Basics of ROS

Summer School

Soham and Tarun

June 10 2025

INTERNATIONAL INSTITUTE OF INFORMATION TECHNOLOGY

HYDERABAD

Table of Contents

Table of Contents

Our Plan

- We want to focus on the **physical aspect** of robotics, and not go by the standard method of teaching through abstract individual concepts
- We'll try to introduce concepts *implicitly* through practical examples We don't want to teach basics in isolation; we want to show how they integrate in real-world robotic systems

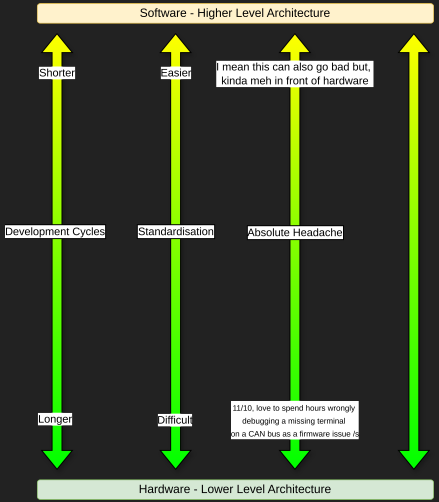
Table of Contents

Motivation behind this Lecture

- Emphasis on **physical robotics problems**:
  - Hardware–software integration challenges
  - Real-time constraints, safety, and reliability
- How to structure your software development to **tackle physical-layer issues**
- Overview of common pitfalls in robotic software design

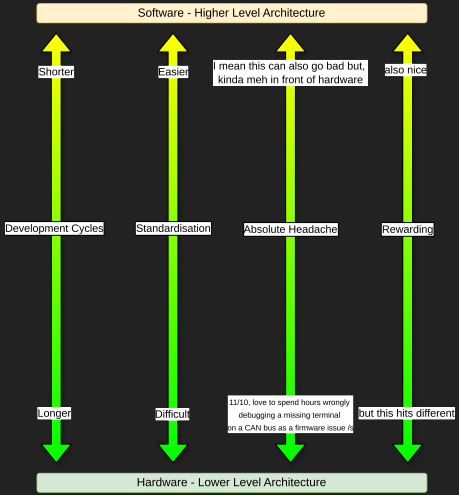
Hardware

2 Motivation behind this Lecture



Hardware

2 Motivation behind this Lecture



History and Rationale

2 Motivation behind this Lecture

- Enter Keenan Wyrobek and Eric Berger, Stanford Students
- **Origins**: Stanford AI Lab, 2007–2009 (STAIR project & PR1)
- Willow Garage adoption (late 2007); first public ROS commit on November 7, 2007
- **Philosophy**: “Don’t reinvent the wheel” — leverage community packages/tools
- Vast ecosystem of packages, tools, and active community engagement

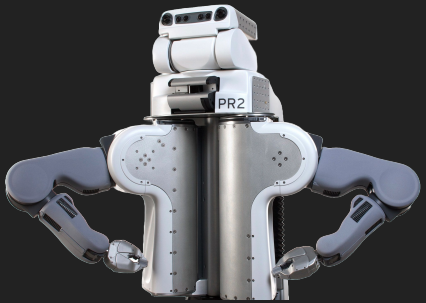


Figure 1: PR2—the flagship mobile manipulator

Don’t Reinvent the Wheel

2 Motivation behind this Lecture

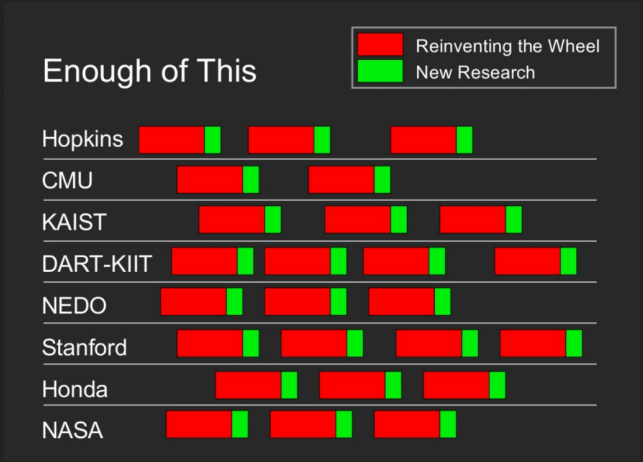


Figure 2: Actual slide from Keenan’s and Eric’s pitchdeck

And thus the community grew

2 Motivation behind this Lecture

Figure 3: ROS Releases throughout the years

Where are we now?

2 Motivation behind this Lecture

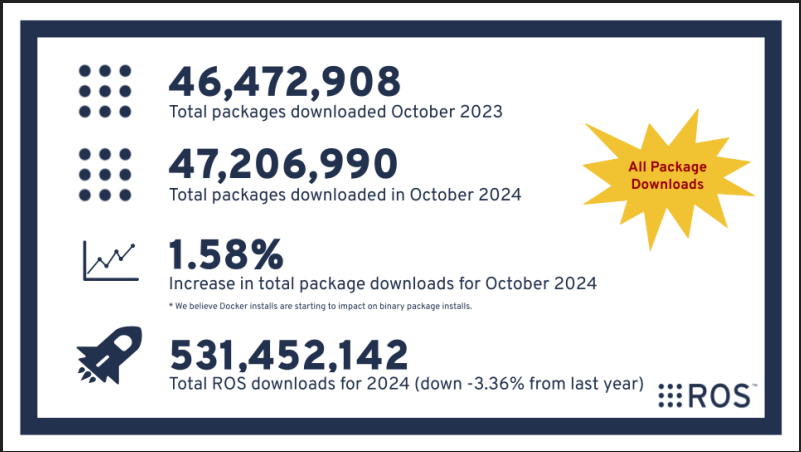


Figure 4: ROS Metrics for 2024

Table of Contents

Demo Overview: `Mecanum Robot`

3 Full-Fledged ROS Demonstration

- Objective: Show a complete ROS-based robot pipeline from simulation to hardware
- Components involved:
  1. URDF model
  2. `ros2_control` controllers
  3. `nav2` navigation stack
  4. Gazebo simulation
  5. RViz visualization
  6. Teleoperation node (teleop)
- Purpose: Provide big-picture context before diving into each module

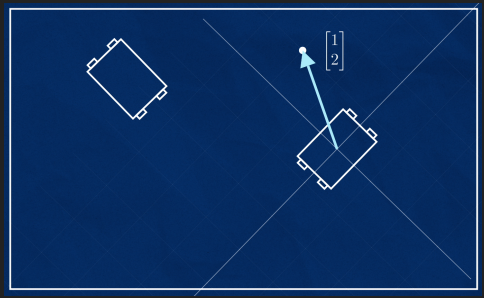
Table of Contents

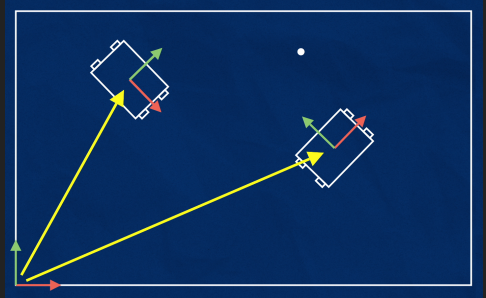
The Transform System (tf2)

4 ROS Components

Why Transforms?

- They are a mathematical tool to take points or measurements that are represented from one point of view, and represent them in a different point of view that is more useful.

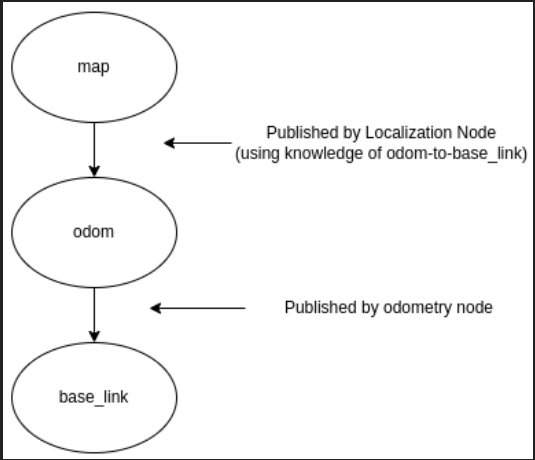




Transforms in ROS

4 ROS Components

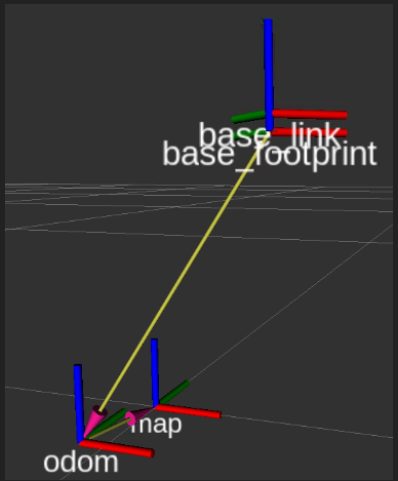
- ROS provides a system called `tf2` (TransForm version 2) to handle these transformations for us. Any node can use the `tf2` libraries to broadcast a transform from one frame to another.
- Transforms will need to form a tree structure, where each frame is defined by one (and only one) transform from another frame, but can have any number of frames dependent on it.



Transforms in RVIZ

4 ROS Components

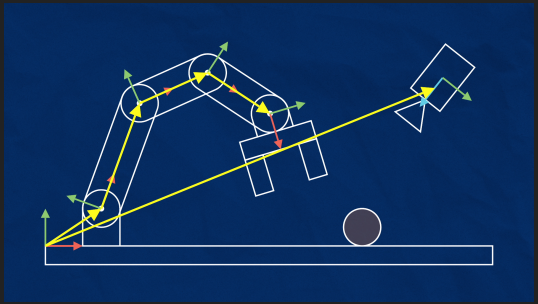
- In ROS 2, RViz (the ROS visualization tool) is called `rviz2`, which can display all kinds of different data. To display TF data, we click the ”Add” button in the bottom-left corner, and select ”TF”.
- We can also configure the data displayed in RViz using the left panel.



Broadcasting a Moving Robot

4 ROS Components

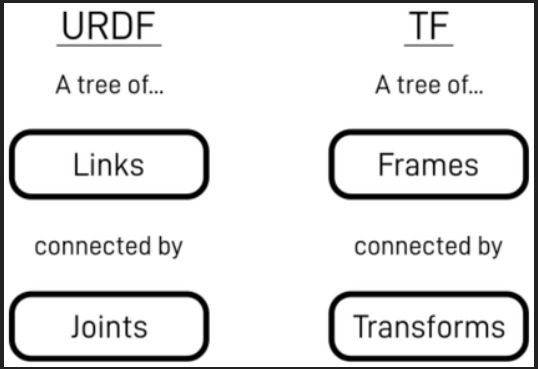
- The first step in doing this is to make sure we have a URDF file for our robot, which is a kind of configuration file where we can specify all sorts of physical characteristics of the robot's components, such as their size, shape, colour, and more.



URDF (Unified Robot Description Format)

4 ROS Components

- In URDF, a robot is made up of a series of rigid components called links, and the links are joined together in a parent-child tree structure, where the relationships between different links are called joints.



URDF (Unified Robot Description Format)

4 ROS Components

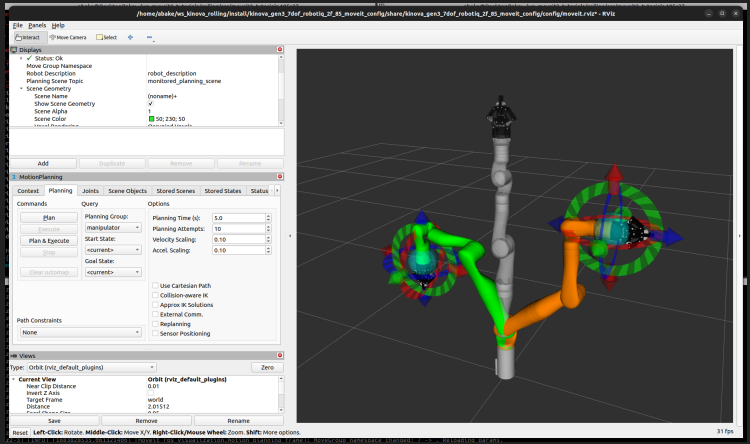
- The `/robot_state_publisher` which can take in a URDF file and automatically broadcast all the transforms from it. It will also publish the full contents of the URDF file to the topic `/robot_description` so that any other nodes that need it can all use the same file. URDF files are written in XML.

```
<robot xmlns:xacro="http://www.ros.org/wiki/xacro"  name="robot">

    <xacro:arg name="sim_mode" default="false"/>

    <xacro:include filename="robot_core.xacro" />

    <xacro:if value="$(arg use_ros2_control)">
        <xacro:include filename="ros2_control.xacro" />
    </xacro:if>
    <xacro:unless value="$(arg use_ros2_control)">
        <xacro:include filename="gazebo_control.xacro" />
    </xacro:unless>
    <xacro:include filename="lidar.xacro" />
    <xacro:include filename="camera.xacro" />

</robot
```

RViz: Visualization Tool

4 ROS Components

- RViz is a 3D visualizer for the Robot Operating System (ROS) framework.



Gazebo: Physics Simulation

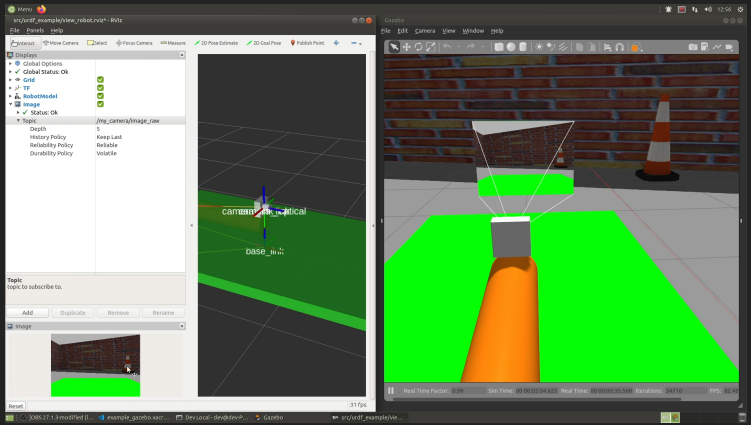
4 ROS Components

- Gazebo is a free, open-source robot simulation environment maintained by Open Robotics. With Gazebo, we can create a virtual ”world”, and load simulated versions of our robots into it. Simulated sensors can detect the environment, and publish the data to the same ROS topics that real sensors would, allowing easy testing of algorithms. Then, forces can be applied to the simulated actuators on the robot, taking into account things like friction.

Gazebo to Hardware Transition

4 ROS Components

- Simulated sensor and actuator data mirrors real-world hardware.

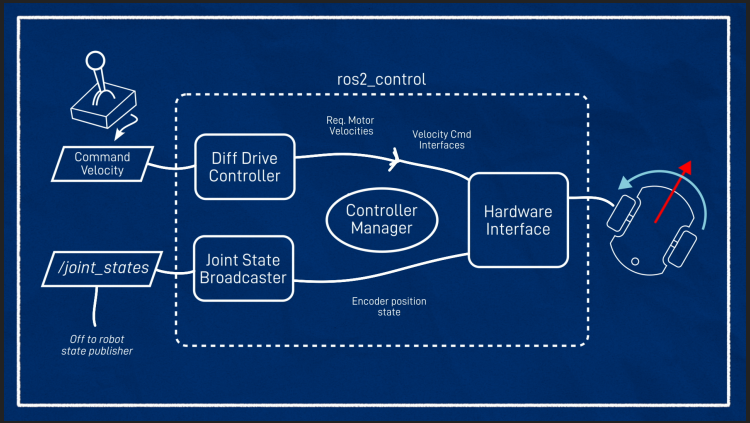


ros2_control: Hardware Abstraction Layer

- `ros2_control` in layman's terms:
  - is the kernel for your robot.
  - provides a way to standardize hardware interaction.
  - is a framework to provide real time guarantees on interaction between hardware drivers and controllers.
- Bonus: It also provides a way to switch easily between hardware and simulation.

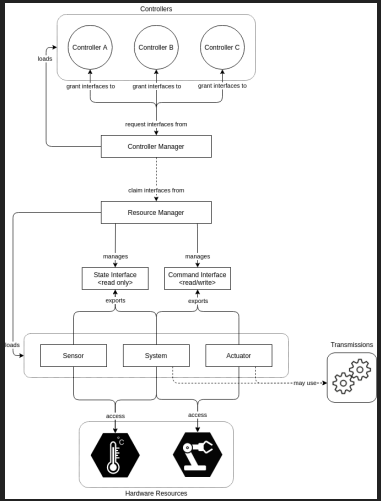
ros2_control: Basic Architecture

4 ROS Components



ros2_control: Slightly In-Depth Architecture

4 ROS Components



Teleoperation

4 ROS Components

- Manual control interface: keyboard or joystick

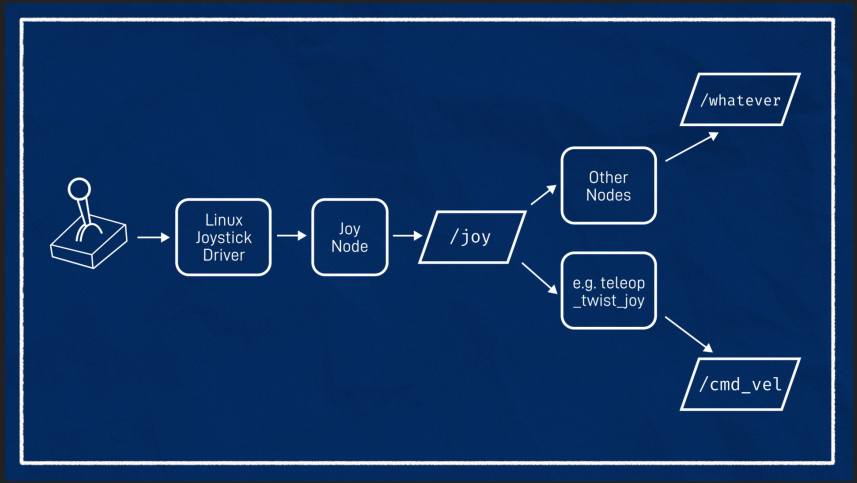
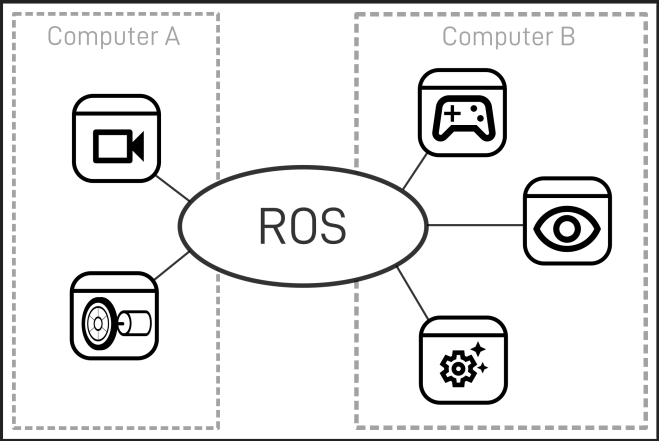


Table of Contents

Nodes

5 Basic ROS Concepts

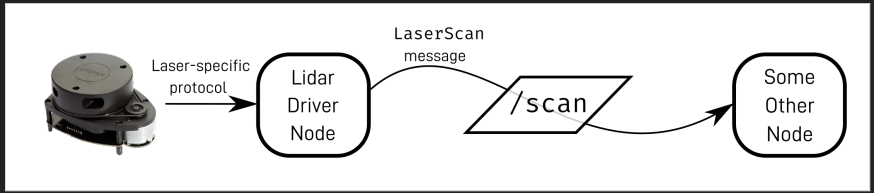
- A ROS system is made up of a bunch of smaller programs that all run at once and talk to each other. Each of these programs is called a node.



Topics and Messages

5 Basic ROS Concepts

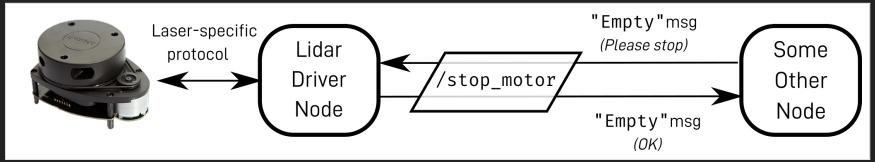
- The way ROS nodes communicate with each other is with topics and messages. A topic is a named location that one (or occasionally multiple) nodes can publish a message to. These nodes are called publishers, and other nodes (called subscribers) can subscribe to the topic to receive them.



Services

5 Basic ROS Concepts

- Unlike topics, which allow a node to share many messages to anyone who cares to listen, services offer a single request/reply communication from one node to another. The content of the reply may be useful information, such as the result of a calculation, or it may just be an indication that the request has been received.



Launch Files

5 Basic ROS Concepts

Transforms in ROS

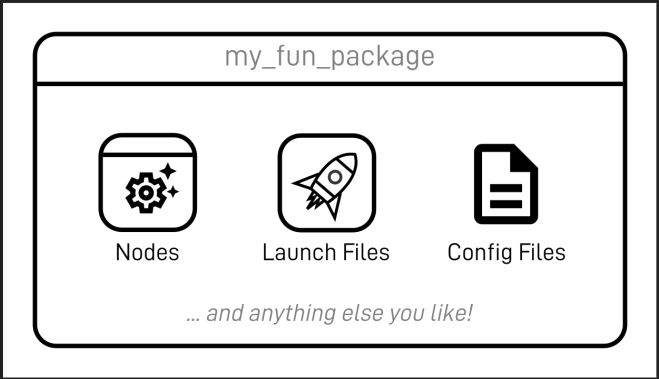
- When we're working on a ROS project we will often be running the same nodes over and over again with the same parameters and remappings. This becomes pretty tedious as you need to remember to open all the terminals and get all the parameters right, then stop them all when you're done.

```python
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='rplidar_ros',
            executable='rplidar_node',
            parameters=[{
                'serial_port': '/dev/ttyUSB0'
            }],
            remappings=[
                ('/scan', '/scan_1')
            ]
        )
    ])
```

ROS Packages

5 Basic ROS Concepts

- Packages are the basic unit of organization in ROS, grouping related nodes, configuration files, models, or libraries together for reuse and modular development.



ROS Workspaces

5 Basic ROS Concepts

- Packages are the basic unit of organization in ROS, grouping related nodes, configuration files, models, or libraries together for reuse and modular development.

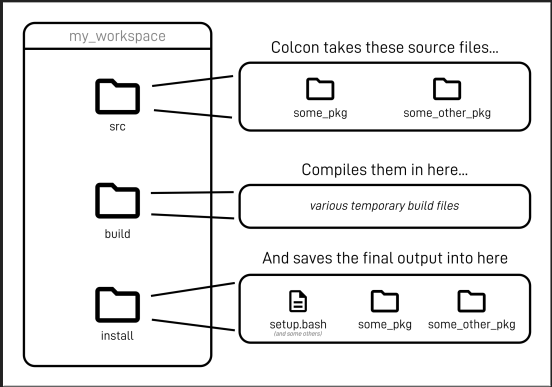


Table of Contents

6 Extending to Other Robotic Systems

From Ground Robots to Manipulators & Drones

6 Extending to Other Robotic Systems

- Many concepts translate directly:
  - URDF for robot description (e.g., robotic arm kinematics)
  - `ros2_control` for joint controllers (arm motors, drone ESCs)
  - `nav2`/motion planning libraries for manipulators (MoveIt! integration)
  - Simulation in Gazebo with appropriate plugins (e.g., aerial dynamics)
- Show real-world examples:
  - Manipulator pick-and-place using MoveIt! + `ros2_control`
  - Drone waypoint navigation using MAVROS + custom controllers

Table of Contents

7 Hands-On: Environment Setup

Why Docker & Devcontainers?

7 Hands-On: Environment Setup

- Ensuring **repeatable, sustainable development** environments
- Isolation of dependencies (ROS 2 versions, Gazebo, libraries)
- Consistent experience across different machines
- Ease of collaboration (share Devcontainer configurations)

Devcontainer Configuration Example

- `.devcontainer/devcontainer.json`:
  - Base image: `ros:foxy-desktop`
  - VSCode extensions: `ms-vscode.cpptools`, `ms-iot.vscode-ros`
  - Forwarded ports, mounted volumes
- `Dockerfile`:
  - Install Gazebo, additional ROS packages, system dependencies

Table of Contents

Problem Setup

8 Assignment: Teleop to Hardware

- Clone the provided repository:
- `https://github.com/soham2560/MecaBot`
- `https://github.com/rtarun1/HepoBot`
- Pre-existing components:
  - Controller implementation is already provided.
  - Focus on understanding the velocity interface.
- Tasks:
  - Understand the velocity interface and publish to it
  - Observe and validate results in simulation and on real hardware.
  - Implement Joy interface for manual control.
  - Implement teleoperation (teleop) for both simulation and real hardware testing.
- Submission criteria:
  - Working velocity interface demo in simulation
  - Successful hardware test with teleop
  - Completion of Joy control integration.

Table of Contents

Conclusion & Next Steps

- How to continue beyond this lecture:
  - Explore MoveIt! for manipulation
  - Integrate perception pipelines (OpenCV, PCL)
  - Multi-robot systems and ROS 2's DDS advantages
- Q&A and feedback

Questions?

9 Conclusion & Next Steps

Basics of ROS

Thank you