
First, someone publishes...
...a paper with a proof-of-concept robot.
This prompts another lab to try to build on this result...
...but they can't get any details on the software used to make it work...
...and all the code used by previous lab members is a mess.
So, a grandiose plan is formed to write a new software API...
...and countless sleepless nights are spent writing code from scratch.
But inevitably, time runs out...
...and they write code that barely works but lets them publish...
BARF!

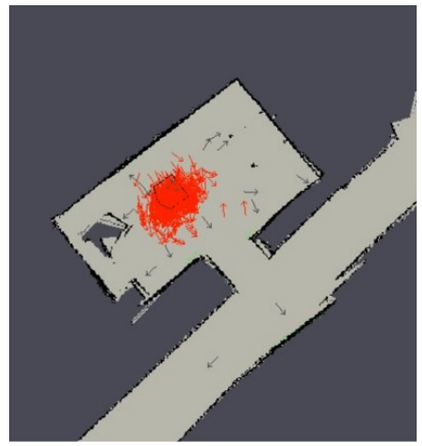
Why ROS?


PHD COMICS visits Willow Garage
WILLOW GARAGE IS A START-UP ROBOTICS COMPANY IN THE HEART OF SILICON VALLEY.
PING-PONG TABLE
FREE FOOD!
ROBOTS!
KEN
AN ENTIRE COMPANY DEDICATED TO ONE GOAL: TO MAKE PERSONAL ROBOTICS A REALITY.
THEIR STRATEGY: TO ACCELERATE THE RATE OF ROBOTICS RESEARCH EVERYWHERE.
IT'S ONLY A MATTER OF TIME. WE WANT TO MAKE IT HAPPEN FASTER.
STEVE COUSINS, CEO
HOW ARE THEY DOING THIS?
FIRST, WILLOW GARAGE ANNOUNCED THEY WOULD GIVE AWAY TEN STATE-OF-THE-ART PERSONAL ROBOTS (THE PR2) TO LABS AND INSTITUTIONS AROUND THE GLOBE.
THE GOAL IS FOR EVERYONE TO HAVE A COMMON PLATFORM ON WHICH TO SHARE AND COMPARE RESULTS.
ROS
DRIVERS
NAVIGATION ALGORITHMS
DEVELOPMENT TOOLS
THEY ALSO STARTED A FREE ONLINE OPEN-SOURCE ROBOT PLATFORM CALLED ROS (ROBOT OPERATING SYSTEM).
ANYONE CAN DOWNLOAD THE CODE, IMPROVE IT, AND SHARE THEIR INNOVATIONS.
YOU CAN USE IT ON ANY ROBOT, NOT JUST THE PR2!
FREE ROBOTS!? CAN I HAVE ONE?
UH, NO.
A TYPICAL GRAD STUDENT SPENDS 70% OF THEIR TIME RE-INVENTING THE WHEEL.
KEENAN AND ERIC, CO-DIRECTORS
BY BRINGING THE COMMUNITY TOGETHER, AND PROVIDING TOOLS FOR SHARING AND BUILDING ON EACH OTHER'S WORK, WE'LL BE ABLE TO ACCOMPLISH SO MUCH MORE.
WE WANT TO TAKE THE GRUNT WORK OUT OF GRAD WORK.
OK, WHAT IF I DRAW A COMIC ABOUT YOU?
CAN I GET A ROBOT THEN?

Why ROS?


AEROSTACK
A Software Framework for Aerial Robotic Systems
Gregale
Version 1.0.0
Gregale Distribution
vision4UAV
POLITÉCNICA
GPL
3
GOBIERNO DE ESPAÑA
MINISTERIO DE ECONOMÍA Y COMPETITIVIDAD





Why ROS?

- Robots running ROS

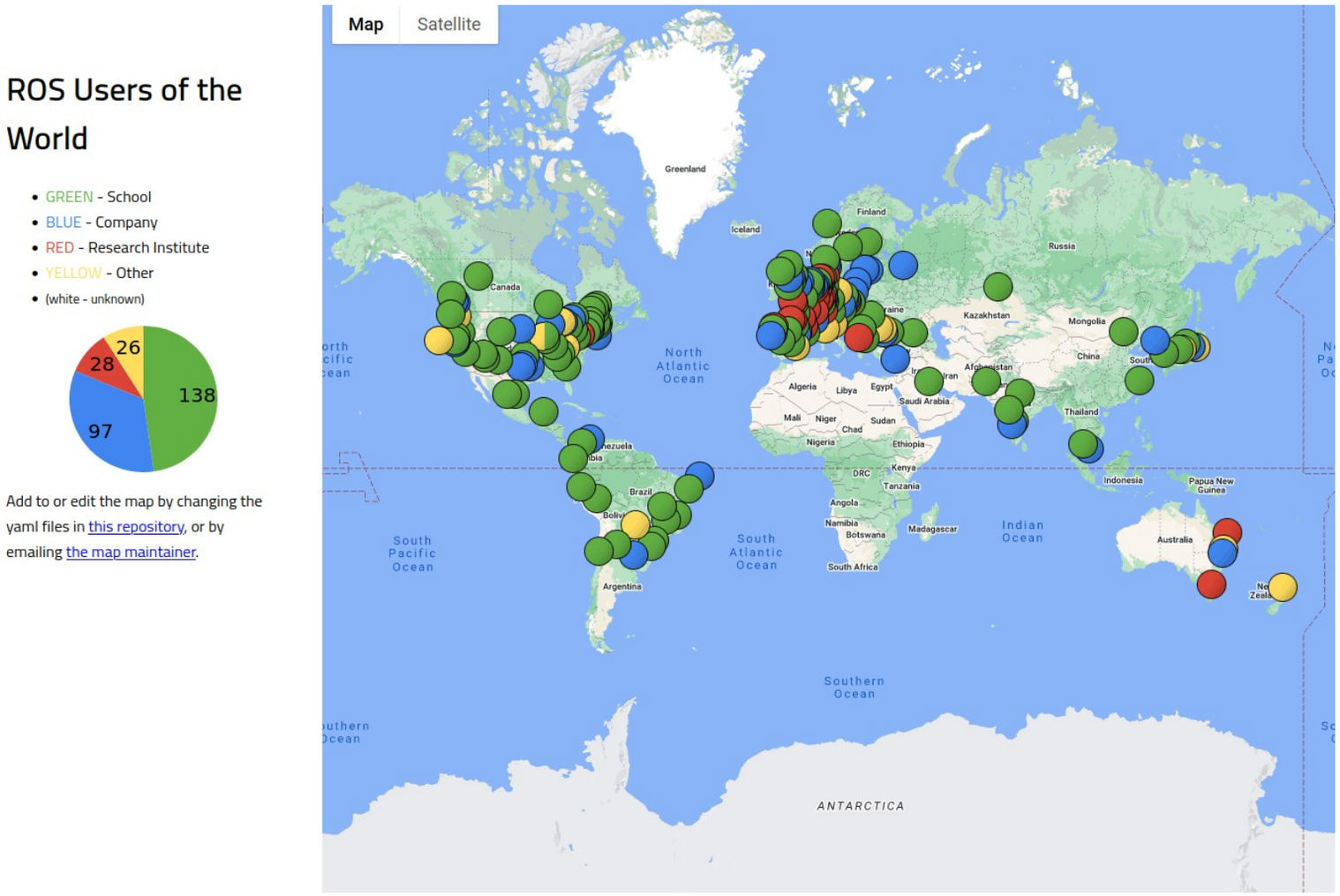
https://robots.ros.org/

Why ROS?

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

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

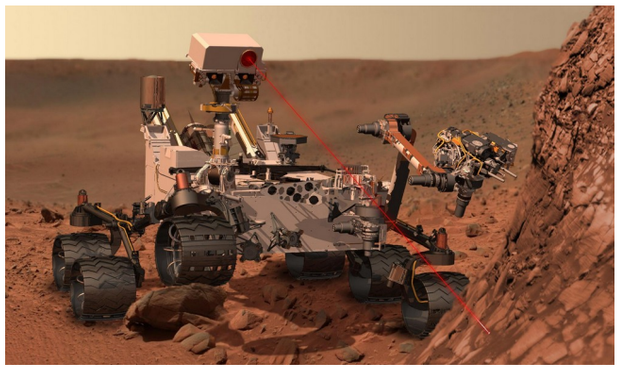
Why ROS?



http://metrorobots.com/rosmap.html

Why ROS?

- NASA, Blue Origin and industry partners are collaborating to develop the Space Robot Operating System (Space ROS)
- Flight-quality robotic and autonomous space systems
- A step toward a reusable standard that enables organizations to develop robots that seamlessly integrate and interoperate
- Request for Information (RFI) January 2022



Why ROS?

- SPACE ROS, RFI

System/Mission Attributes	Comments / Description
Robotic Cardinality	(Single Robot, Multi-Robot System, Human-Robot System, Spacecraft-Robot Interoperability, etc.)
Operational Environment	(LEO, GEO, Cis-Lunar, LLO, Lunar Surface, Planetary, etc.)
Degree and Mode of Perception	(LIDAR, Stereo Vision, Touch Sensor, Odometry, etc.)
Degree and Mode of Locomotion/Mobility	(Wheeled, legged, tracked, none, etc.)
Degree and Mode of Manipulation	(Arm, Multi-Armed, Other, etc.)
Degree and Mode of Human Interaction	(Remote-supervisory, Local-supervisory, Tele-operation, Human-in-Workspace, etc.)
Unique constraints and considerations	(Force control, onboard vs. on ground safety, etc.)
Greatest perceived robotic technical challenge	(Dexterous manipulation, Object perception, etc.)
Similar Reference Missions	(List examples of similar prior missions)
Expected Launch or Deployment date	(e.g., Planned and Projected Mission timeframes)
Software Certification Requirements	(DO 178C, ISO, etc.)
System/Mission Capabilities	Comments / Description
Degree of Autonomy	(Teleoperated, Shared Control, Supervisory control, Autonomous)
Control Topology	(Centralized, Distributed, Hierarchical, etc.)
System Communication	(What kinds of communications, Datalink, Communication protocols for each of the following: Inter-Robot, Inter-Process, Robot-Spacecraft, Robot-Earth, etc.)
Sensors and Actuator Drivers	(Sensors, Tools, Instruments, Locomotion, etc.)
Processor considerations	(Real-Time, Non-real-time, Safety-Critical, Mixed-criticality, redundancy, single/multi-string etc.)
Sensor data processors	(e.g., SoC, Arm cores, FPGAs, GPUs, Embedded, edge, cloud)
Degree and Mode of Planning	(High-level / deliberative planning, Scripting, Sequencing)
Logging and Reporting	(Health and Status, Data Formats, Telemetry)
Multi Agent Capabilities	Cooperative, Non-Cooperative, Centralized, Distributed
Multi Agent Interface Language	COAP, FIPA
Software/Framework Interoperability	cFS, Fprime, ROS/ROS2
Hardware Interfaces	Motor Controllers, Force Sensors, Accelerometers, LIDAR, RADAR, Power Control Systems
Algorithms	Motion Control, Sensor Fusion, FDIR, EO/IR Processing, Perception

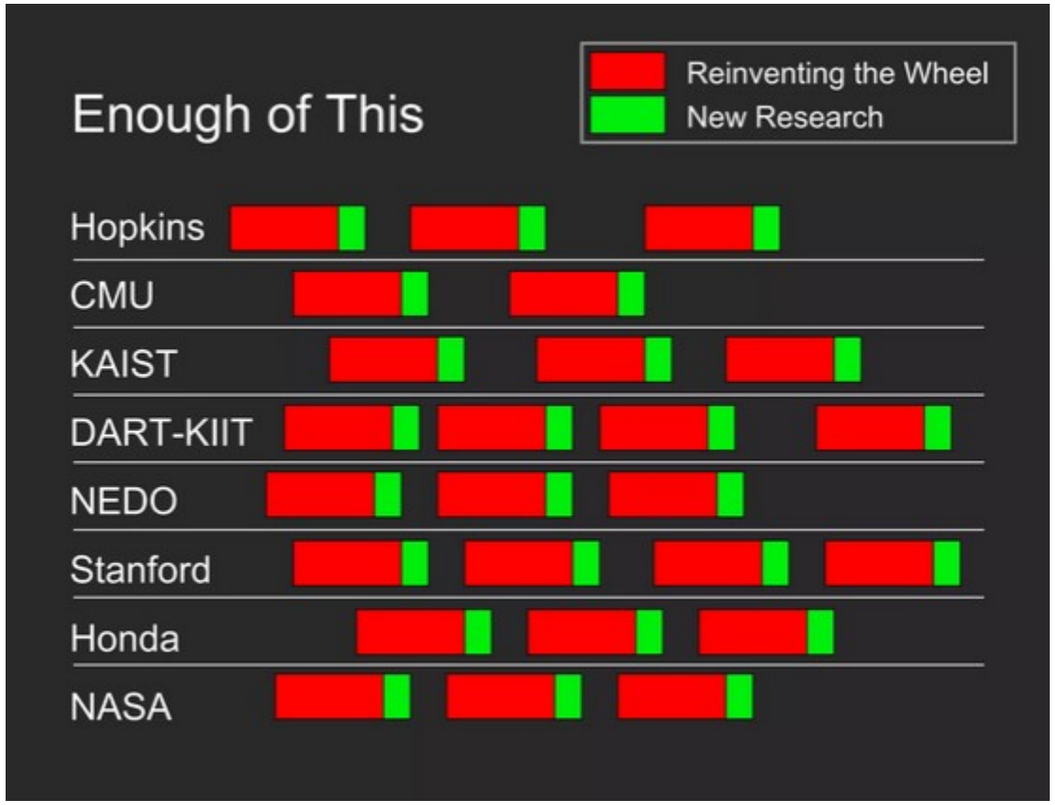
- A description of lessons learned using ROS in a flight or technology demo application
- A description on ROS flaws, shortcomings, or undesirable features
- ROS packages currently in use, and desired packages to be included in Space ROS

The origins of ROS

- Eric Berger and Keenan Wyrobek, PhD students at Standford, 2006
- Goal: $4 million, initially 50k
- Develop 10 PR robots, record cool videos, get community involved
- Scott Hassan supports the project to create the “Linux of Robotics” at Willow Garage
- Many others joined: K. Conley, B. Gerkey, M. Quigley…
- Great success story, more at https://spectrum.ieee.org/

ROS: An Open-Source Robot Operating System. By: Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ICRA workshop on open source software. Vol. 3. No. 3.2. 2009.

Why ROS?



By: Berger & Wyrobeck, 2006

Management of ROS

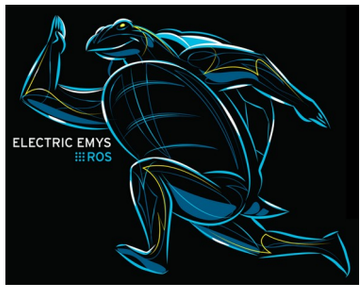
- Open Robotics (spin off from Willow Garage) in 2012: Open Source Robotics Foundation (OSRF)
  - *To support the development, distribution, and adoption of open source software for use in robotics research, education, and product development*
- Open Robotics branched out in 2016 from a strict non-profit to also take on some high-profile projects → Open Source Robotics Corporation (OSRC)
- OSRC office in Singapore opened in 2019
- OSRC acquired by Alphabet's Intrinsic in December 2022
  - *SRF continues as the independent nonprofit it's always been, with the same mission, now with some new faces and a clearer focus on governance, community engagement, and other stewardship activities.*

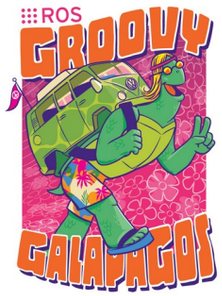
ROS versions

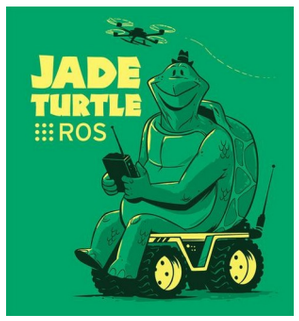
- There are many ROS distribution releases

ROS 1.0
http://ros.org/

Box Turtle







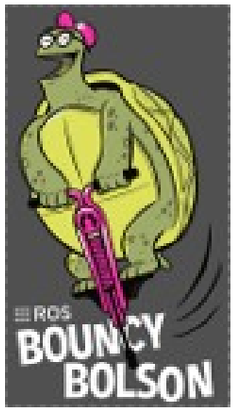
ROS limitations

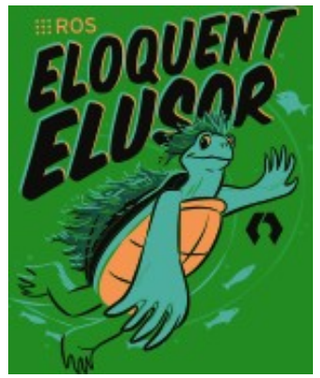
- Centralized in Master
- Own networking implementation
  - No discovery of resources
  - No QoS
- No multirobot
- No Real Time
- No multiplatform
- Important components are patches
- Only C++ and Python independent implementations
- Security

By: Francisco Martín Rico, 2023

ROS2 versions

- There are several ROS2 distribution releases



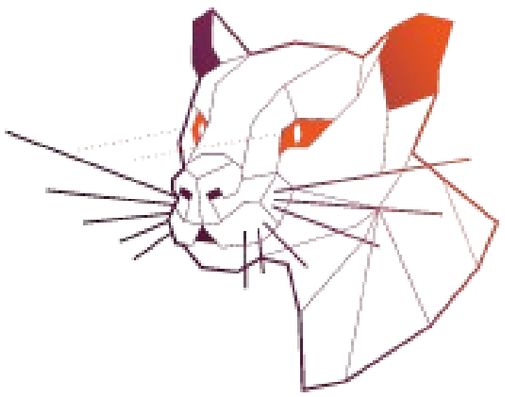


UBUNTU AND BASIC TERMINAL COMMANDS

Ubuntu

- Ubuntu is a Debian-based Linux operating system
- We will use Ubuntu 22.04 (Jammy Jellyfish)






Ubuntu

- Terminal (essential for developing ROS applications)

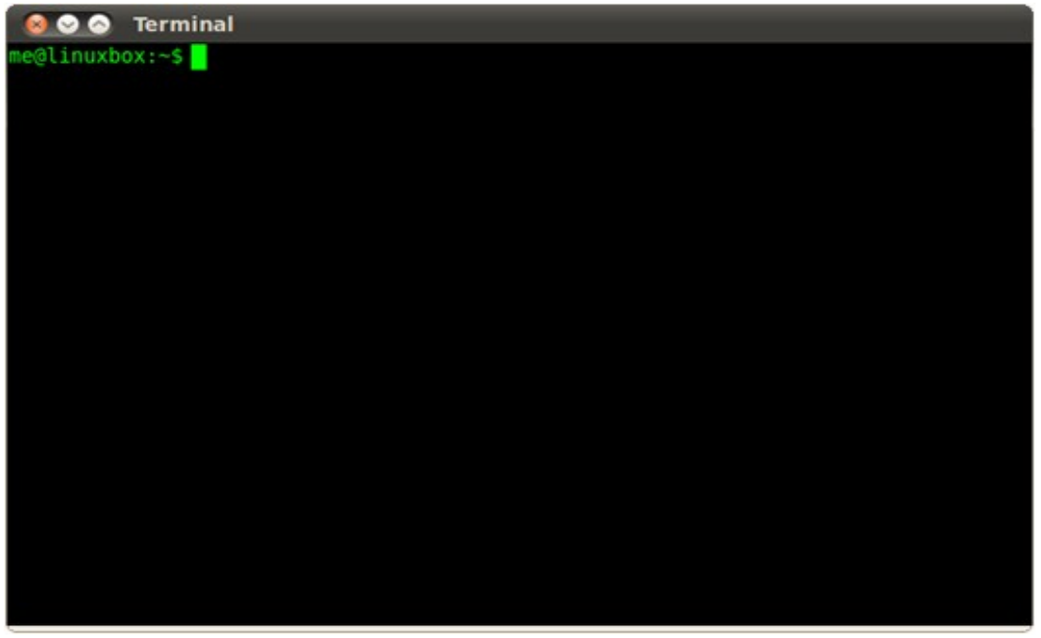
Command-line interpreter

Commands are powerful

New tab: **ctrl+shift+t**

New window: **ctrl+shift+n**

Use the tab key for auto-complete!



Ubuntu

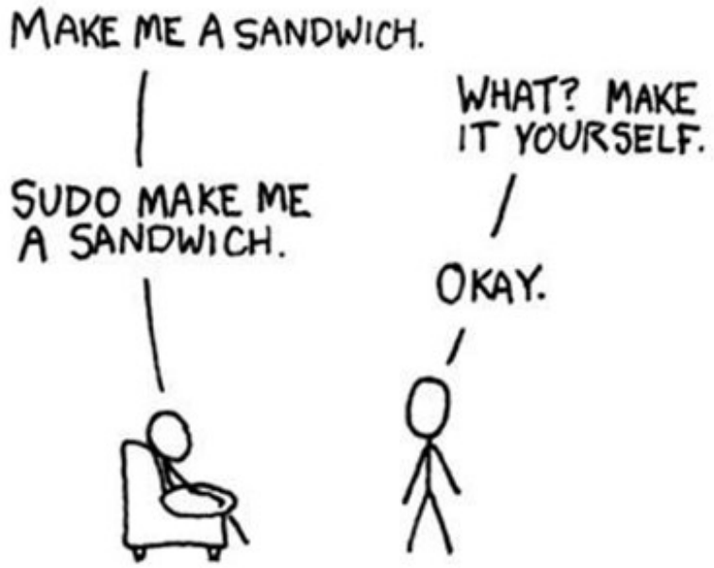
- List of basic commands

cd : to change directories
mkdir : to create a directory
rmdir : to remove a directory
pwd : to print the current directory
ls : to show the content of a directory
cat : to show the content of a file
nano: to edit the content of a file
rm : to remove a file
cp : to copy a file
mv : to move a file
apt-get install : to install programs
ps ax: to show all processes running on computer

Ubuntu

- List of basic commands

whatis : to show a single line description about a command
man : to show the complete manual page for a command
sudo :
sudo -> **s**uper **u**ser **do**
to use commands with superuser permissions



Ubuntu

- Example

1. Go to the user directory (cd)
2. Discover where you are (pwd)
3. Create a directory with the name “ros2_ws” (mkdir ros2_ws)
4. Go into the new directory (cd ros2_ws)
5. Print a text into a new file (echo “example_text” > file.txt)
6. See the content of the directory with details (ls -l)
7. Show the content of the file (cat file.txt)
8. Remove the file (rm file.txt)
9. See the content of the directory again (ls)
10. Realize you didn’t want to delete
11. Cry like a baby because there isn’t an undo option when you use the rm command

Ubuntu

- To edit your code:
  - Nano or vi command line editor
  - Text Editor installed by default in Ubuntu
  - Advanced IDES
    - Eclipse
    - Qt Creator → Qt Creator IDE Plug-in for ROS
    - Codeblocks

ROS2 INSTALLATION AND CONFIGURATION

RO2 Installation

https://docs.ros.org/en/humble/Installation/Ubuntu-Install-Debians.html

- Select Desktop Install option
- In a terminal, execute:

  $ gedit ~/.bashrc

- When the file opens, copy the next lines there:

source /opt/ros/humble/setup.bash

export ROS_DOMAIN_ID=1 #choose a numer of your choice, between 1 and 101

#export ROS_LOCALHOST_ONLY=1

ROS2 Configuration

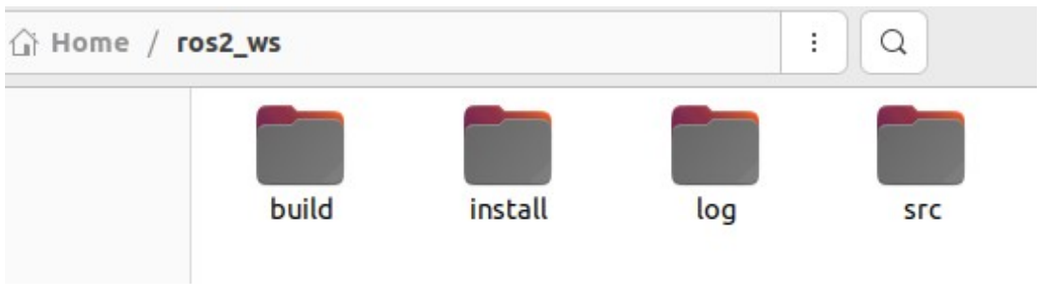
- Creating a ROS workspace

$ mkdir -p ~/ros2_ws/src

$ colcon build –symlink-install

Add the next line to you .bashrc file:

source ~/ros2_ws/install/local_setup.bash



https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html

ROS2 Configuration

- The core ROS 2 workspace is called the underlay
- Subsequent local workspaces are called overlays
- The underlay must contain the dependencies of all the packages in the overlay
- Packages in the overlay will override packages in the underlay.

https://docs.ros.org/en/humble/Tutorials/Beginner-Client-Libraries/Creating-A-Workspace/Creating-A-Workspace.html

ROS2 Configuration

- File system

```
workspace_folder/
    src/
      cpp_package_1/
          CMakeLists.txt
          include/cpp_package_1/
          package.xml
          src/

      py_package_1/
          package.xml
          resource/py_package_1
          setup.cfg
          setup.py
          py_package_1/
      ...
      cpp_package_n/
          CMakeLists.txt
          include/cpp_package_n/
          package.xml
          src/
```

Introduction

ROS2 GRAPH

ROS2 Configuration

- The **ROS2 graph** is a network of ROS 2 elements processing data together at the same time. It encompasses all executables and the connections between them.
- **Nodes**: a node is a participant in the ROS 2 graph, which uses a client library to communicate with other nodes. Nodes will only establish connections with other nodes if they have compatible Quality of Service settings.
- **Messages**: Messages are a way for a ROS 2 node to send data on the network to other ROS nodes, with no response expected.
- **Topics**: Nodes can publish messages to a topic as well as subscribe to a topic
- **Services**: Services are a request/response communication, where the client (requester) is waiting for the server (responder) to make a short computation and return a result.
- **Actions**: Actions are used for long-running goals that can be preempted.

ROS2 COMMAND LINE TOOLS

ROS2 Configuration

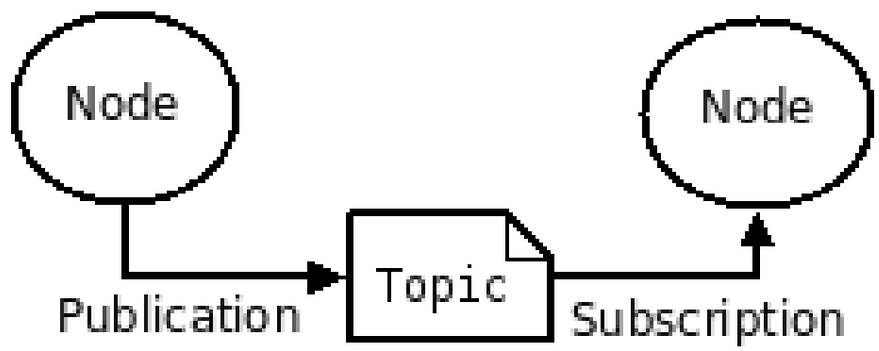
ros2 run: run a package specific executable
ros2 node -h: options to get information about the nodes that are running
ros2 node list: show a list of active nodes
ros2 topic -h: options to get information about the topics that are running
ros2 topic list: show a list of active topics
ros2 topic echo: show the messages published to a topic
ros2 topic pub: publish a message to a topic
ros2 service -h: commands for ROS’s client/service framework
ros2 launch: start nodes defined in a launch file

ROS TOPICS, SERVICES AND ACTIONS

ROS Topics

- Nodes can publish or subscribe to a topic to send or receive messages
- Topics are conceived for unidirectional, streaming communication
- Topics must have a name and associated type of message
- Messages are described and defined in .msg files composed of two parts: fields and constants.
- Find existing interfaces at: https://github.com/ros2/common_interfaces

$ ros2 interface list -m



https://docs.ros.org/en/humble/Concepts/Basic/About-Topics.html

.msg examples

example_interfaces/msg/String Message

File: example_interfaces/msg/String.msg

Raw Message Definition

```
# This is an example message of using a primitive datatype, string.
# If you want to test with this that's fine, but if you are deploying
# it into a system you should create a semantically meaningful message type.
# If you want to embed it in another message, use the primitive data type instead.
string data
```

Compact Message Definition

```
string data
```

File: geometry_msgs/msg/Point.msg

Raw Message Definition

```
# This contains the position of a point in free space
float64 x
float64 y
float64 z
```

Compact Message Definition

```
double x
double y
double z
```

File: geometry_msgs/msg/Pose.msg

Raw Message Definition

```
# A representation of pose in free space, composed of position and orientation.

Point position
Quaternion orientation
```

Compact Message Definition

```
geometry_msgs/msg/Point position
geometry_msgs/msg/Quaternion orientation
```

File: geometry_msgs/msg/PoseWithCovariance.msg

Raw Message Definition

```
# This represents a pose in free space with uncertainty.

Pose pose
# Row-major representation of the 6x6 covariance matrix
# The orientation parameters use a fixed-axis representation.
# In order, the parameters are:
# (x, y, z, rotation about X axis, rotation about Y axis, rotation about Z axis)
float64[36] covariance
```

Compact Message Definition

```
geometry_msgs/msg/Pose pose
double[36] covariance
```

Examples

$ros2 run examples_rclcpp_minimal_publisher publisher_member_function

$ros2 node info /minimal_publisher

```
/minimal_publisher
  Subscribers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
    /topic: std_msgs/msg/String
  Service Servers:
    /minimal_publisher/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /minimal_publisher/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /minimal_publisher/get_parameters: rcl_interfaces/srv/GetParameters
    /minimal_publisher/list_parameters: rcl_interfaces/srv/ListParameters
    /minimal_publisher/set_parameters: rcl_interfaces/srv/SetParameters
    /minimal_publisher/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:

  Action Servers:

  Action Clients:
```

Examples

$ros2 run examples_rclcpp_minimal_subscriber subscriber_member_function

$ros2 node info /minimal_subscriber

```
/minimal_publisher
  Subscribers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
    /topic: std_msgs/msg/String
  Service Servers:
    /minimal_publisher/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /minimal_publisher/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /minimal_publisher/get_parameters: rcl_interfaces/srv/GetParameters
    /minimal_publisher/list_parameters: rcl_interfaces/srv/ListParameters
    /minimal_publisher/set_parameters: rcl_interfaces/srv/SetParameters
    /minimal_publisher/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:

  Action Servers:

  Action Clients:
```

Exercise

1. Check which nodes are running and available topics

   $ ros2 node list

   $ ros2 topic list

2. Kill the subscriber node. Open a new terminal and type:

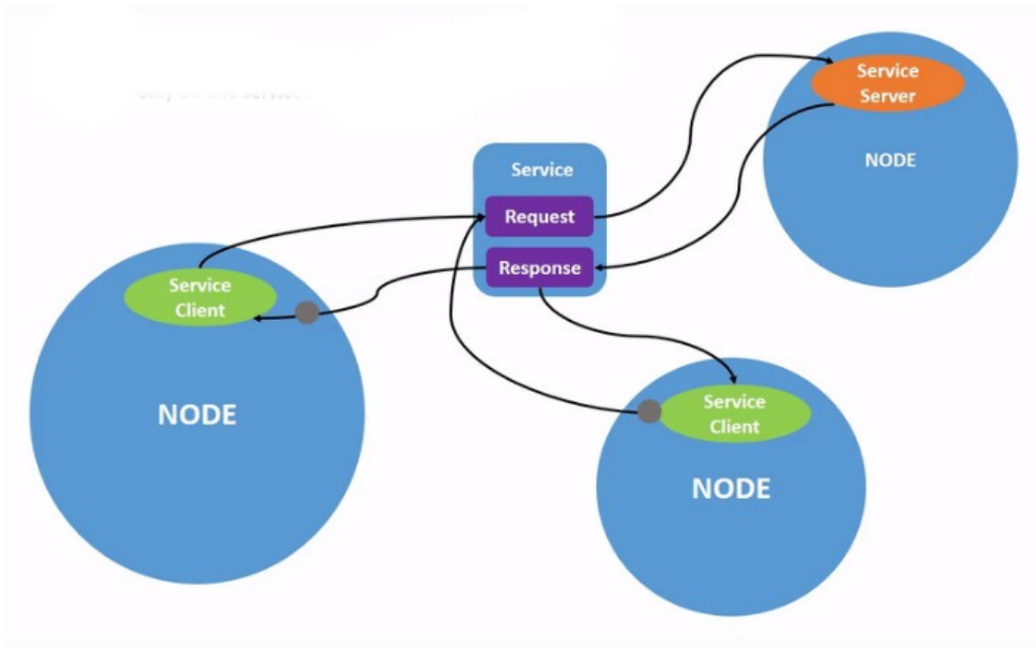
   $ ros2 topic echo /topic

3. Run the subscriber again. Kill the publisher. Do not close the echo terminal. Publish a text message of your own:

   $ ros2 topic pub -1 /topic std_msgs/msg/String "{data: 'Hi, it is me'}"

ROS services

- The publish/subscribe model is not appropriate for RPC request/reply interactions.
- Request/reply is done via services, which are defined by a pair of messages: one for the request and one for the reply
- Services are defined using srv files, which are compiled into source code by a ROS client library.



https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Services/Understanding-ROS2-Services.html

.srv examples

std_srvs/srv/Empty Service

File: std_srvs/srv/Empty.msg

Raw Message Definition

```
---
```

Compact Message Definition

```
---
```

example_interfaces/srv/SetBool Service

File: example_interfaces/srv/SetBool.msg

Raw Message Definition

```
# This is an example of a service to set a boolean value.
# This can be used for testing but a semantically meaningful
# one should be created to be built upon.

#bool data # e.g. for hardware enabling / disabling

---
#bool success # indicate successful run of triggered service
#string message # informational, e.g. for error messages
```

Compact Message Definition

```
boolean success
string message
---
boolean data
```

std_srvs/srv/Trigger Service

File: std_srvs/srv/Trigger.msg

Raw Message Definition

```
---
#bool success # indicate successful run of triggered service
#string message # informational, e.g. for error messages
```

Compact Message Definition

```
boolean success
string message
---
```

Exercise

1. Check the available service commands:

   $ ros2 service -h

2. Check the available services:

   $ ros2 service list

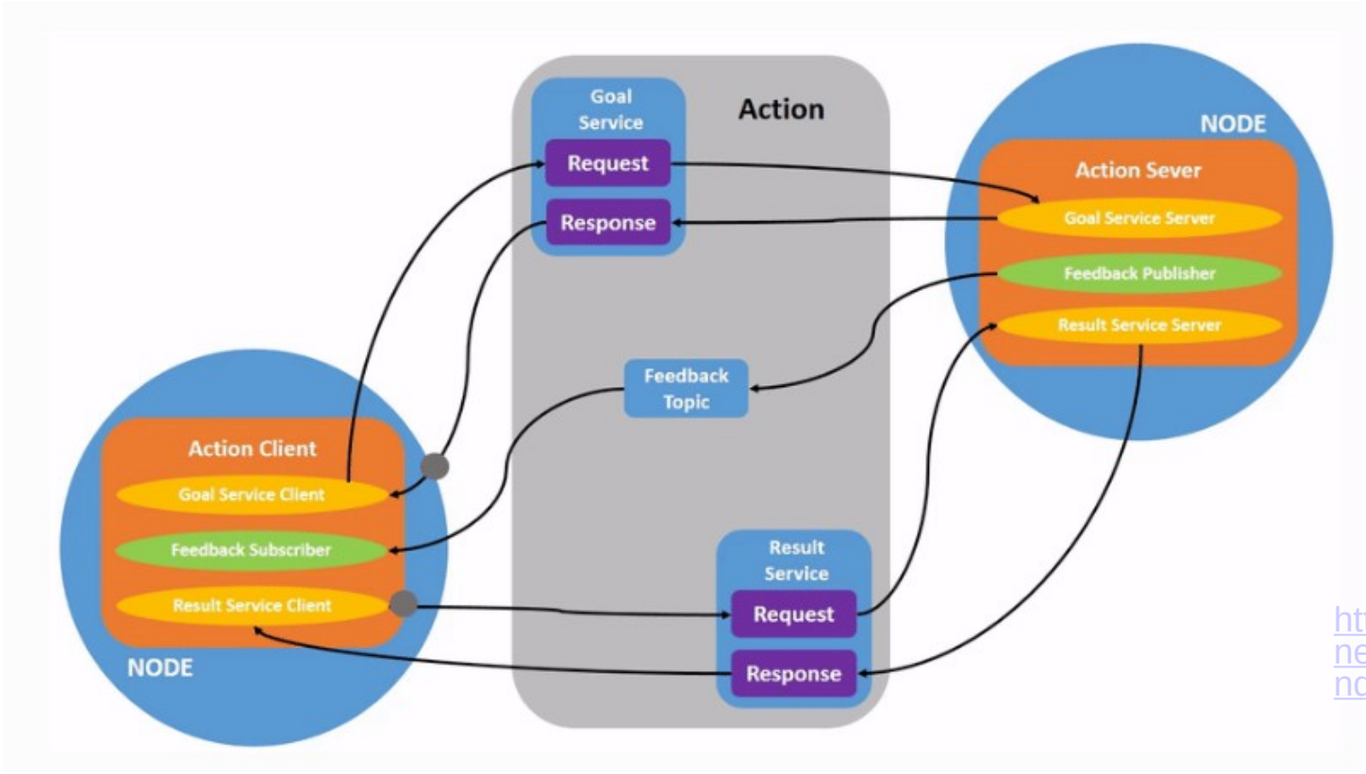
3. Call one of them:

   $ ros2 service type /minimal_subscriber/get_parameters

   $ ros2 service call /minimal_subscriber/get_parameters rcl_interfaces/srv/GetParameters

ROS actions

- Actions are intended for long running tasks
- They consist of three parts: a goal, feedback, and a result
- Actions use a client-server model
- Actions are defined using .action files, which are compiled into source code by a ROS client library

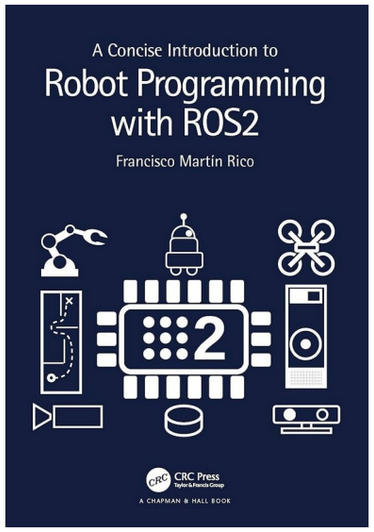


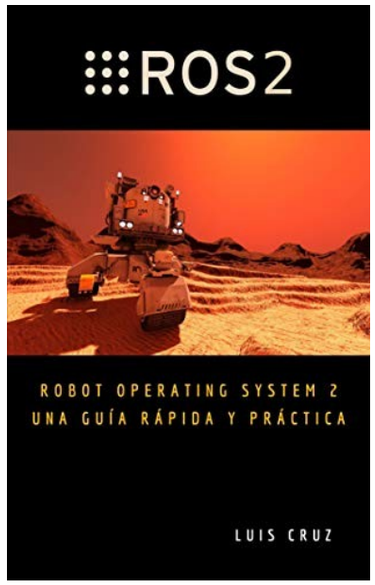
https://docs.ros.org/en/humble/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Actions/Understanding-ROS2-Actions.html

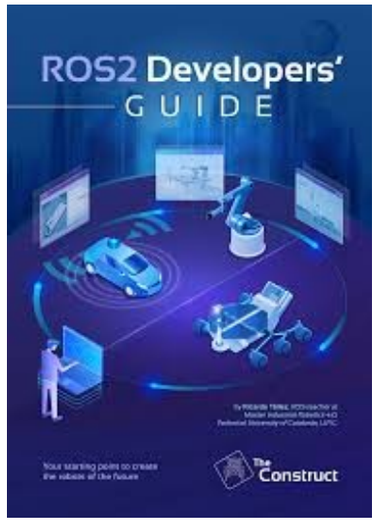
Resources

- Ubuntu
  - http://askubuntu.com
- ROS
  - http://www.ros.org
    - Tutorials: https://docs.ros.org/en/humble/Tutorials.html
    - ROS Answers → migrated to https://robotics.stackexchange.com/
- https://github.com/ubuntu-robotics/ros2_cheats_sheet
- https://docs.ros.org/en/rolling/The-ROS2-Project/Contributing.html

Resources


A Concise Introduction to
Robot Programming with ROS2
Francisco Martín Rico
CRC Press
A CHAPMAN & HALL BOOK


ROS2
ROBOT OPERATING SYSTEM 2
UNA GUÍA RÁPIDA Y PRÁCTICA
LUIS CRUZ


ROS2 Developers'
GUIDE
The Construct


EDOUARD RENARD
ROS2
FOR BEGINNERS
MANNING

ROS INTRODUCTION

End of lesson