ROS 2 Introduction

Mobile Robotics

What is ROS 2?

- Second generation of the Robot Operating System
  - *Not actually an OS*
  - We are using ROS 2 Humble Hawksbill (or just Humble)

- Open source framework (used by hundreds of companies) which works closely with the OS to control all sorts of robots.
- Can be used on Windows, MacOS, or Linux
  - Linux has by far the best support and compatibility
- Made up of modular building blocks that communicate with each other in a node graph structure
  - Allows control of actuators, sensors, and control systems

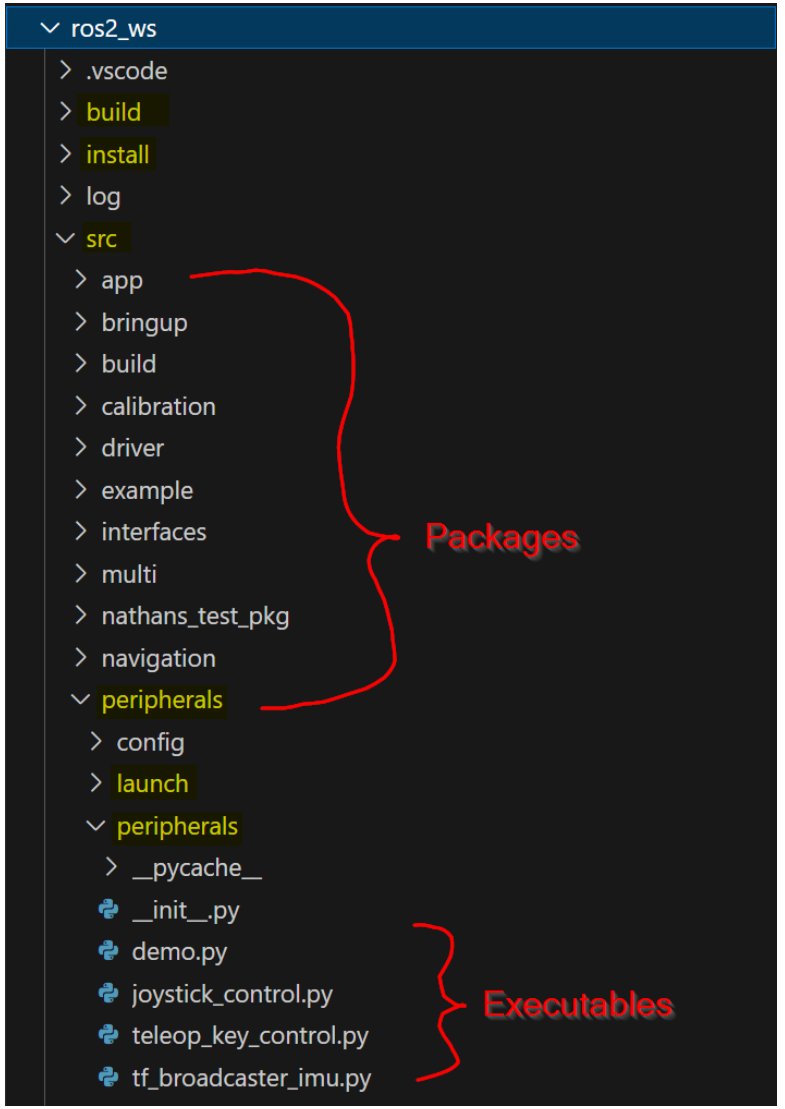
- Examples of uses include
  - NASA's Robonaut 2
  - iRobot's Vacuum robots
  - Spirit AeroSystems' Scan-N-Plan™
  - Used a lot in academic research and R&D

Helpful Learning Resources

- ROS2 Tutorials and Docs
  - Beginner: CLI tools — ROS 2 Documentation: Humble documentation
- ArticulatedRobotics YouTube Channel
- The Construct
  - Helpful tutorials with virtual environments to practice in
  - Requires free account for virtual environments, but tutorials can be found on YouTube

File Structure

- All code specific to MentorPi is in ros_ws directory
  - Here you will find install, build, and src directories as well as some files used by the robot during setup
  - Write pkgs in source directory
  - Build and install are for building and running pkgs
- Each package in src will have a directory inside with the same name
  - This is where we write nodes
- They also house other files
  - Launch files
  - setup.py (which must be configured before building)



Sourcing the ROS 2 Environment

- To use ROS 2 command shortcuts in Linux we must source the base ROS 2 installation in every terminal that we want to have access to those commands in
  - Sourcing the environment runs a series of commands which make tools more accessible to us

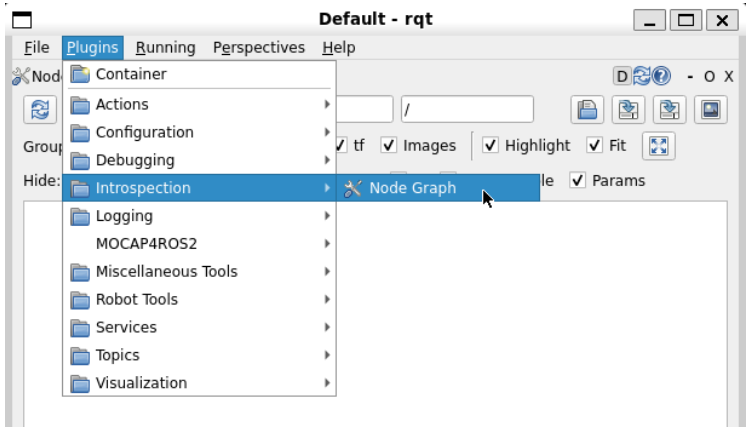
- $ source /opt/ros/humble/setup.sh
  - The MentorPi robots do this automatically when a new terminator terminal is opened

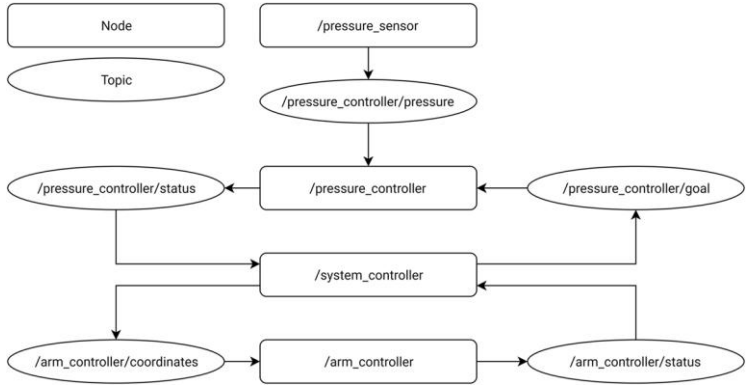
- ROS packages we create are conventionally stored in a workspace
- Each time you build a package, we must source this workspace
  - $ source ~/ros2_ws/install/setup.zsh
  - Make sure you have sourced the base environment first

Using RQT to Visualize Node Graph

- RQT is a powerful ROS 2 tool used to view information from ROS 2
- Launch RQT using $ rqt
  - There are many plugins to explore
  - The most basic is the Node Graph
- The Node Graph has a launch shortcut since it is used so often
  - $ rqt_graph

- Here you can view the ROS 2 node tree, which in some cases can become very complex

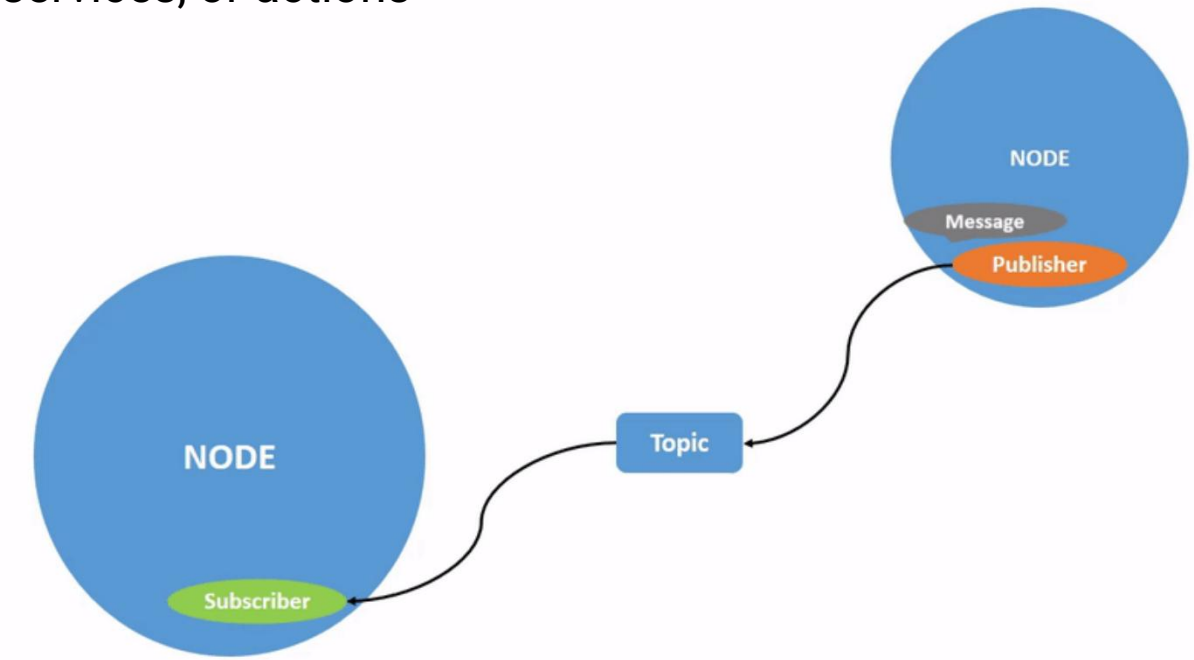




Nodes

- Nodes are the most basic building block of ROS
- Each node should be ideally in charge of a single, modular task
- Nodes can send and receive data from other nodes via topics, services, or actions

- Some possible nodes include
  - Obstacle detection
  - Battery monitoring
  - Gripper control

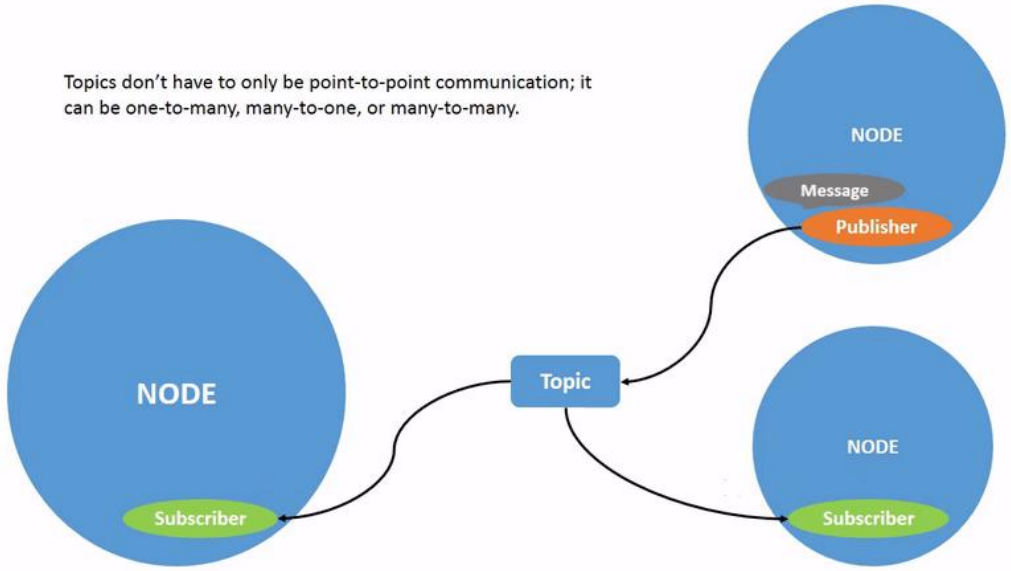


Common Node Commands

Keyword	Description	Example
run	Launches an executable (which typically contains one node)	ros2 run <pkg_name> <executable_name>
<remapping>	Remap default node properties (node name, topic names, etc.)	ros2 run <pkg_name> <executable_name> --ros-args --remap __node:=new_node_name
list	Lists running nodes	ros2 node list
info	Lists ROS graph connections that interact with the node (subscribers, publishers, services, actions, etc.)	ros2 node info <node_name>

Topics

- Topics are the glue between nodes, acting as a bus to carry information between them
- Nodes can publish data to topics, while also being subscribed (to receive data) to other topics
- Each topic has a specific message type (data structure) that it carries
  - Published messages must be in this format
- Examples of possible topics
  - /object_detections
  - /battery_status
  - /robot1/gripper_position



Common Topic Commands

Keyword	Description	Example
list	Lists running topics (-t to show msg types)	ros2 topic list -t
echo	Prints the message from the topic	ros2 topic echo <topic_name>
info	Shows type, publisher count, sub count	ros2 topic info <topic_name>
Interface show	Shows what type of message the topic is expecting	ros2 interface show <msg_type>
pub	Manually publish messages to the topic -r <value> to specify rate in HZ message is sent --once -w <num> to wait until <num> topics are listening, then publish one msg	ros2 topic pub --once --w 2 <topic_name> <msg_type> “<args>”
find	Searches all topics for the specified message type	ros2 topic find <msg_type>

Important Command to Stop MentorPi

```
ros2 topic pub --once --w 1 /controller/cmd_vel
geometry_msgs/Twist "linear:
  x: 0.0
  y: 0.0
  z: 0.0
angular:
  x: 0.0
  y: 0.0
  z: 0.0"
```

Creating pkgs

1. To create a new pkg
   - navigate to ~/ros_ws/src
   - $ ros2 pkg create <your_package_name> --build-type ament_python

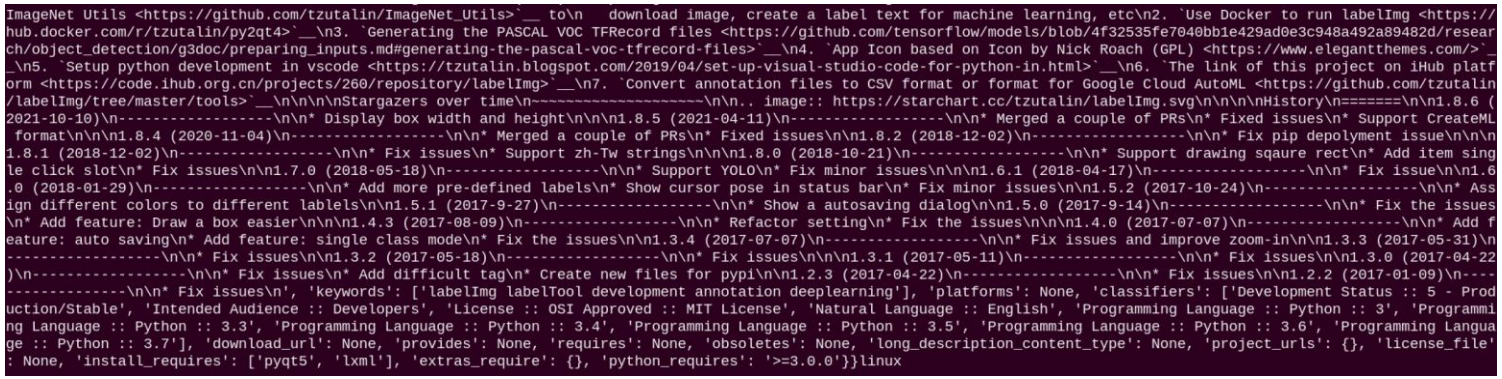
Talker Listener Nodes

Building pkgs

1. Nodes and launch files must be added to setup.py inside entry points

```
#setup.py file
entry_points={
        'console_scripts': [
            ‘node_name = pkg_name.node_file_name:main',
        ],
<!-- package.xml file -->
<depend>rclpy</depend>
<!-- add the following line if you need to reference an external pkg in a launch file -->
<exec_depend>external_pkg_name</ exec_ depend>
```

2. ANYTIME BEFORE BUILDING make sure you are in ~/ros_ws
   - Otherwise build and install files will go in places you don’t want them to



Building pkgs

3. Build command
   - $ colcon build --symlink-install --packages-select <your_package_name>

4. After building, remember to source the ws
   - $ source ~/ros2_ws/install/setup.zsh

- When should you rebuild pkgs?
  - Even with symlink installs, you need to rebuild your package if:
  - You change the setup.py or package.xml files.
  - You add or remove new nodes or entry points.
  - You modify C++ or other compiled components.

Parameters

- Configurable inputs for nodes
- Can be set when running nodes individually from the CLI or set from a launch file
- Examples of parameters
  - Setting which color we want to isolate in an image
  - An offset to adjust for noise in different lighting conditions

Common Parameter Commands

Keyword	Description	Example
list	Lists parameters associated with active nodes	ros2 param list
get	Prints current value of parameter	ros2 param get <node_name> <parameter_name>
set	Changes a parameter at runtime	ros2 param set <node_name> <parameter_name> <value>
Load param file at node startup	Set multiple parameters using a parameter file (typically a .yaml file) at startup	ros2 run <package_name> <executable_name> --ros-args --params-file <file_name>
dump	Outputs all parameters of a node	ros2 param dump <node_name>
load	Set multiple parameters using a parameter file (typically a .yaml file)	ros2 param load <node_name> <parameter_file>

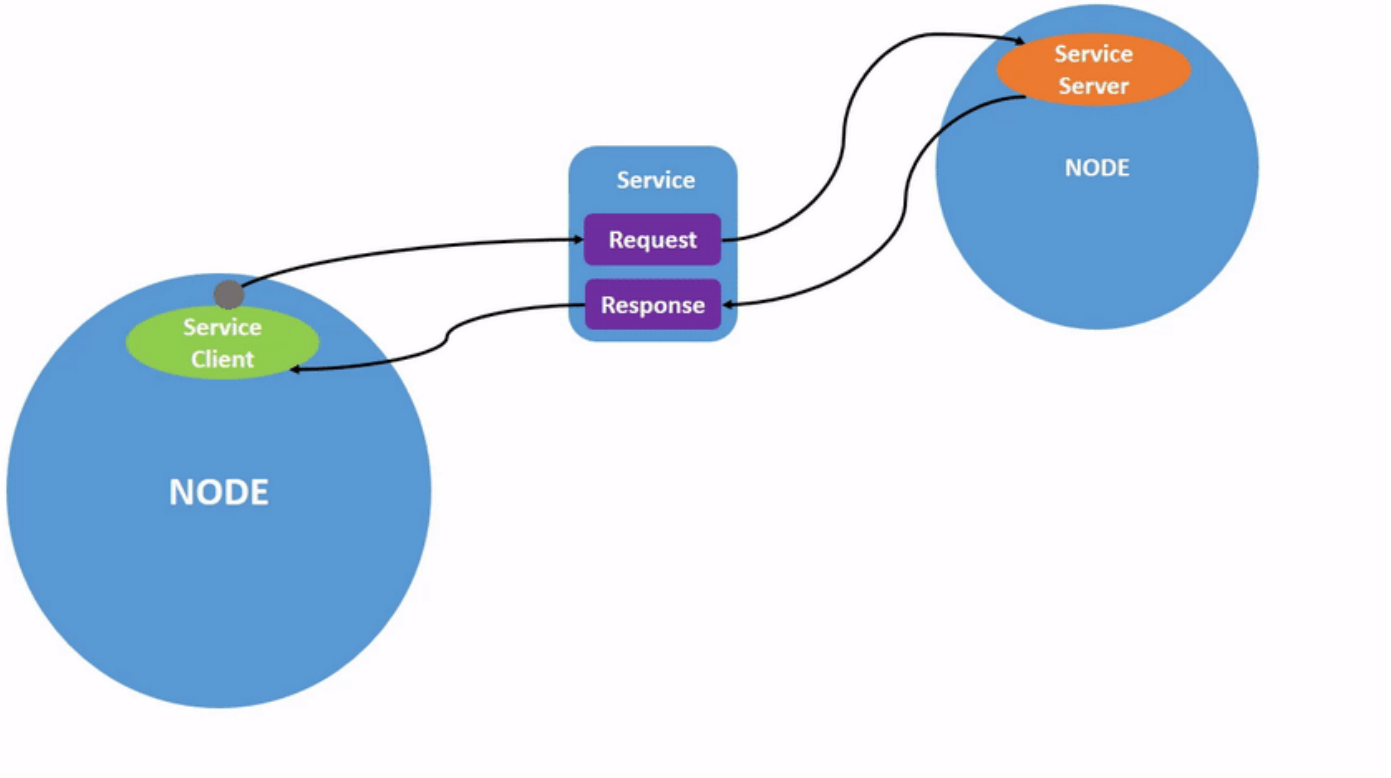
Adding Parameter to Talker Listener

Services

- Another method of communicating between nodes, similar to topics
- However while topics continuously receive and send msgs, services only do provide data when called

- Can be manually run from RQT

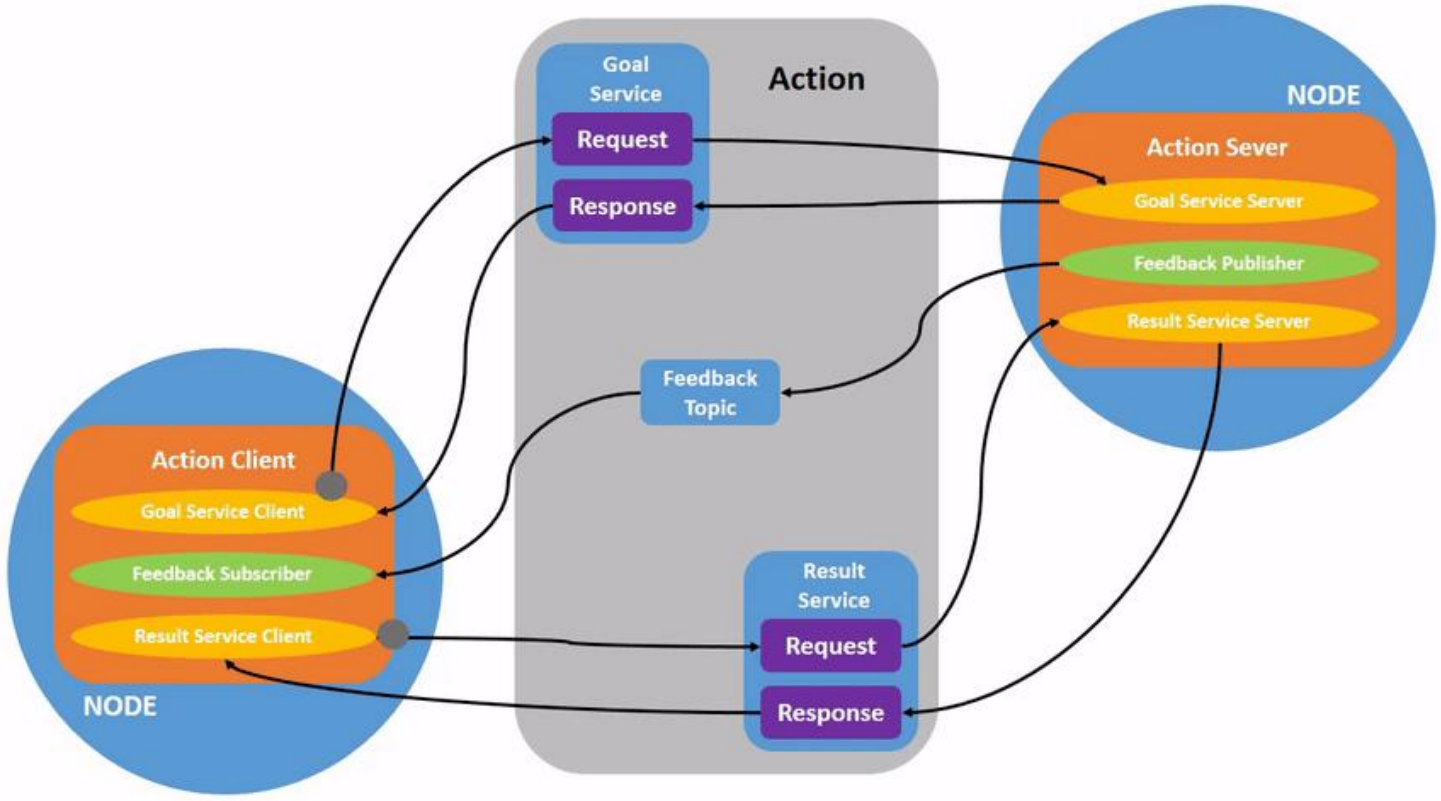
- Examples of services
  - Save a SLAM map
  - Toggle obstacle avoidance
  - Get current pose



Actions

- Another communication type intended for “long” running tasks
- Have three parts: a goal, feedback, and result

- Examples of actions
  - Move to a waypoint
  - Patrol a specified area
  - Calibrate sensors



Launch Files

- Launch files are used to help the user describe the configuration of the system and execute it.
  - Includes what programs to run, where to run them, what arguments to pass, etc.
- Can start and stop different nodes, respond to events, and even run other launch files

- Importantly for us, we can use launch files to launch systems that are pre-defined on the robot
  - e.g. booting up the camera or lidar, initializing the robot pose, etc.

- To use a launch file: $ ros2 launch pkg_name <launch_file_name>

Add Launch Files to Pkg

1. Create directory in pkg named “launch”
2. Write launch files in this directory
3. Add the following to setup.py

```
import os
from glob import glob


# Add the following in the data files list of the setup function
(os.path.join('share', package_name, 'launch/'),
glob('launch/*launch.[pxy][yma]*')),
```

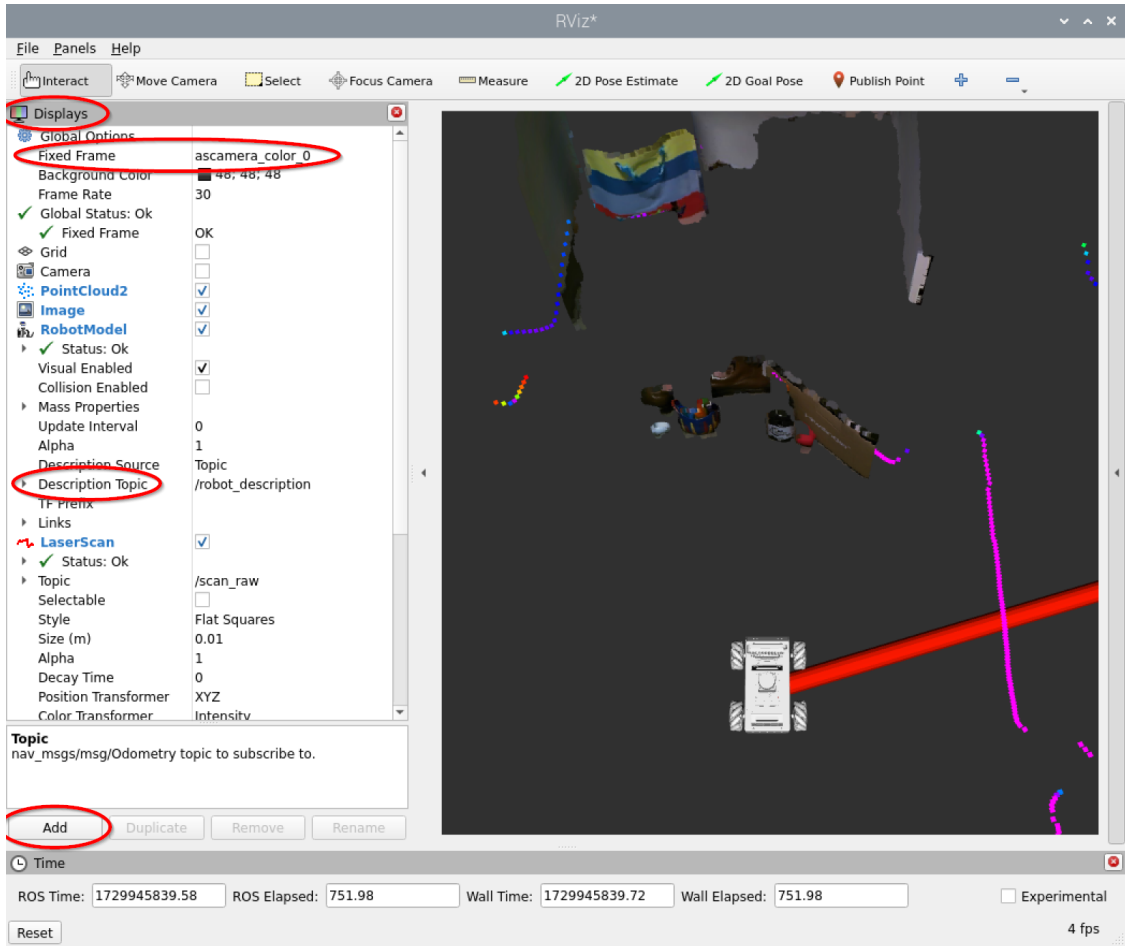
Rviz2

- Another tool to visualize data from MentorPi
- Run with: $ rviz2

- Displays panel allows you to choose what data is being displayed
  - Images from camera
  - Lidar
  - Odometry

- Fixed frame decides which transform the data is displayed from

- Each visualization has a topic parameter
  - Select which topic it should look at to get visualization from



Helpful Topics and Types

- Here are some topic and their types, which you will use often

- Camera depth image
  - /ascamera/camera_publisher/depth0/image_raw [sensor_msgs/msg/Image]

- Camera RGB image
  - /ascamera/camera_publisher/rgb0/image [sensor_msgs/msg/Image]

- Robot velocity control
  - /cmd_vel [geometry_msgs/msg/Twist]
  - Please limit to linear to “-0.6 to 0.6” and angular to “-2 to 2”

- Lidar 2D pointcloud
  - /scan_raw [sensor_msgs/msg/LaserScan]