

ROS 2 Tutorial

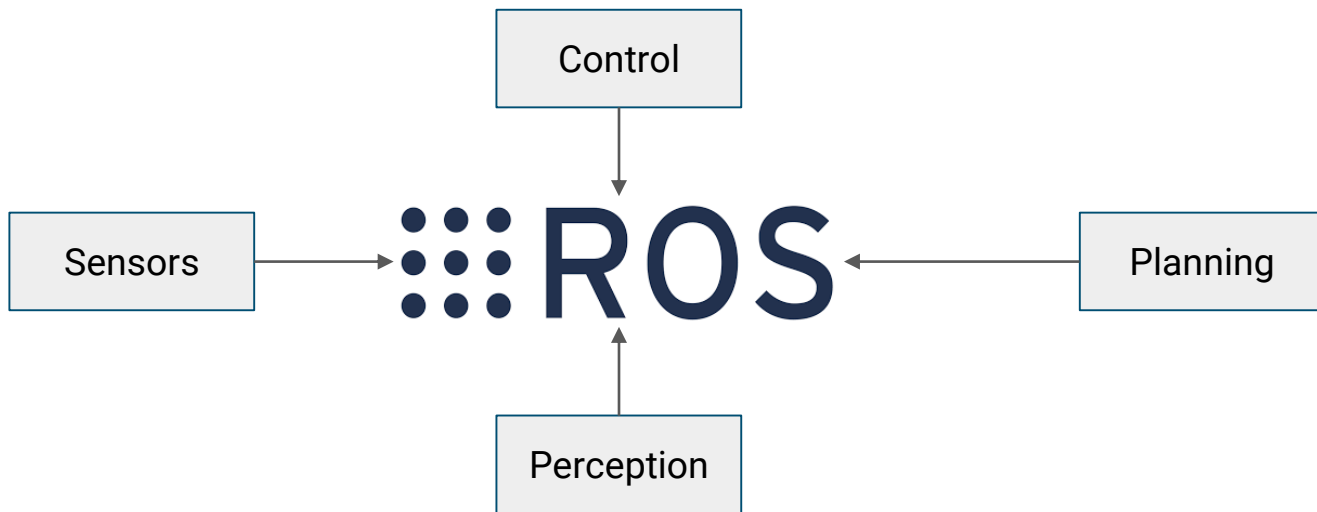
Charvi Shukla, Shiladitya Biswas

Outline

- What is ROS? Why ROS?
- Core Concepts of ROS 2
- Useful Tools & Packages
- Demos
- Useful references

What is ROS?

- Robot Operating System, but not your typical OS
- Middleware that glues all parts of robots together



ROS vs ROS 2

The ROS logo consists of a 3x3 grid of dots to the left of the text "ROS" in a large, bold, sans-serif font.

- Beginning of ROS (EOL 2025)

The ROS 2 logo consists of a 3x3 grid of dots to the left of the text "ROS 2" in a large, bold, sans-serif font, with a trademark symbol (TM) to the upper right of the "2".

- Support Real-time
- Multi-robot communication
- Cross-platform (Linux, Windows, macOS, embedded)
- Better security

Why ROS?

- Standardize communication between modules
- Huge collections of packages (drivers, SLAM, navigation, manipulation,...)
- If you want to get a job...

TLDR; No reinventing the wheel for every robot + Every robotics company uses ROS

Core concepts and terminology in ROS/ROS 2

- Node
 - A process that perform a specific task
 - Ex: LiDAR driver node, object detection node, or motor controller node
 - Robots will run several nodes at once

Core concepts in ROS/ROS 2

- Node
 - A process that perform a specific task
- Topic
 - A channel for **streaming data** between nodes
 - Publisher
 - Subscribers
 - Example:
 - LiDAR node can publish a point cloud on `/lidar/points_raw`
 - Object detection can subscribe to `/lidar/points_raw`

Core concepts in ROS/ROS 2

- Node
 - A process that perform a specific task
- Topic
 - A channel for **streaming data** between nodes
- Message
 - [Data Structure exchange](#) over topics
 - Make sure ros2 domain id is the same!
 - A ROS 2 domain ID is a number used to separate different logical networks that share the same physical network

sensor_msgs/PointCloud2 Message

File: `sensor_msgs/PointCloud2.msg`

Raw Message Definition

```
# This message holds a collection of N-dimensional points, which may
# contain additional information such as normals, intensity, etc. The
# point data is stored as a binary blob, its layout described by the
# contents of the "fields" array.

# The point cloud data may be organized 2d (image-like) or 1d
# (unordered). Point clouds organized as 2d images may be produced by
# camera depth sensors such as stereo or time-of-flight.

# Time of sensor data acquisition, and the coordinate frame ID (for 3d
# points).
Header header

# 2D structure of the point cloud. If the cloud is unordered, height is
# 1 and width is the length of the point cloud.
uint32 height
uint32 width

# Describes the channels and their layout in the binary data blob.
PointField[] fields

bool    is_bigendian  # Is this data bigendian?
uint32  point_step    # Length of a point in bytes
uint32  row_step       # Length of a row in bytes
uint8[] data          # Actual point data, size is (row_step*height)

bool is_dense         # True if there are no invalid points
```

Ref: ROS documentation

Core concepts in ROS/ROS 2

- Node
 - A process that perform a specific task
- Topic
 - A channel for **streaming data** between nodes
- Message
 - Data Structure exchange over topics
- Service
 - A channel that will only provide data upon request from a client
 - Example:
 - Request: SaveMap
 - Response: MapSavedSuccess (bool)

Core concepts in ROS/ROS 2

- Node
 - A process that perform a specific task
- Topic
 - A channel for **streaming data** between nodes
- Message
 - Data Structure exchange over topics
- Service
 - A channel only provide data upon request
- Launch file
 - A script to start multiple nodes with configured parameters all at once

Core concepts in ROS/ROS 2

- Node
 - A process that perform a specific task
- Topic
 - A channel for **streaming data** between nodes
- Message
 - Data Structure exchange over topics
- Service
 - A channel only provide data upon request
- Launch file
 - A script to start multiple nodes with configured parameters all at once

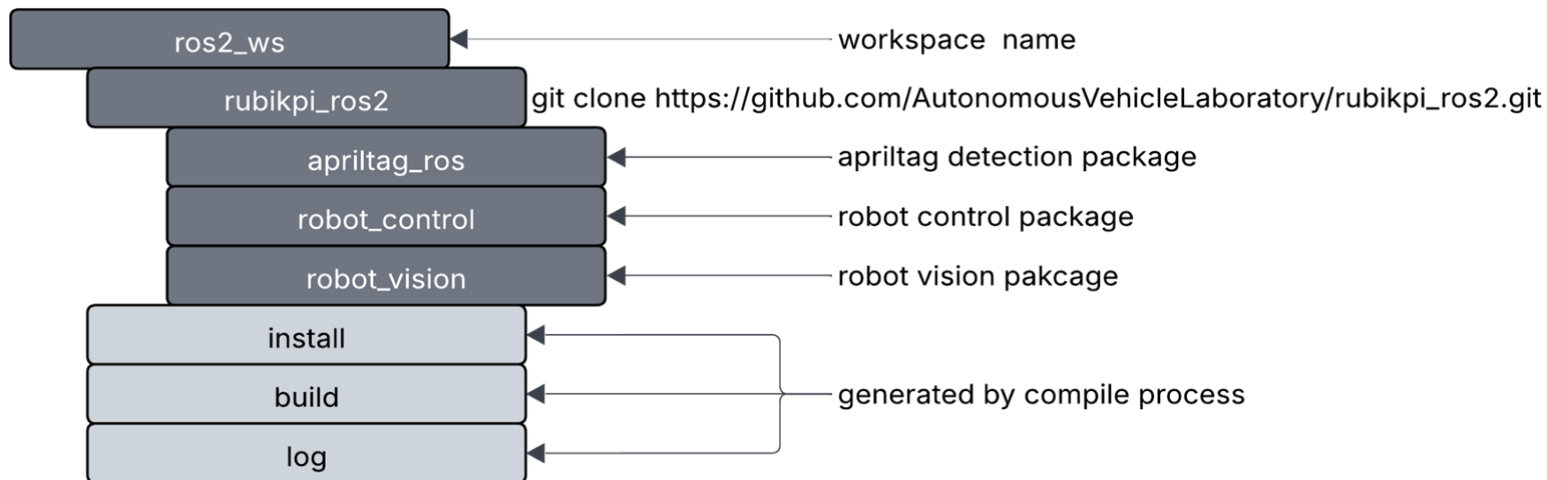
Pre-requisites: Useful tools / packages

- **tmux**
 - A tool that can manage multiple terminal sessions in a single window, even after disconnecting
- **Rviz2 / foxglove**
 - ros2 visualization tools that can
 - visualize sensor data
 - overlay output on top of sensor data

Quick Tmux Demo

ROS 2 Workspace

- A workspace is a directory containing ROS 2 packages.
- Important: Before using ROS 2, it's necessary to source your ROS 2 installation workspace (command on future slide)



ROS 2 Packages

- A package is an organizational unit for your ROS 2 code.
 - If you want to be able to install your code or share it with others, then you'll need it organized in a package
 - A package will have the following structure:

```
my_package/  
  package.xml  
  resource/my_package  
  setup.cfg  
  setup.py  
  my_package/  
  |
```

- Creating a ROS 2 Package (Python Example)

- `cd ~/ros2_ws/src`
- `ros2 pkg create --build-type ament_python <package_name>`

ROS 2 Packages

- Building a ROS 2 Package
 - `cd ~/ros2_ws`
 - To build all packages in the workspace run: `colcon build`
 - To build a specific package in the workspace, run: `colcon build --packages-select <my_package>`
 - If you would like to use symbolic links (instead of copying files into install/): `colcon build --symlink-install`
- To use your new package and executable, first open a new terminal and source your main ROS 2 installation.
 - `source ~/ros2_ws/install/setup.bash` **(IMPORTANT!!)**
- Also make sure that you have ROS 2 Jazzy in your path
 - `source /opt/ros/jazzy/setup.bash` **(IMPORTANT!!)**

Running the nodes and listing topics

- Open setup.py and add the entry points for the nodes

- `entry_points={`
 - `'console_scripts': [`
 - `'node_name = <ros_ws_name>.<name_of_node_file>:main',`
 - `],`
 - `}`

- Run the nodes

- `ros2 run ros2_ws_name node_name`

- Inspect the topics

- `ros2 topic list`
 - `ros2 topic echo /topic_name`
 - `ros2 topic info /topic_name`
 - `ros2 topic hz /topic_name`

Let's code some basic ROS 2 Nodes!

Example Publisher Node (code)

- In ROS 2, `rclpy.node.Node` is the primary class used to create a ROS 2 node in Python.
 - Can be used to create publisher, subscriber, service etc.

```
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 |
        self.timer = self.create_timer(timer_period, self.timer_callback)

        self.i = 0

    def timer_callback(self):
        msg = String()
        msg.data = 'Welcome to CSE 276A: %d' % self.i
        self.publisher_.publish(msg)
        self.get_logger().info('Publishing: "%s"' % msg.data)
        self.i += 1

def main(args=None):
    rclpy.init(args=args)
    minimal_publisher = MinimalPublisher() # initialize publisher
    rclpy.spin(minimal_publisher)         # "spins" the node so callback!
    minimal_publisher.destroy_node()      # Destroy the node explicitly
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Example Subscriber Node (code)

- Furthermore, we can make a node to subscribe to the publisher

```
import rclpy
from rclpy.node import Node
from std_msgs.msg import String

class MinimalSubscriber(Node):

    def __init__(self):
        super().__init__('minimal_subscriber')
        self.subscription = self.create_subscription(
            String,
            'topic',
            self.listener_callback,
            10)
        self.subscription  # prevent unused variable warning

    def listener_callback(self, msg):
        self.get_logger().info('I heard: "%s"' % msg.data)

def main(args=None):
    rclpy.init(args=args)
    minimal_subscriber = MinimalSubscriber()
    rclpy.spin(minimal_subscriber)
    minimal_subscriber.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()
```

Foxglove - A 3rd-party tool

Web-based & cross-platform.

Multi-system support.

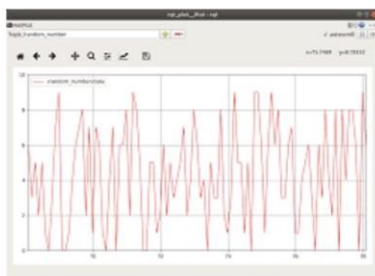
Data logging & playback.

Open source.

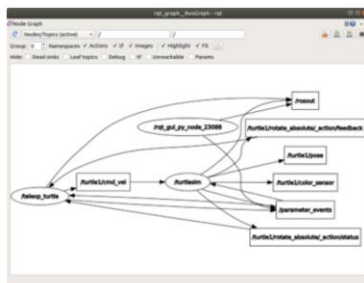
Quick Foxglove Demo

Viewing the TF tree

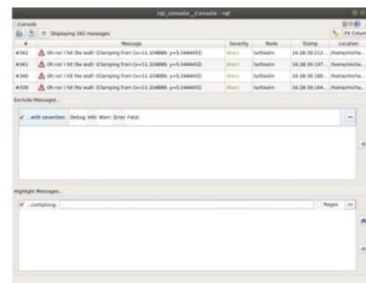
- Can run: `ros2 run tf2_tools view_frames`
- Alternatively, you can use RQT:
 - RQt is an another graphical user interface (GUI) framework in ROS2.
 - RVIZ2 -> focuses on 3D data visualization (maps, point clouds, robot models)
 - RQT -> focuses on GUI tools for debugging & monitoring (plots, topic inspection, logs)



plots



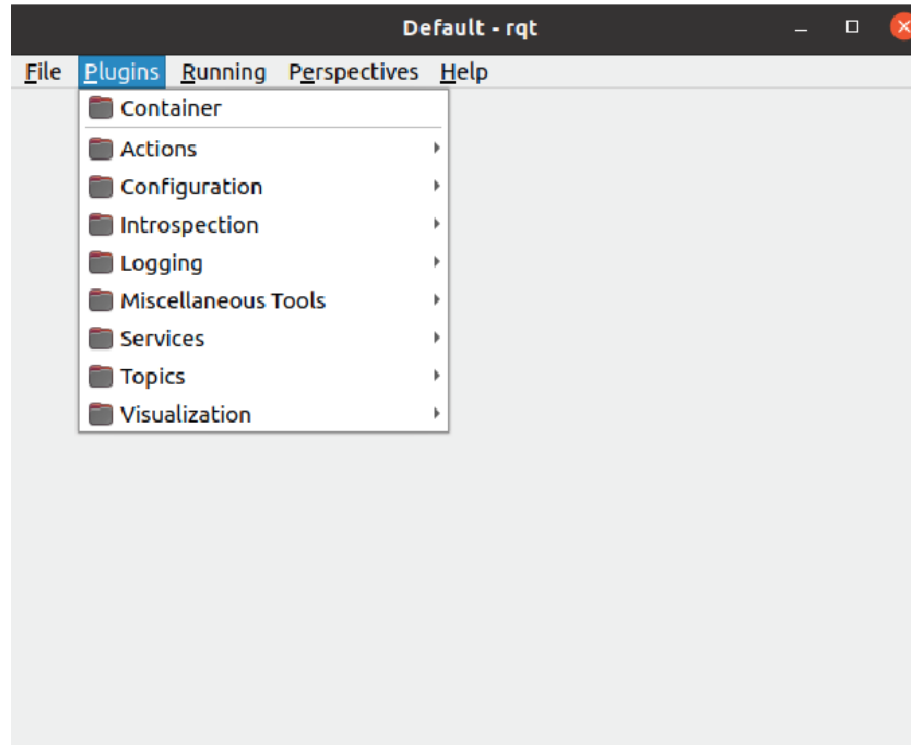
logs



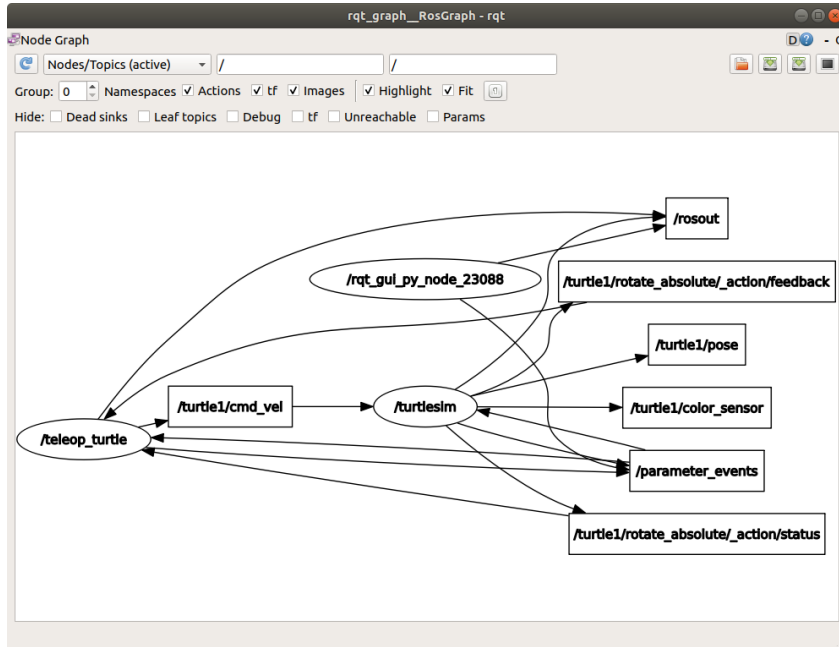
nodes

RQT

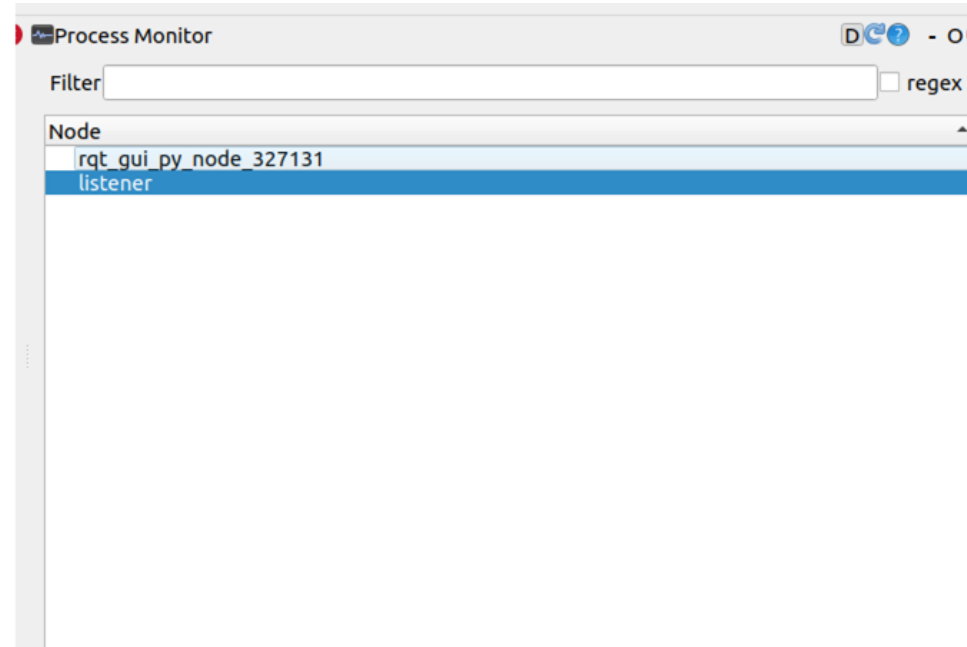
Also plugin-based design.



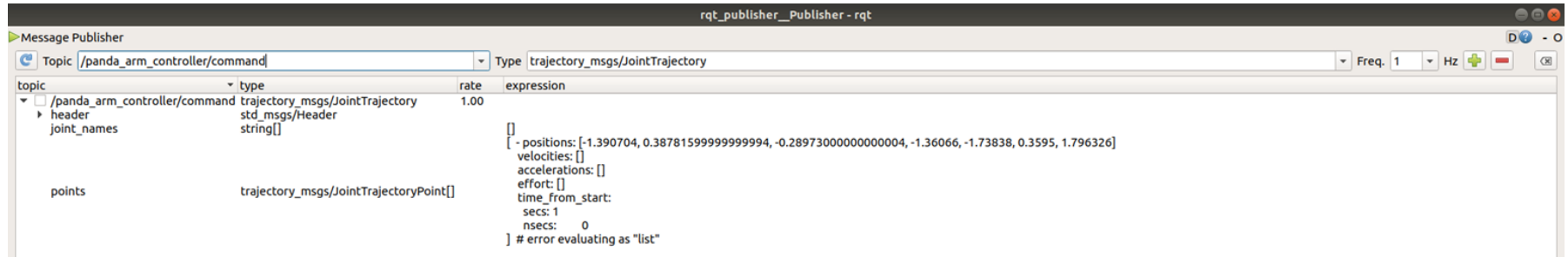
Introspection / Node Graph Monitor



Introspection / Process Monitor



Topic/ Message Publisher



Service /Service Caller

Service Caller

Service:

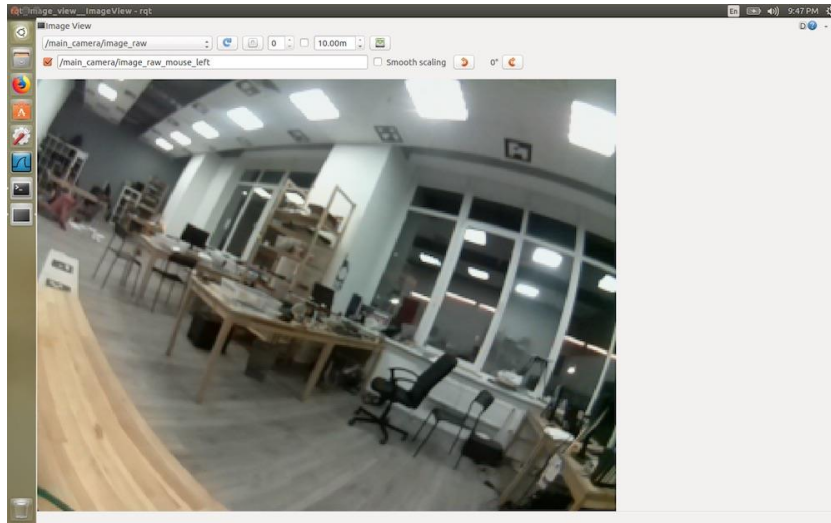
Request

Topic	Type	Expression
▼ /listener/describe_parameters names	rcl_interfaces/srv/DescribeParameters sequence<string>	[]

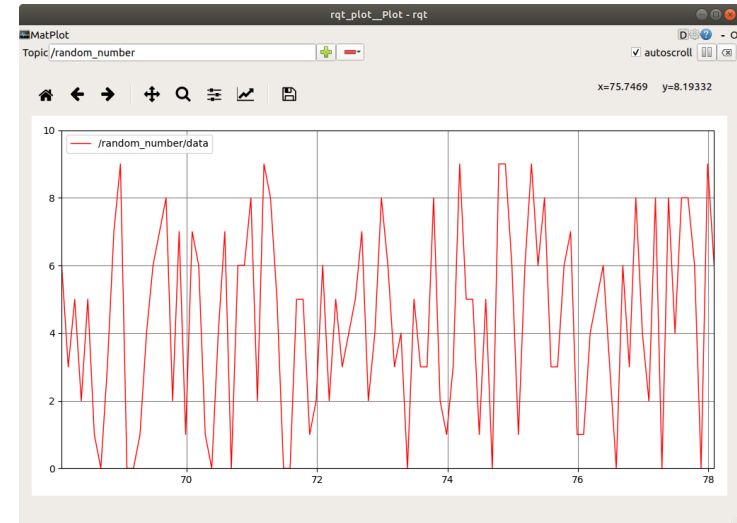
Response

Field	Type	Value
-------	------	-------

Image

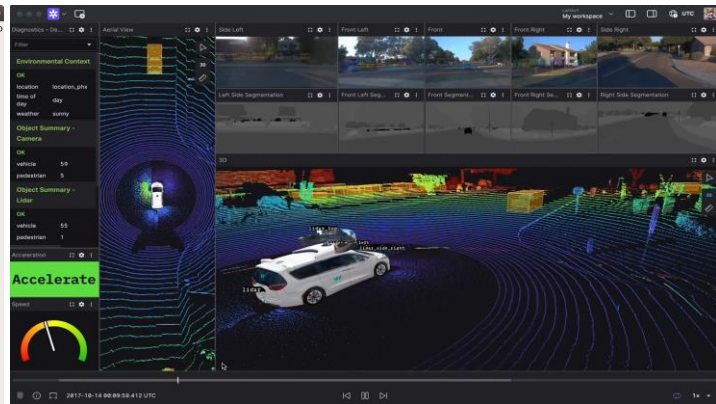
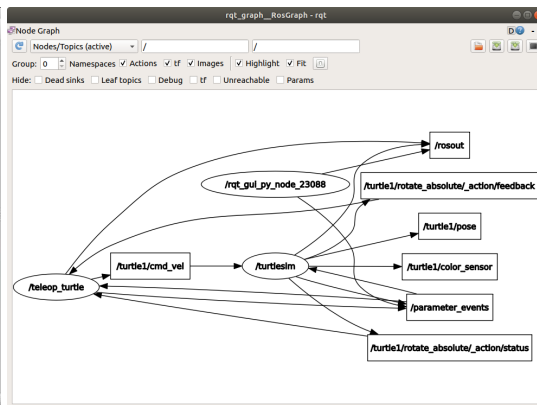
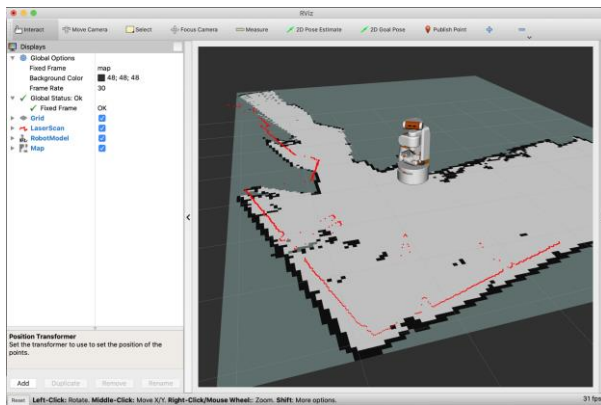


Matplot



Visualization Tools

- RVIZ2 – 3D visualization of robot data
- RQT (ROS2 QT) – Flexible GUI for debugging and monitoring
- Foxglove – Modern web-based visualization and analysis (we saw earlier)



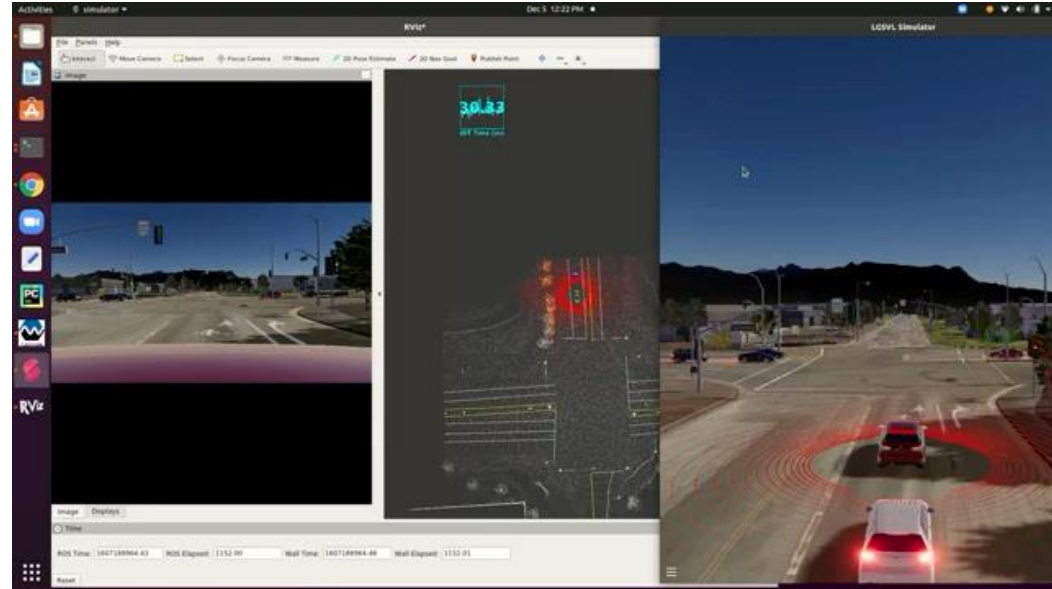
RVIZ2

What data does RVIZ2 visualize?

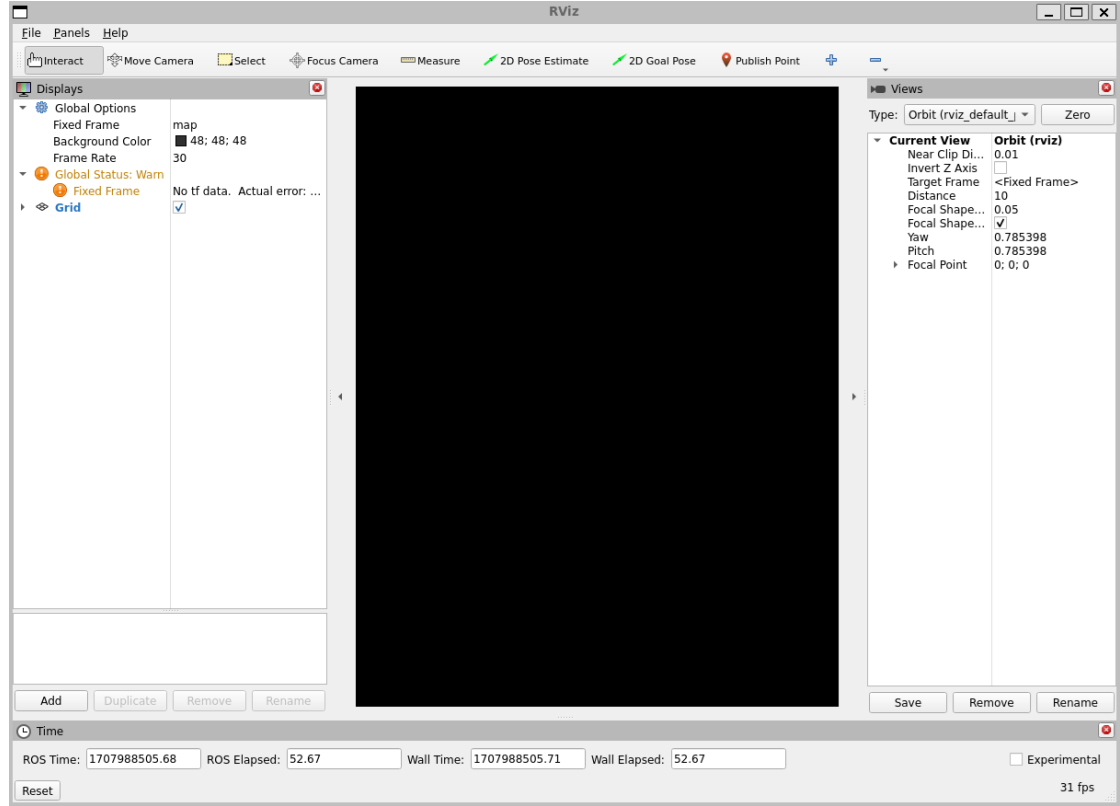
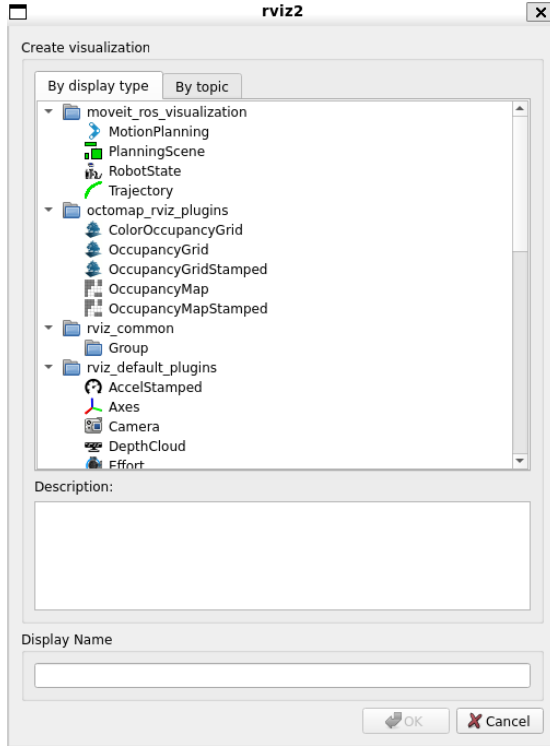
Image, Point Cloud, Map, TF(transform), Robot model。

Why we need visualization?

Making abstract data intuitive and visible.



Plugin-based design.

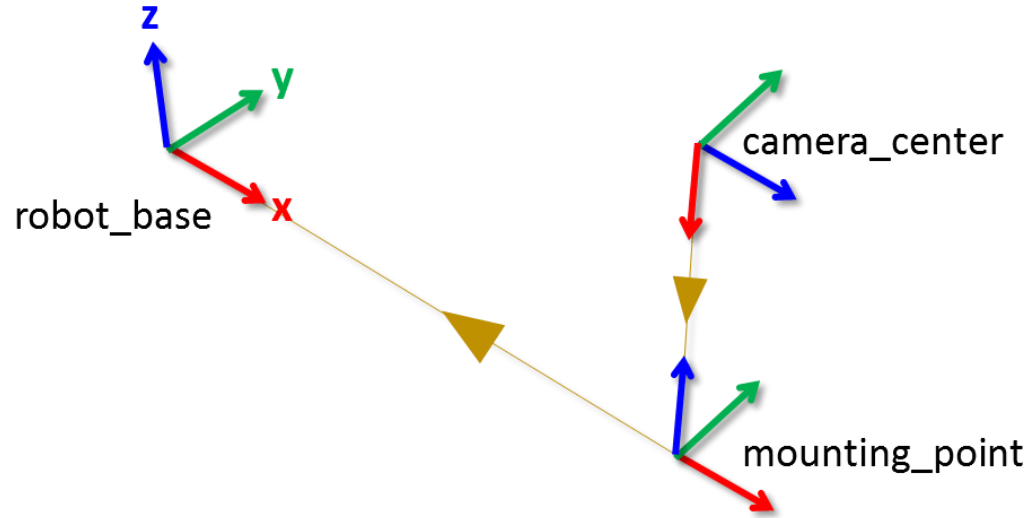


Frame

A frame = a coordinate system

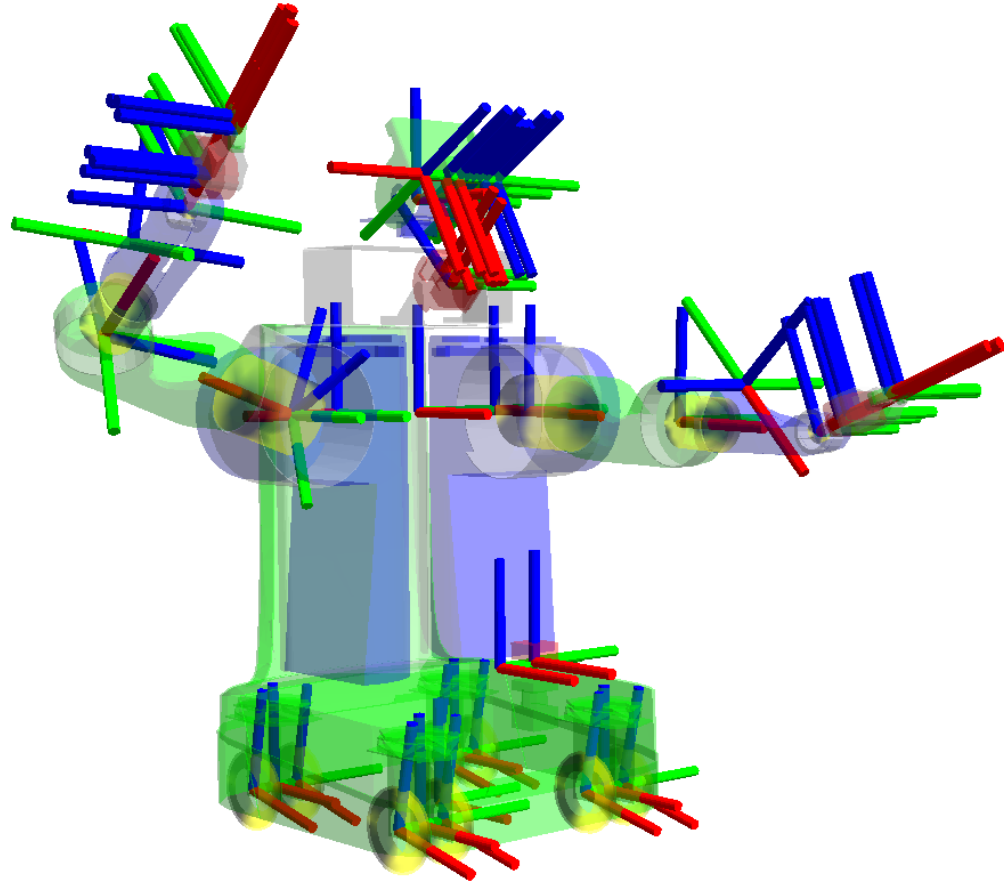
Includes position and orientation information.

Use transformation matrices (rotation + translation) to convert between frames



TF2 (TransForm)

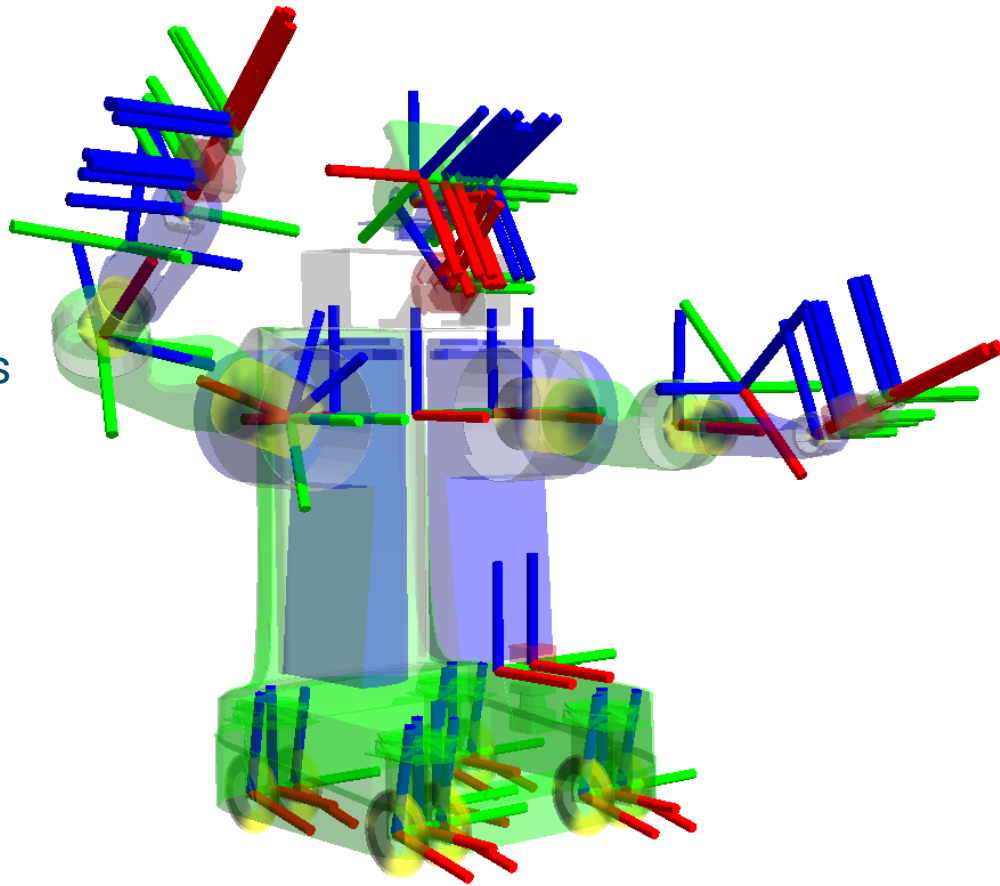
Lets try this.



TF2 (TransForm)

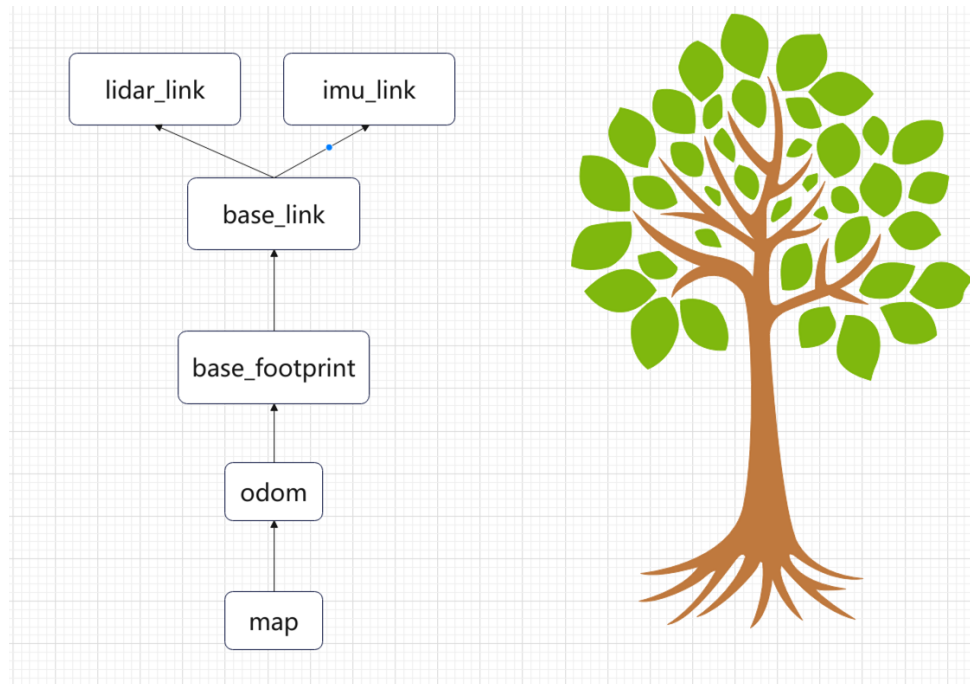
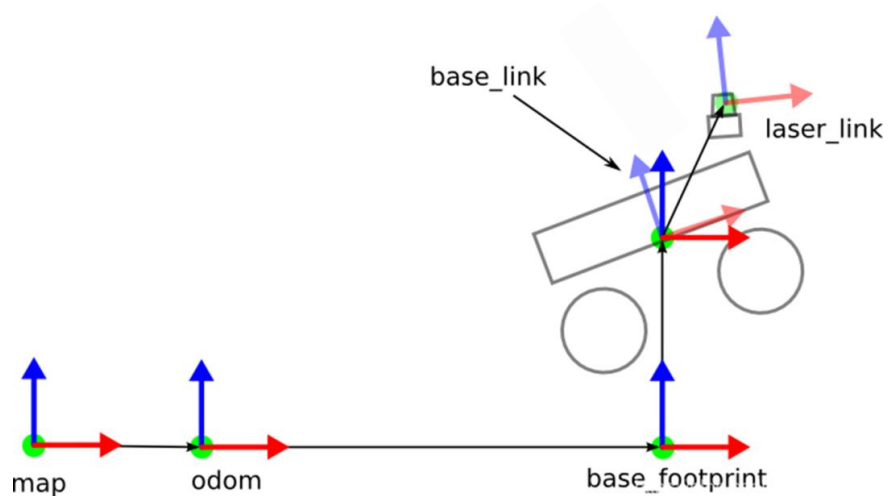
Lets try this.

TF2 is the transform library, which lets the user keep track of multiple coordinate frames over time.



TF2 (TransForm)

TF Tree Example



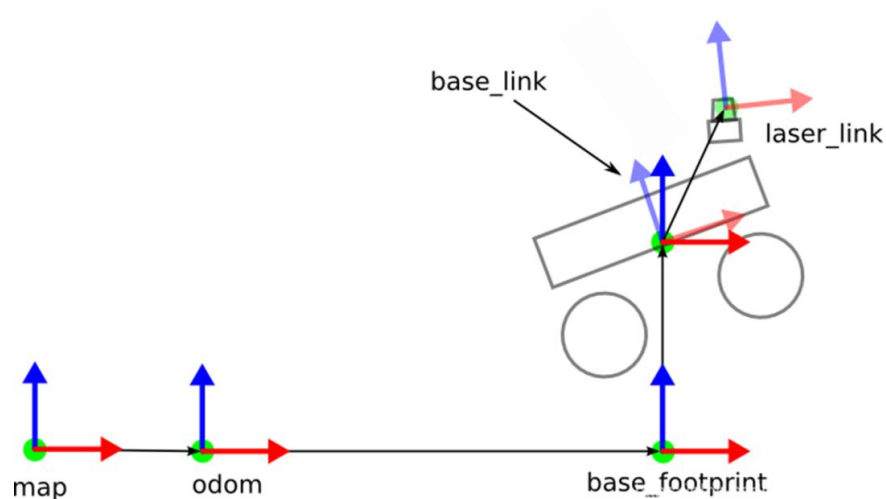
TF2 (TransForm)

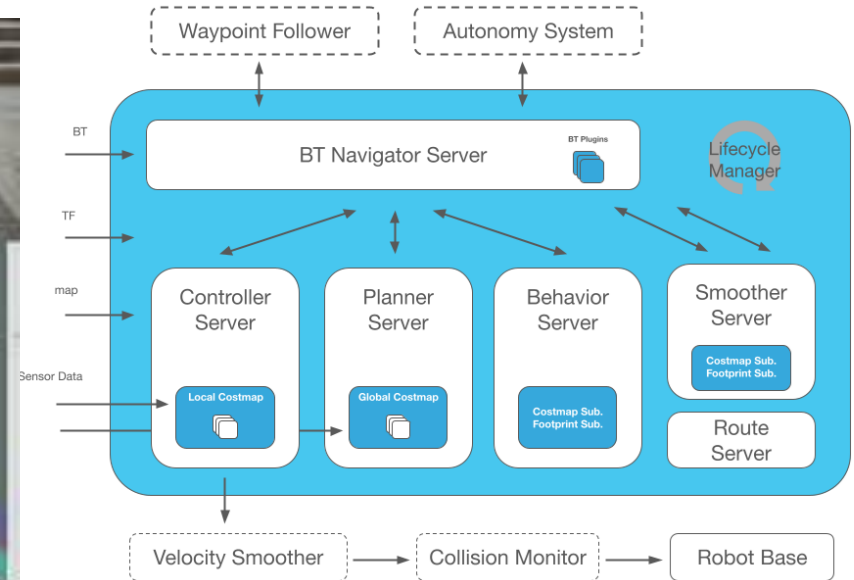
Static TF

By defining transformations between the base_link frame (robot frame) and each sensor frame

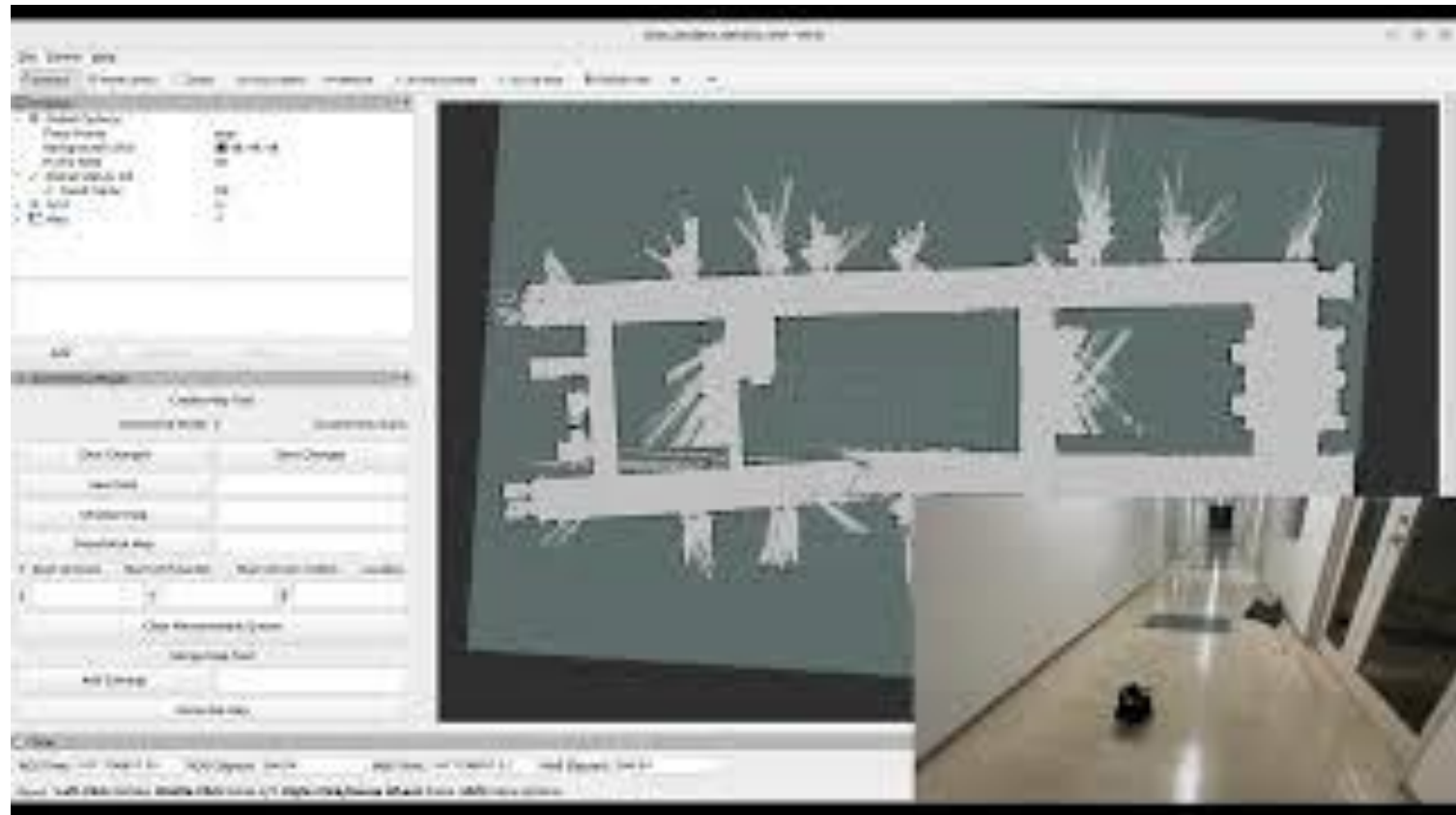
Dynamic TF

Used for moving joints, wheels, manipulators

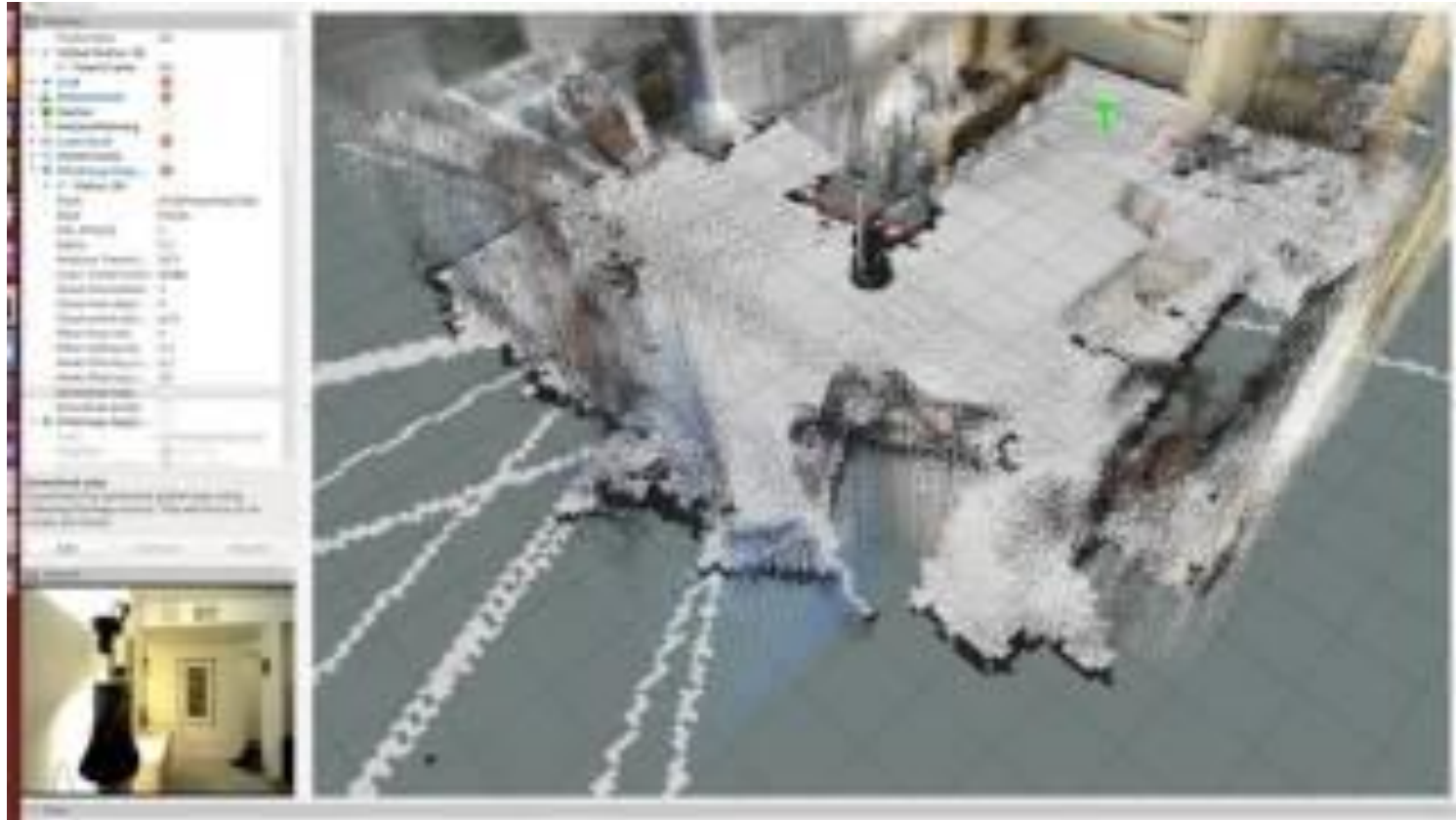




slam_toolbox



RTAB-Map



Virtual Machine

Windows -> Dual boot systems

-> virtualbox

MacOS -> virtualbox or UTM

Or Use docker

<http://foxglove.dev/blog/installing-ros2-on-macos-with-docker>

The Power of the ROS2 Community

- Thousands of developers worldwide
- Growing ecosystem of packages and tools
- Continuous improvements and new features
- Open source: anyone can contribute
- You can be part of it too! 🚀



Congratulations, you now know ROS2!

Congratulations, you now know ROS2!

HW1: build a Mars Rover with ROS2...



Demo Time

Tmux -> [link](#)

Turtlesim -> [link](#)

TF2 -> [link](#)

Nav2 -> [link](#)

Reference

Waverover robot: [link](#)

Rubik Pi 3: [link](#)

ROS2 Jazzy: [link](#)

Nav2: [link](#)

Slam_toolbox: [link](#)

RTABMap: [link](#)

Open-RMF: [link](#)

Virtual Box: [link](#)

Dual Boot: [link](#)

UTM: [link](#) UTM Guide: [link](#)

[Tmux cheat sheet](#)