University of Idaho
Department of Computer Science
Coeur d'Alene

A Vandalized Introduction to ROS2

written by barbarians for barbarians!

Garrett Wells

October 26, 2023

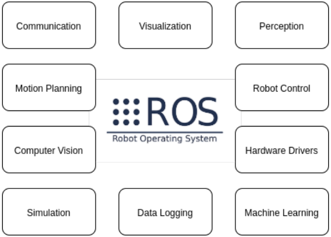
Table of Contents

- What is ROS2?
  - Abstracting Machines as Data
  - The ROS2 Stack
- ROS2 Concepts
  - Topics
    - Topic Illustration
  - Actions
  - Services
  - Client Libraries
  - Workspaces
- Installing ROS2!

What is ROS2?

From the Horse's Mouth

The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. From drivers and state-of-the-art algorithms to powerful developer tools, ROS has the open source tools you need for your next robotics project.



What is ROS2?

From the Horse's Mouth

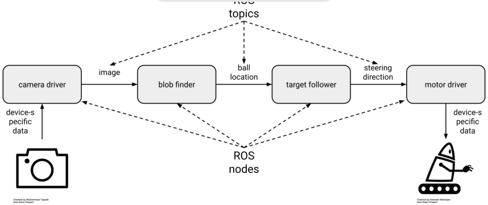
The Robot Operating System (ROS) is a set of software libraries and tools for building robot applications. From drivers and state-of-the-art algorithms to powerful developer tools, ROS has the open source tools you need for your next robotics project.

- Open source software! Whoopee!
- Tries to abstract robots and sensors as **producers** and **consumers** of data
- Solves *almost* as many problems as it creates

Abstracting Machines as Streams of Data

what we see

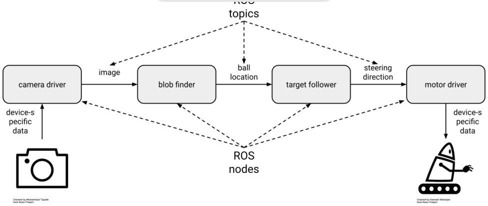
what ros sees



Abstracting Machines as Streams of Data

what we see

what ros sees



Created by Muhammad Tajudin from Noun Project

Created by Kamesh Maharjan from Noun Project

What is a Node?

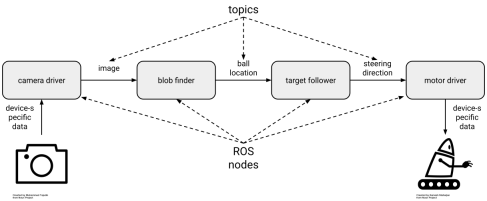
A node is a participant in the ROS 2 graph, which uses a client library to communicate with other nodes. Nodes can communicate with other nodes within the same process, in a different process, or on a different machine. Nodes are typically the unit of computation in a ROS graph; each node should do one logical thing.

ROS Concepts Documentation Link

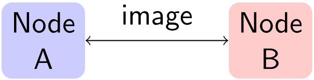
Abstracting Machines as Streams of Data

what we see

what ros sees



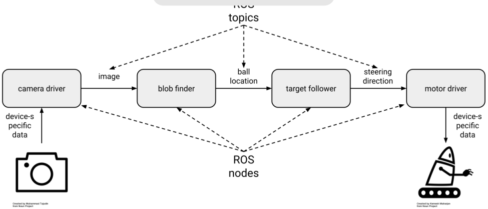
Nodes in Example: camera driver, blob finder, target follower, motor driver



Abstracting Machines as Streams of Data

what we see

what ros sees

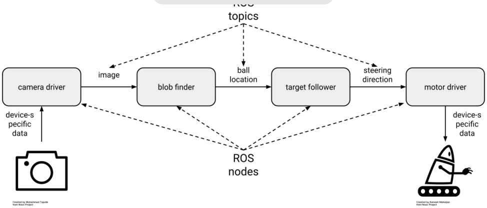


- **camera driver :** **producer** interfaces with hardware to capture image, then publishes to topic (**image**)

Abstracting Machines as Streams of Data

what we see

what ros sees



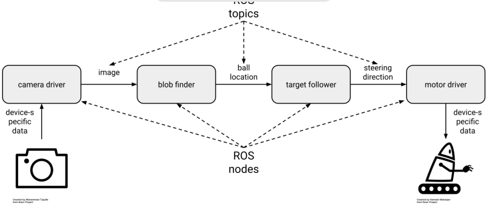
- **camera driver :** **producer** interfaces with hardware to capture image, then publishes to topic (**image**)
- **motor driver :** **consumer** consumes data from topic (**steering direction**) and then interfaces with motors to move forward

University *of* Idaho
Department of Computer Science
Coeur d'Alene

Abstracting Machines as Streams of Data

what we see

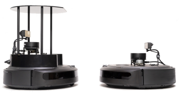
what ros sees



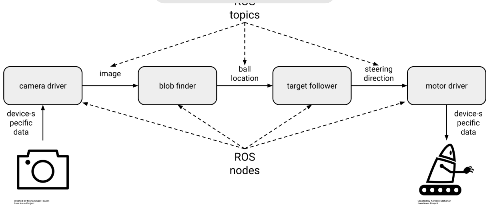
- ROS2 refers to nodes that are **producers** as **publishers**
- can publish as needed or at a frequency, ex. 10Hz
- topics are identified with a path, ex. */robot/odom*

Abstracting Machines as Streams of Data

what we see

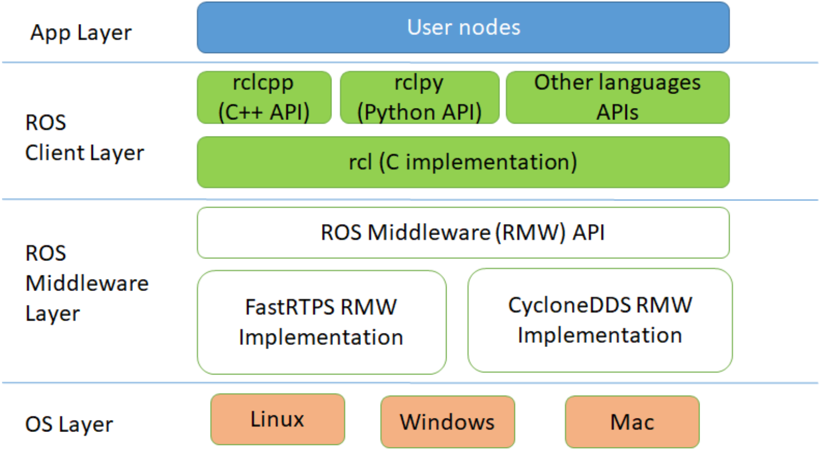


what ros sees

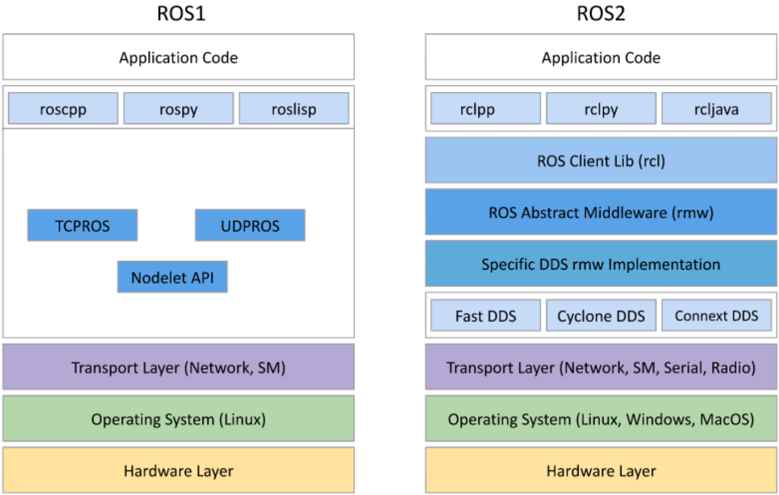


- ROS2 refers to nodes that are **consumers** as **subscribers**
- subscribers use path to subscribe to a topic

The ROS2 Stack



The ROS2 Stack



ROS2 Concepts

Topics, Actions, Servers, Workspaces, and Client Libraries

Topics

- topics created by publishing/subscribing nodes
- available to all devices on network
- topics are forward slash separated paths
- messages are strongly typed structs
- messages are anonymous by default

ROS2 Topics Example Link

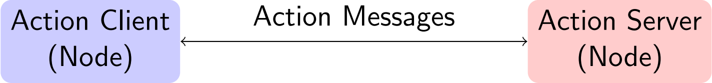
Topic Illustration: Using A Standard ROS2 Topic

So what does a topic message consist of?

topic: `/odom [nav_msgs/msg/Odometry]`
topic message:

```
# This represents an estimate of a position and velocity in free space.
# The pose in this message should be specified in the coordinate frame given by header.frame_id.
# The twist in this message should be specified in the coordinate frame given by the child_frame_id
Header header
string child_frame_id
geometry_msgs/PoseWithCovariance pose
geometry_msgs/TwistWithCovariance twist
```

Actions



Action Client (Node)

Action Messages

Action Server (Node)

Action

Consists of three parts:

- Goal: *a message containing what you wish to be done*
- Feedback: *a message from the action server*
- Result: *the success/failure message from server*

Check out the example tutorial and documentation:

ROS2 Actions Documentation Link

Services are call and response versions of topics

Services

While topics allow nodes to subscribe to data streams and get continual updates, services **only provide data when they are specifically called by a client**.

Used to request that computation heavy tasks be completed by another node. (Google, tell me the 10th digit of Pi!)

- consist of a *request* and a *response*

Services GIF Link

Client Libraries provide access to the ROS2 API including:

- Node functions
- topic pub/sub functions
- actions
- services

available for C++, Python, and more!

ROS Client Libraries Link

RCL Example

```
node = rclpy.create_node('minimal_publisher')
publisher = node.create_publisher(String, 'topic', 10)
```

GitHub RCL Examples Link

RCL Inheritance

```
import rclpy
from rclpy.node import Node

from std_msgs.msg import String


class MinimalPublisher(Node):

    def __init__(self):
        super().__init__(’minimal_publisher’)
        self.publisher_ = self.create_publisher(String, ’topic’, 10)
        timer_period = 0.5  # seconds
        self.timer = self.create_timer(timer_period, self.timer_callback)
        self.i = 0
```

Example Source Code

Workspaces

Workspaces

A workspace is a directory containing ROS 2 packages. Before using ROS 2, it's necessary to **source** your ROS 2 installation workspace in the terminal you plan to work in. This makes ROS 2's packages available for you to use in that terminal.

Package

A package is an organizational unit (directory) for your ROS 2 code. Packages may contain metadata and scripts that help run and build the package.

Example Workspace Structure: Pre-Build

```
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/
```

Workspaces Expanded

Workspaces:

- start as a directory with a `src/` subdirectory
- use `ros2` CLI to generate package metadata files `src/<pkg_name>`
- building generates more files, which are placed in the root of the workspace directory

Workspaces Expanded

Package Dependencies

Packages may have dependencies on other packages in the workspace/ROS2 install! How will packages in `ros_ws/src/<pkg_name>` resolve dependencies??

Workspaces Expanded

Sourcing!

Sourcing refers to the `source` command in Linux. Sourcing causes the system shell (ex. bash) to execute the contents of a file, *making the contents available to the system.*

Example

After building a package, `local_setup.sh` is generated and placed in `ros_ws/install`, running `source install/local_setup.sh` allows ROS2 code to be run inside the workspace.

```
$ source local_setup.sh
```

Installing ROS2

So. . . what next?

Guides and Resources

- ROS2 Humble Hawksbill Installation Guide Link
- ROS2 Humble Hawksbill Documentation Link
- ROS2 REPs Link
- ROS Community Forums Link

Tutorials

- ROS2 CLI Beginner Tutorial
- ROS2 Beginner Client Libraries