Introduction to ROS2

IEEE RAS - Slides adapted from
https://frezza.pages.centralesupelec.fr/st5-drones/

September 22, 2025

Objectifs et organisation

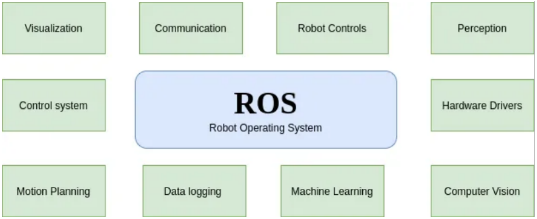
Cours objectives

- Understanding the key concepts of ROS2
    - Nodes, messages, services, . . .
- Getting to know the tools and ecosystem around ROS2
    - Colcon for building project
    - Introspecting and visualization tools
    - Simulation tools (Gazebo)

ROS is a middleware

Robot Operating System (ROS)

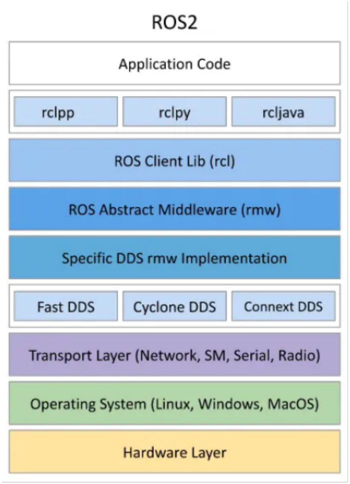
- Open-source middleware (not an OS)
- Development environment
  - Visualization and introspection tools
- Communication library and tools
- Packaging system
  - *colcon* command
- Plenty of existing modules
- Active community



Images from medium.com

ROS abstract layers

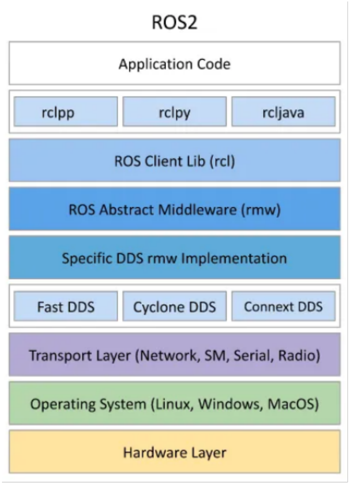
- DDS (Data Distribution Service)
  - Standard that enable data exchanges using a publish–subscribe pattern.
  - Middleware providing high-performance, scalable, and secure way for nodes to exchange data and communicate with each other.
  - Used as the communication layer for ROS2.
- rmw (ROS MiddleWare)
  - Middleware providing the underlying communication infrastructure for ROS2 using DDS
  - Abstracts the complexity of DDS
  - → developers do not need to worry about the details of how DDS works.


ROS2
Application Code
rclpp
rclpy
rcljava
ROS Client Lib (rcl)
ROS Abstract Middleware (rmw)
Specific DDS rmw Implementation
Fast DDS
Cyclone DDS
Connext DDS
Transport Layer (Network, SM, Serial, Radio)
Operating System (Linux, Windows, MacOS)
Hardware Layer

Images from medium.com

ROS abstract layers 2

- rcl (ROS Client Library)
    - Middleware providing a high-level interface for building and running robot applications using rmw.
    - Abstract the complexity of rmw
    - → developers do not need to worry about the details of how rmw works.
- rclcpp
    - C++ implementation of rcl
    - Provides a set of C++ classes and functions for building and running robot applications using rcl
    - Provides a number of features and tools that make it easier to develop and debug robot applications.
- rclpy (resp. rcljava)
    - same as rclpp by in python (resp. java)


ROS2
Application Code
rclpp
rclpy
rcljava
ROS Client Lib (rcl)
ROS Abstract Middleware (rmw)
Specific DDS rmw Implementation
Fast DDS
Cyclone DDS
Connext DDS
Transport Layer (Network, SM, Serial, Radio)
Operating System (Linux, Windows, MacOS)
Hardware Layer

Images from medium.com

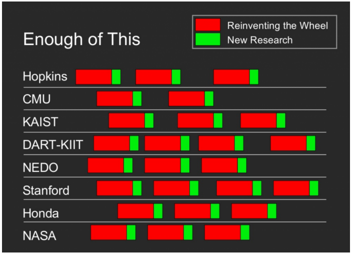
What ROS is not

Robot Operating System

- Not a (computer) operating system but rather a Meta Operating System
  - Officially available on Ubuntu Gnu/Linux
  - Experimental support for: macos, MS Windows, Fedora, Gentoo, Debian. . .
- Not a programming language
  - ROS programs written in C++ and Python.
  - Experimental: Java, Lisp, Octave. . .
- Not a hard real-time environment
- Not a development environment
  - Can be used from IDE, text editor and command line

History of ROS 1/3

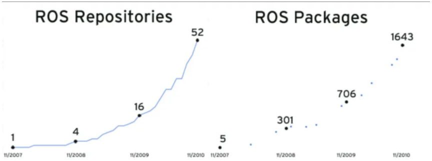
- Originally from Stanford University (2006)
  - personal project of Keenan Wyrobek and Eric Berger



Slide from Eric and Keenan pitch deck

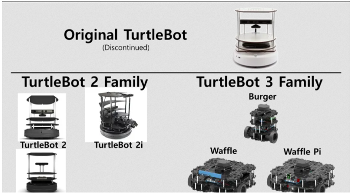
History of ROS 2/3

- 2008 Investment by Willow Garage
  - Fist version in 2009 (ROS Mango Tango)
  - . . .
  - ROS Groovy Galapagos, in 2012



From Willow Garage

- TurtleBot launched on 2011



History of ROS 3/3

- 2014 Open Source Robotics Foundation
  - ROS Hydro Medusa, in 2013
  - . . .
  - ROS Melodic Morenia, in 2018

  - ROS2 realeased on 2017

Why ROS ?

- Open Source:
  - Active community, still evolving and improving

Why ROS ?

- Open Source:
  - Active community, still evolving and improving
- Reusability:
  - Provides various open-sourced tools and software, making it easy to contribute, adapt and share

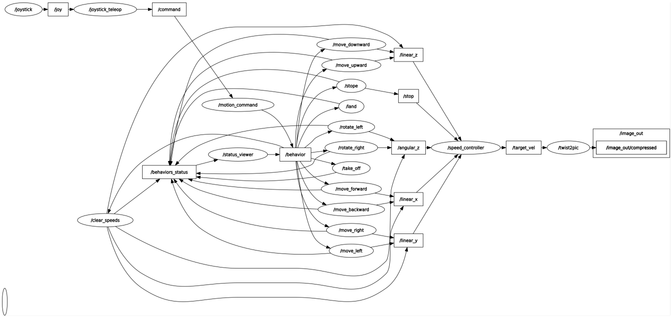
Why ROS ?

- Open Source:
  - Active community, still evolving and improving
- Reusability:
  - Provides various open-sourced tools and software, making it easy to contribute, adapt and share
- Support for Development Tools:
  - Debugging tools, 2D & 3D visualization tools (Rviz)

Why ROS ?

- Open Source:
  - Active community, still evolving and improving
- Reusability:
  - Provides various open-sourced tools and software, making it easy to contribute, adapt and share
- Support for Development Tools:
  - Debugging tools, 2D & 3D visualization tools (Rviz)
- Rapid Testing:
  - Before testing on physical robots, ROS provide simulators (Gazebo) and simple way to record and playback sensor data (rosbags)

Computational graph



The computational graph is a network of nodes (a distributed system) that solves a problem:

- Nodes are processes doing some computation
  - nodes are represented with rectangles in the graph
- Nodes talk to each others by sending messages
  - communication between nodes are represented by arrows and ellypses

Nodes

- ROS applications consist of a composition of individual nodes
  - Nodes perform narrow tasks
  - Nodes are are decoupled from other parts of the system
    - Individual processes that do not share memory
  - Nodes talk to each other using a publish-subscribe or request-response messaging patterns (see next slide)

Ex: Drones

- A node linking ROS and the drone driver
- A node linking ROS and the joystick driver
- Several nodes that manage basic behavior
- A node that manages different behaviors
- ...

Messages

- Messages are bits of data sent from one node to another over a topic.
- They are written in the ROS Message Description Language (.msg)
- Many common messages are available in `https://github.com/ros2/common_interfaces`

Example: ColorRGBA.msg

```
float32 r
float32 g
float32 b
float32 a
```

Custom messages

If you need a custom message

- Create a specific package
  - Do not add you custom message in the package it is used to prevent dependecy issues
- Use semantically meaningful names
- By convention, name this package `custom_name`_interfaces
- By convention, use CamelCase for the name of the interface.
  - Ex: “TargetCoordinates.msg”
- Your message can include any number of
  - ROS2 primitive data types
  - Or already existing messages

Example: Command.msg

```
string command
```

Publisher and Subscriber

- Publishers and Subscribers are key components for communication.

Publisher

- Role:
  - Sends data (typed messages) to a specific topic.
- Characteristics:
  - Publishes messages of a specific type.
  - Multiple publishers can write to the same topic.

Example (in Python)

```
pub = Node.create_publisher(String, ’my_topic’)
msg = String()
msg.data = ’Hello, ROS 2!’
pub.publish(msg)
```

Publisher and Subscriber 2

Subscriber

- Role:
  - Receives data (typed messages) from a specific topic.
  - Trigger a callback functions
- Characteristics:
  - Subscribes to messages of a specific type.
  - Multiple subscribers can read from the same topic.

Example (in Python)

```
Node.create_subscription(String, ’my_topic’, my_callback)

def my_callback(msg):
   print(f"Received: {msg.data}")
```

Timer

- Role:
  - Create timers to schedule callback functions at specified intervals.

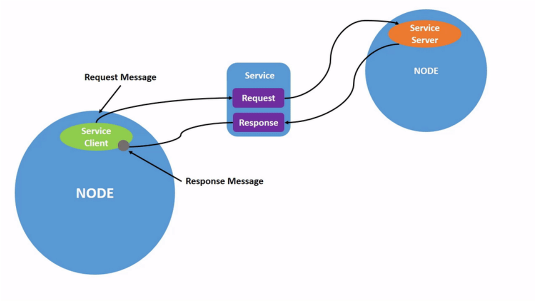
Example (in Python)

```
def timer_callback():
  # Your callback logic here
  pass

# Execute this timer_callback() every 1.5 seconds
timer = self.create_timer(1.5, timer_callback)
```

Service

- Use Cases
  - for synchronous, request-response interactions.
  - Commonly used for tasks setting parameters, requesting data, or triggering actions.
- Benefits
  - Synchronous communication for real-time control.
  - Robust error handling through response messages.



Images from ROS2 documentation

Service - .srv interface

- Defines the structure of requests and responses.
- Enables nodes to communicate seamlessly.
- Standard srv interfaces available in
  - https://github.com/ros2/common_interfaces/tree/rolling/std_srvs/srv

AddTwoInts.srv

```
int64 a
int64 b
---
int64 sum
```

Service - Server

- Role:
  - Listens for incoming requests.
  - Provides a service upon request.
- Characteristics:
  - Offers a specific service with a well-defined interface.

Example (in Python)

```
from example_interfaces.srv import AddTwoInts

class MinimalService(Node):
  def init(self):
    super().__init__(’minimal_service’)
    self.srv = self.create_service(
      AddTwoInts, ’add_two_ints’,
      self.add_two_ints_callback)

  def add_two_ints_callback(self, request, response):
    response.sum = request.a + request.b
    return response
```

Service - Client

- Role:
  - Sends requests to a service server.
  - Receives responses from the server.
- Characteristics:
  - Calls a specific service with the same well-defined interface as the server

Example (in Python)

```
import sys

from example_interfaces.srv import AddTwoInts

class MinimalClientAsync(Node):

  def __init__(self):
    super().__init__(’minimal_client_async’)
    self.cli = self.create_client(AddTwoInts, ’add_two_ints’)
    while not self.cli.wait_for_service(timeout_sec=1.0):
      self.get_logger().info(’service not available, waiting’)
    self.req = AddTwoInts.Request()
```

Service - Client 2

Example (In Python)

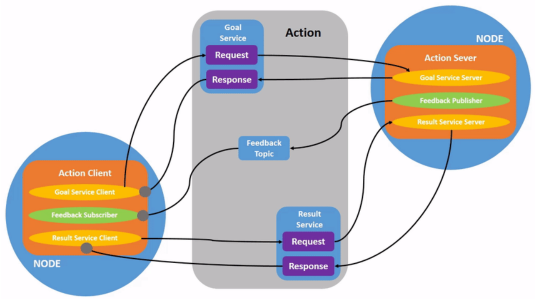
```python
class MinimalClientAsync(Node):

  [...]

  def send_request(self, a, b):
    self.req.a = a
    self.req.b = b
    self.future = self.cli.call_async(self.req)
    rclpy.spin_until_future_complete(self, self.future)
    return self.future.result()
```

Actions

- Use Cases
  - Actions are suitable for long-running, asynchronous tasks.
    - ★ Ex: path planning, robot navigation, . . .
- Benefits
  - Goal status monitoring and feedback.
    - ★ enable clients to send goals and servers to execute and provide steady feedback, as opposed to services which return a single response.
  - Similar to services (client-server) except actions are preemptable (you can cancel them while executing).
  - Robust error handling through result messages.



ros2 commandline

- `ros2 pkg list`
  - Lists all the packages that are currently installed in your ROS 2 workspace.
- `ros2 pkg create`
  - Creates a new ROS 2 package in your workspace.

ros2 commandline

- `ros2 pkg list`
  - Lists all the packages that are currently installed in your ROS 2 workspace.
- `ros2 pkg create`
  - Creates a new ROS 2 package in your workspace.
- `ros2 pkg lint`
  - Lints one or more packages to ensure that they meet the ROS 2 package guidelines.

ros2 commandline

- `ros2 pkg list`
  - Lists all the packages that are currently installed in your ROS 2 workspace.
- `ros2 pkg create`
  - Creates a new ROS 2 package in your workspace.
- `ros2 pkg lint`
  - Lints one or more packages to ensure that they meet the ROS 2 package guidelines.
- `ros2 pkg test`
  - Runs the tests for one or more packages in your ROS 2 workspace.

ros2 commandline

- `ros2 pkg list`
  - Lists all the packages that are currently installed in your ROS 2 workspace.
- `ros2 pkg create`
  - Creates a new ROS 2 package in your workspace.
- `ros2 pkg lint`
  - Lints one or more packages to ensure that they meet the ROS 2 package guidelines.
- `ros2 pkg test`
  - Runs the tests for one or more packages in your ROS 2 workspace.
- `ros2 doctor`
  - checks all aspects of ROS 2, including platform, version, network, environment, running systems and more

Bag files

- A binary file format for recording and playing back ROS 2 data.
- Allows to record and play back sensor data, messages, and events.
  - Example: Record sensor data during a robot's operation for later analysis.
  - Debugging: Replay bag files to identify issues in your robot's behavior.
  - Share bag files to enable others to replicate your experiments.

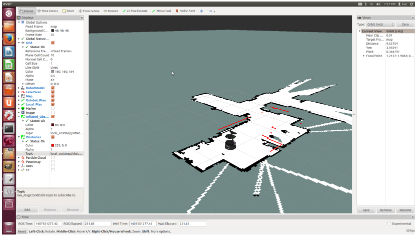
Content of a Bag File

- Timestamped data
  - each message in a bag file has a timestamp indicating when it was recorded.
- Topics and messages
- Metadata
  - ROS version, start date, end date, duratino, bag size, . . .

Third party tools - RVIZ

RViz (ROS Visualization)

- 3D visualization software tool for robots, sensors, and algorithms
- Enables to see the robot's perception of its world (real or simulated).



Global Options	
Fixed Frame	map
Background C...	48; 48; 48
Frame Rate	30
✓ Global Status:...	
Grid	✓
✓ Status: Ok	
Reference Fra...	<Fixed Frame>
Plane Cell Count	10
Normal Cell C...	0
Cell Size	1
Line Style	Lines
Color	160; 160; 164
Alpha	0.5
Plane	XY
Offset	0; 0; 0
RobotModel	✓
LaserScan	✓
Map	✓
Golobal_Plan	✓
Local_Plan	✓
Marker	□
Image	□
Inflated_Obs...	✓
✓ Status: Ok	
Color	85; 0; 0
Alpha	1
Topic	local_costmap/inflat...
Obstacles	✓
✓ Status: Ok	
Color	255; 0; 0
Alpha	1
Topic	local_costmap/obst...
Particle Cloud	□
PoseArray	□
Axes	□
TF	□

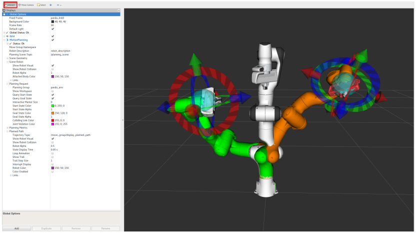
Topic
nav_msgs::GridCells topic to subscribe to.

Type: Orbit (rviz) Zero

Current View	Orbit (rviz)
Near Clip ...	0.01
Target Fra...	map
Distance	9.52754
Yaw	3.95541
Pitch	0.364797
Focal Point	1.2137; 1.9063; 0...

ROS Time: 1407551277.42 ROS Elapsed: 251.65 Wall Time: 1407551277.46 Wall Elapsed: 251.65

Left-Click: Rotate. **Middle-Click:** Move X/Y. **Right-Click/Mouse Wheel:** Zoom. **Shift:** More options.



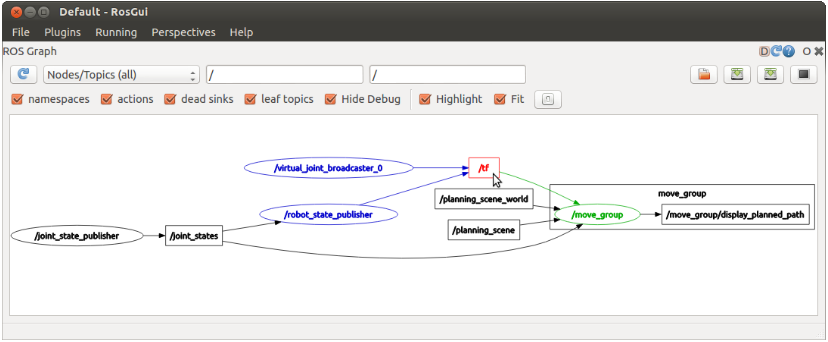
Global Options	
Fixed Frame	panda_link0
Background Color	48; 48; 48
Frame Rate	30
Default Light	✓
✓ Global Status: Ok	
Grid	✓
MotionPlanning	✓
✓ Status: Ok	
Move Group Namespace	
Robot Description	robot_description
Planning Scene Topic	/planning_scene
Scene Geometry	
Scene Robot	
Show Robot Visual	✓
Show Robot Collision	□
Robot Alpha	1
Attached Body Color	150; 50; 150
Links	
Planning Request	
Planning Group	panda_arm
Show Workspace	□
Query Start State	✓
Query Goal State	✓
Interactive Marker Size	0
Start State Color	0; 255; 0
Start State Alpha	1
Goal State Color	250; 128; 0
Goal State Alpha	1
Colliding Link Color	255; 0; 0
Joint Violation Color	255; 0; 255
Planning Metrics	
Planned Path	
Trajectory Topic	/move_group/display_planned_path
Show Robot Visual	✓
Show Robot Collision	□
Robot Alpha	0.5
State Display Time	0.05 s
Loop Animation	□
Show Trail	□
Trail Step Size	1
Interrupt Display	□
Robot Color	150; 50; 150
Color Enabled	□
Links	

Global Options

- Rviz is NOT a simulation tool

Third party tools - Rqt_graph

- rqt_graph displays a visual graph of the processes running in ROS and their connections.



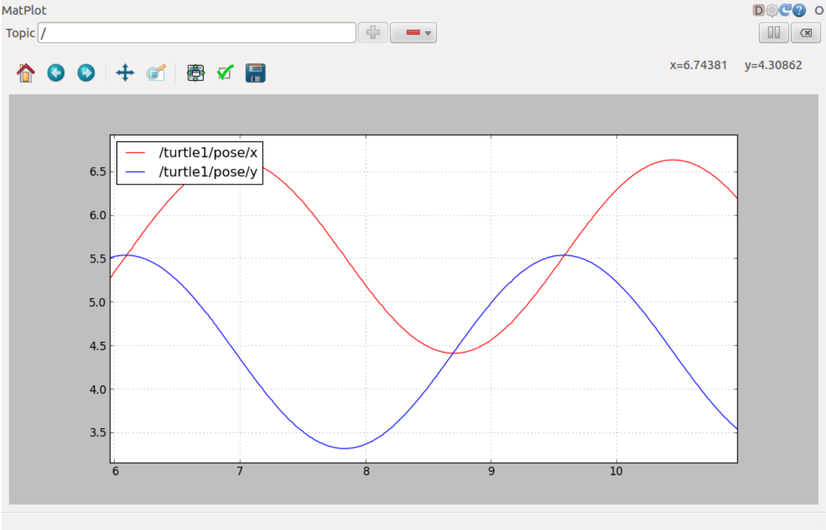
Nodes/Topics (all) | / | /

☑ namespaces ☑ actions ☑ dead sinks ☑ leaf topics ☑ Hide Debug ☑ Highlight ☑ Fit

From ROS documentation

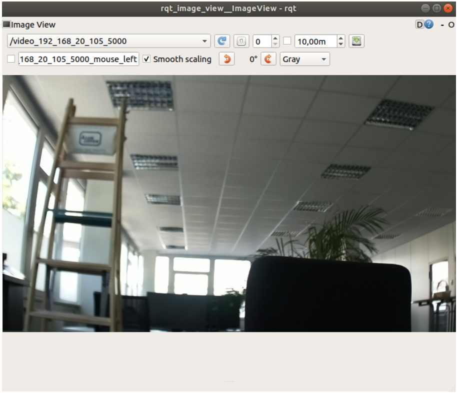
Third party tools - Rqt_plot

- rqt_plot lets you visualize scalar data published to ROS topics.



Third party tools - Rqt_image_view

Rqt_image_view allow to visualize image/video topics.



rqt_image_view__ImageView - rqt

Image View

/video_192_168_20_105_5000

0

□ 10,00m

□ 168_20_105_5000_mouse_left

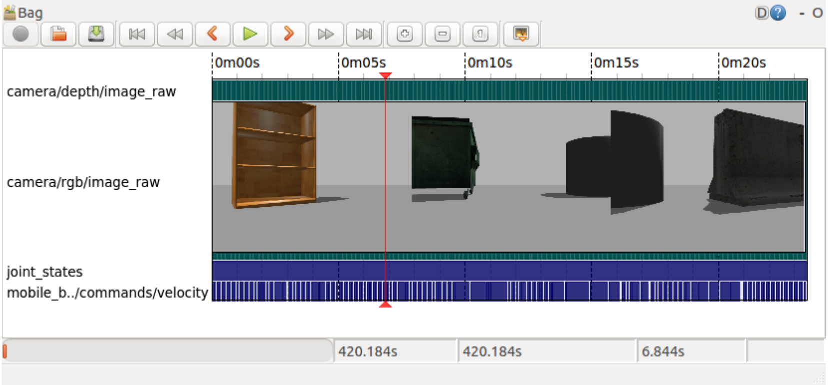
☑ Smooth scaling

0°

Gray

Third party tools - Rqt_bag

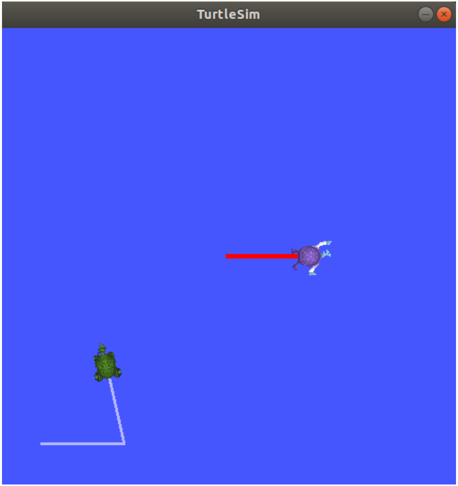
rqt_bag is a visualizer that lets you see and run graphically data recorded in bag files.



From ROS documentation

Turtle

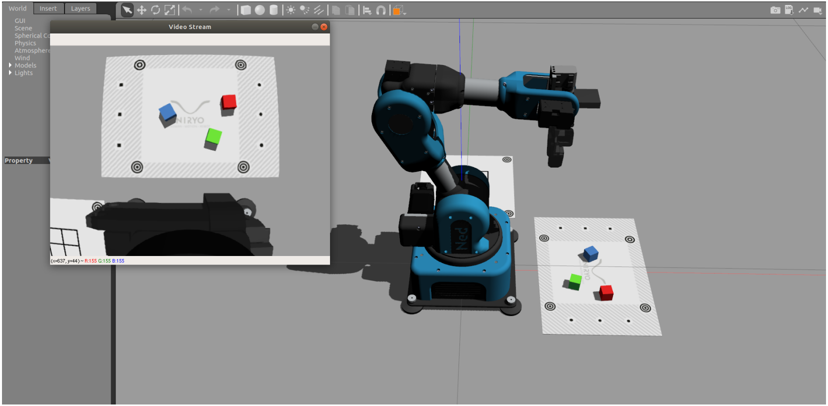
Turtlesim is a lightweight simulator for learning ROS 2.



Images from ros documentation

Simulation with gazebo

Gazebo is a robotics design, development, and simulation suite.



(x=637, y=44) ~ R:155 G:155 B:155