EE5111/EE5061

Introduction to Robot Operating System 2 (ROS 2)

Dr Feng LIN

Email: feng_lin@nus.edu.sg

Electrical and Computer Engineering

Outline

- Introduction
- Simulation Experiment

Learning Objectives

- Understand ROS and ROS 2 fundamentals
- Learn to create, build, and run ROS 2 packages
- Explore Nodes, Topics, Messages, and Services
- Develop Publisher–Subscriber example
- Experiment with Turtlesim & rqt
- Implement cooperative control with two turtles
- Install RX-150 Robotic Arm System (optional)

About ROS

- ROS is an open-source meta-operating system providing:
- Hardware abstraction and device control
- Message passing between processes
- Visualization and simulation tools
- Supported by major robot manufacturers (ABB, KUKA) and software (MATLAB, LabVIEW).

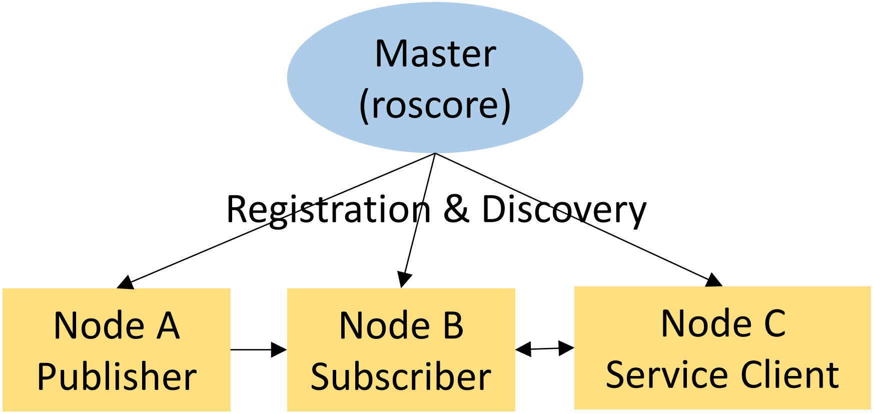
https://docs.clearpathrobotics.com/docs_robots/outdoor_robots/husky/a200/user_manual_husky/

https://github.com/Tesollo-Delto/DELTO_B_ROS2

ROS 1 vs ROS 2

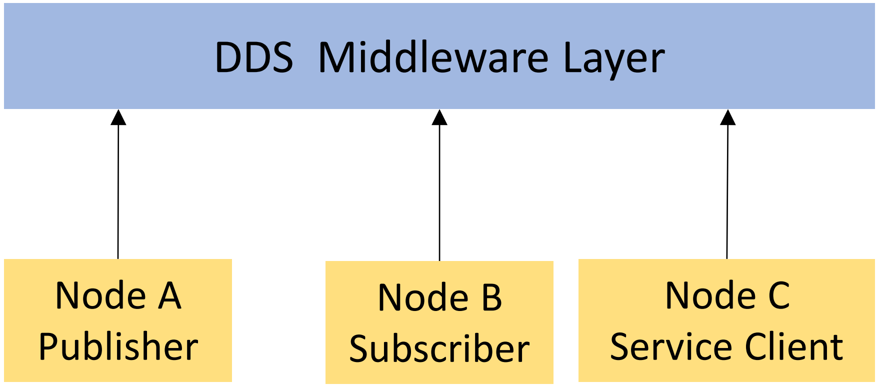
- ROS 1: Centralized master node, non–real-time
- ROS 2: DDS-based decentralized communication, real-time capable
- Supports C++, Python, Java, and Rust
- Improved scalability and long-term support.

ROS 1: Centralized (requires roscore)



Data exchange (TCPROS/UDPROS)

ROS2: Decentralized (no master, DDS-based)



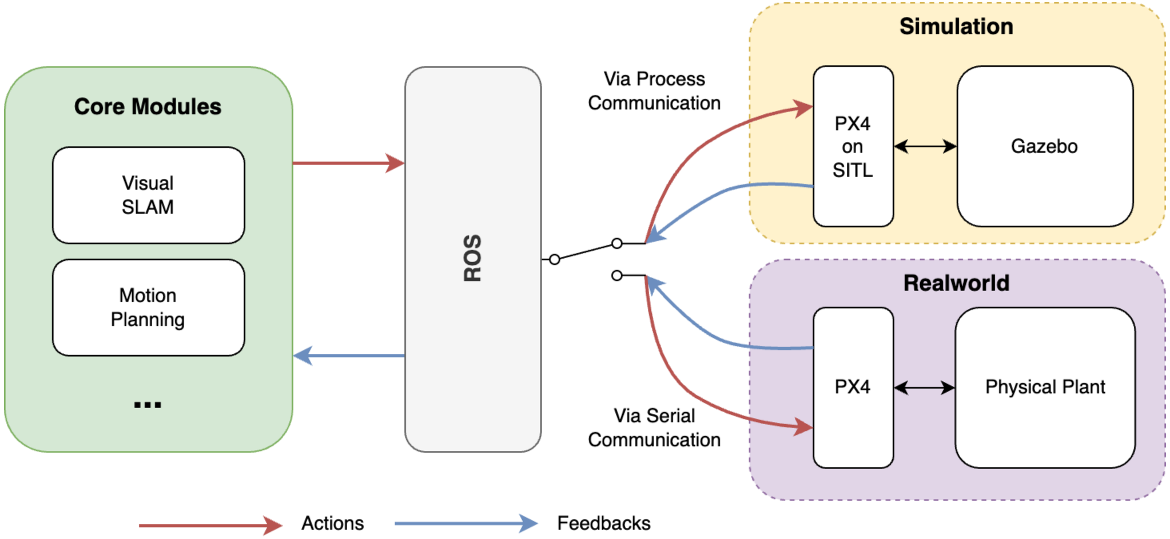
Automatic Peer Discovery & QoS Communication

Main Features of ROS2

- **Tools**: ROS 2 offers a suite of developer tools, which cover a wide range of functionalities and accelerate application development. The tools include launch, introspection, debugging, visualization, plotting, logging, and playback.
- **Capabilities**: ROS 2 provides advanced features and better supports for the realtime computing.
- **Data Distribution Service (DDS)**: ROS 2 facilitates communication between nodes via ROS Middleware Interface, which provide real-time communication, scalability, performance enhancement, and security benefits via Data Distribution Service .
- **Ecosystem**: The ROS 2 ecosystem is supported by a community of developers, researchers, and industry professionals. The community-driven effort ensures continuous improvement and innovation.

Objectives of ROS 2

1. Modularity & Reusability – component-based development
2. Real-time Performance – DDS middleware
3. Scalability – from small robots to multi-agent systems



ROS 2 File System Structure

ROS 2 uses a hierarchical file structure to organize robot-related resources efficiently. It is designed to be **intuitive**, **scalable**, and suitable for various robot configurations.

Directory	**Purpose / Description**
Root (ws)	Top-level workspace for ROS 2 projects (customizable).
Source (src)	Contains source code of ROS 2 packages — each as a subfolder.
Build (build)	Stores intermediate build files generated during compilation.
Install (install)	Holds installed executable files after running colcon build.
Log (log)	Stores log files from ROS 2 node executions for debugging.

File System Tools and Commands

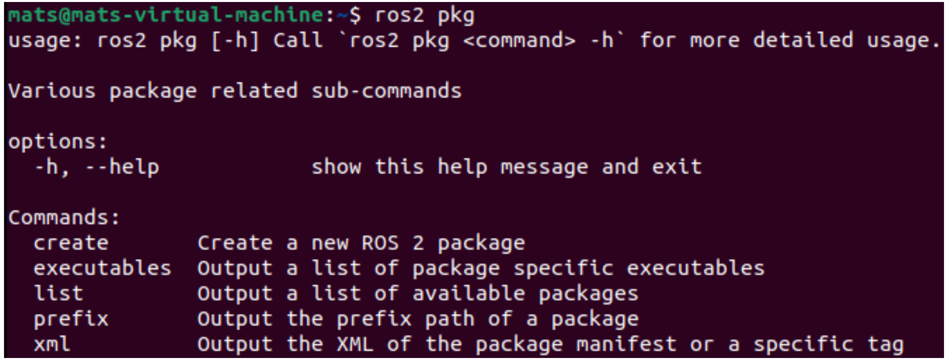
ROS 2 organizes code via **packages**, essential units for sharing and reusing code across ROS 2 systems. Each package can include nodes, launch files, configuration, and other resources.

Component	**Description**
Packages	Basic units of organization for ROS 2 code. Required for installation and sharing.
Build System: ament	Used for building ROS 2 packages (replaces ROS 1's catkin).
Build Tool: colcon	Automates package building and workspace management.
Supported Languages	Packages can be created with **CMake** (C++) or **Python**.

File System Tools and Commands

Useful Command: `ros2 pkg`

```
$ ros2 pkg [command] -h
```



```
mats@mats-virtual-machine:~$ ros2 pkg
usage: ros2 pkg [-h] Call `ros2 pkg <command> -h` for more detailed usage.

Various package related sub-commands

options:
  -h, --help            show this help message and exit

Commands:
  create       Create a new ROS 2 package
  executables  Output a list of package specific executables
  list         Output a list of available packages
  prefix       Output the prefix path of a package
  xml          Output the XML of the package manifest or a specific tag
```

Sub-commands:

- `create` – make a new package
- `executables` – list executables
- `list` – show all available packages
- `prefix` – show install path
- `xml` – view package manifest

💡 *Use `ros2 pkg <command> -h` to see detailed help.*

Package Creation and Build Process

Install Colcon Extensions

```
$ sudo apt install python3-colcon-common-extensions
```

Initialize a Workspace

```
$ mkdir -p ~/<WSName>_ws/src
$ cd ~/<WSName>_ws
$ colcon build
```

Create Your First Package

```
$ cd ~/<WSName>_ws/src
$ ros2 pkg create --build-type ament_cmake <PKGName>
```

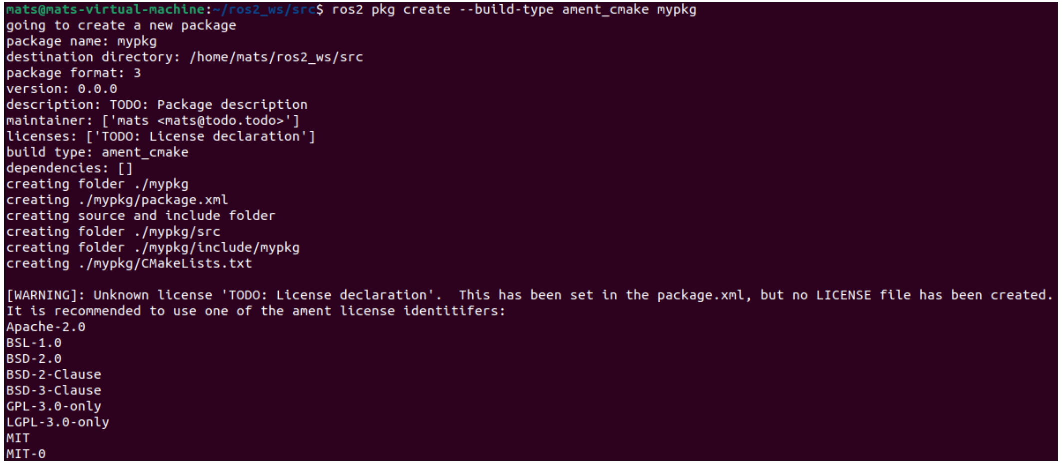
Package Creation and Build Process

Build a Specific Package

```
$  cd ~/<WSName>_ws
$  colcon build --packages-select <PKGName>
```

Build-type examples:

- `ament_cmake` for C++
- `ament_python` for Python



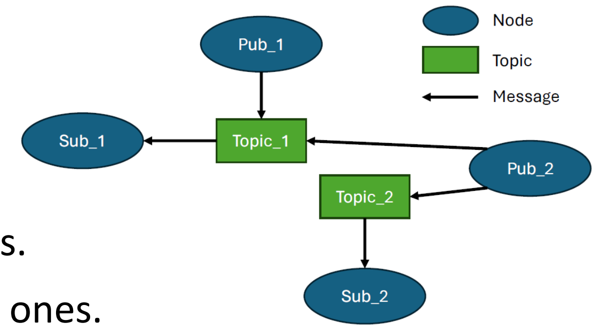
```
mats@mats-virtual-machine:~/ros2_ws/src$ ros2 pkg create --build-type ament_cmake mypkg
going to create a new package
package name: mypkg
destination directory: /home/mats/ros2_ws/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['mats <mats@todo.todo>']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: []
creating folder ./mypkg
creating ./mypkg/package.xml
creating source and include folder
creating folder ./mypkg/src
creating folder ./mypkg/include/mypkg
creating ./mypkg/CMakeLists.txt

[WARNING]: Unknown license 'TODO: License declaration'.  This has been set in the package.xml, but no LICENSE file has been created.
It is recommended to use one of the ament license identitifers:
Apache-2.0
BSL-1.0
BSD-2.0
BSD-2-Clause
BSD-3-Clause
GPL-3.0-only
LGPL-3.0-only
MIT
MIT-0
```

Important Concepts in ROS 2

This section introduces key ROS 2 terms and ideas.
They form the foundation of how data and control flow through a ROS 2 system.

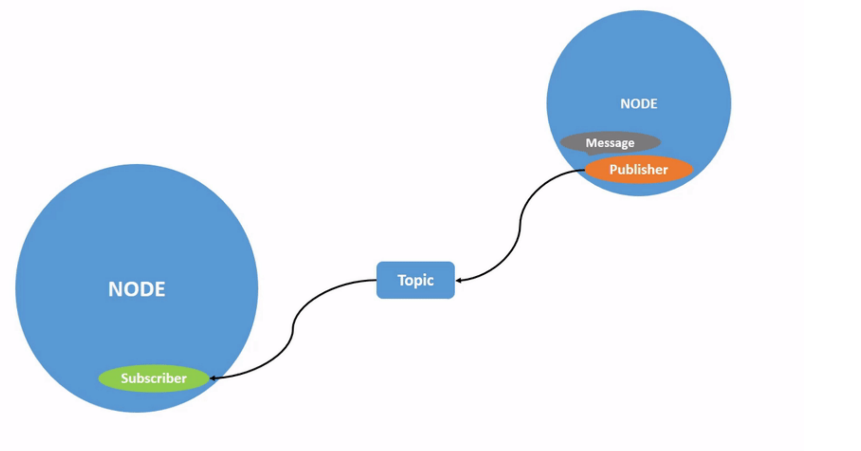
Concept	**Description**
Package	The basic unit in ROS 2, containing nodes, launch/config files, dependencies, and datasets.
Node	A single-purpose process in ROS 2 that sends or receives data via topics, services, or actions.
Topic	A named communication channel used for message passing between nodes. Nodes can publish or subscribe to topics.

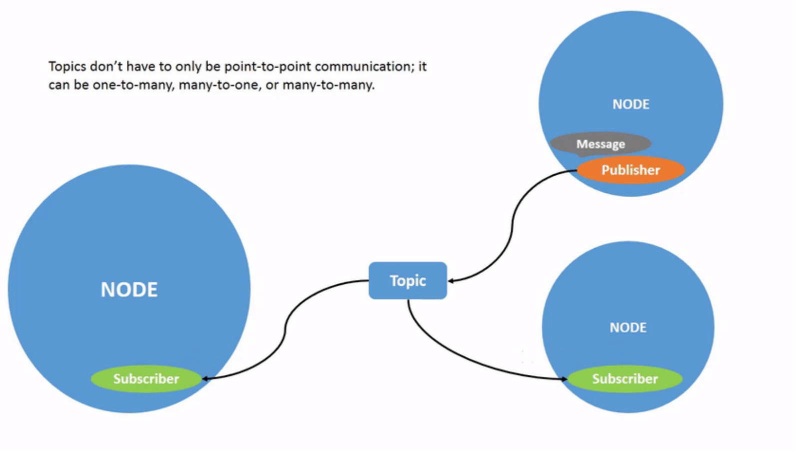
Important Concepts in ROS 2



ROS 2 Topics

- Enable message exchange between publishers and subscribers.
- A node can publish to multiple topics or subscribe to multiple ones.
- Topics are **many-to-many**: multiple publishers and subscribers can share the same topic.




Topics don’t have to only be point-to-point communication; it can be one-to-many, many-to-one, or many-to-many.

https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Topics/Understanding-ROS2-Topics.html

Example: Coding a Publisher and Subscriber

System Setup

- **OS:** Ubuntu 22.04.5 LTS (Jammy Jellyfish)
- **ROS 2 Version:** Humble Hawksbill

Ensure the correct OS + ROS 2 version to avoid build errors

Step 1 – Create a Package

```
$  ros2 pkg create --build-type ament_cmake --license Apache-2.0 cpp_pubsub
```

Step 2 – Download Sample Publisher File

Inside the ‘src’ folder of the newly created package (NOT the workspace), call the following commands to copy the official ROS 2 publisher file:

```
$   wget -O publisher_member_function.cpp
    https://raw.githubusercontent.com/ros2/examples/humble/rclcpp/topics/minimal_publisher/member_function.cpp
```

Example: Coding a Publisher and Subscriber

```
#include "rclcpp/rclcpp.hpp"
class MinimalPublisher : public rclcpp::Node
{
public:
  MinimalPublisher()
  : Node("minimal_publisher"), count_(0)
  {
    publisher_ = this->create_publisher<std_msgs::msg::String>("topic", 10);
    timer_ = this->create_wall_timer(
      500ms, std::bind(&MinimalPublisher::timer_callback, this));
  }
private:
  void timer_callback()
  {
    auto message = std_msgs::msg::String();
    message.data = "Hello, world! " + std::to_string(count_++);
    RCLCPP_INFO(this->get_logger(), "Publishing: '%s'", message.data.c_str());
    publisher_->publish(message);
  }
  rclcpp::Publisher<std_msgs::msg::String>::SharedPtr publisher_;
  rclcpp::TimerBase::SharedPtr timer_;
  size_t count_;
};
```

Example: Coding a Publisher and Subscriber

Main Function

```cpp
int main(int argc, char * argv[])
{
  rclcpp::init(argc, argv);
  rclcpp::spin(std::make_shared<MinimalPublisher>());
  rclcpp::shutdown();
  return 0;
}
```

Adding Dependencies and Build Setup

Edit package.xml

```
<buildtool_depend>ament_cmake</buildtool_depend>
<depend>rclcpp</depend>
<depend>std_msgs</depend>
<test_depend>ament_lint_auto</test_depend>
<test_depend>ament_lint_common</test_depend>
```

Edit CMakeLists.txt:

```
# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)
add_executable(talker src/publisher_member_function.cpp)
ament_target_dependencies(talker rclcpp std_msgs)
install(TARGETS talker DESTINATION lib/${PROJECT_NAME})
```

Subscriber Setup

Purpose:
To set up the subscriber node corresponding to the previously created publisher.

Download the Sample Subscriber File

```
$   wget -O subscriber_member_function.cpp
    https://raw.githubusercontent.com/ros2/examples/humble/rclcpp/topics/minimal_subscriber/member_function.cpp
```

Understand the Subscriber Node

Structure is **similar** to the publisher.

Key Difference:

> No timer is used — the subscriber waits to **receive** messages instead of publishing periodically.

The node runs by spinning the MinimalSubscriber class, which listens for messages on a given topic.

➢ *Subscriber nodes are event-driven, reacting only when a message arrives.*

Final CMakeLists Configuration

Final version of CMakeLists.txt:

```
# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(std_msgs REQUIRED)

add_executable(talker src/publisher_member_function.cpp)
ament_target_dependencies(talker rclcpp std_msgs)
install(TARGETS talker DESTINATION lib/${PROJECT_NAME})

add_executable(listener src/subscriber_member_function.cpp)
ament_target_dependencies(listener rclcpp std_msgs)
install(TARGETS talker listener DESTINATION lib/${PROJECT_NAME})
```

Build and Run the ROS 2 Package

Check and Install Dependencies

```
$  rosdep install -i --from-path src --rosdistro humble –y
```

Build the Package

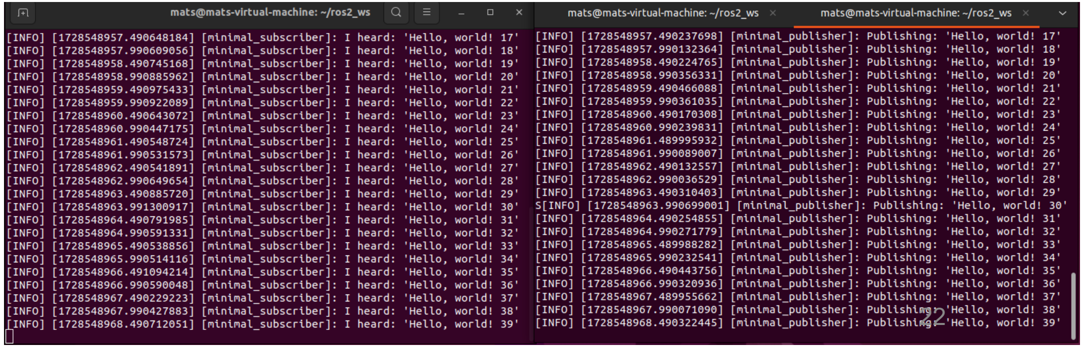
If all dependencies are installed, in the root of your workspace, build the package.

```
$  colcon build --packages-select cpp_pubsub
```

Source the Workspace

Open two new terminals and navigate both to ‘ros2_ws’. For both terminals, source the setup files.

```
$  . install/setup.bash
```


mats@mats-virtual-machine: ~/ros2_ws
mats@mats-virtual-machine: ~/ros2_ws
mats@mats-virtual-machine: ~/ros2_ws

```
[INFO] [1728548957.490648184] [minimal_subscriber]: I heard: 'Hello, world! 17'
[INFO] [1728548957.990609056] [minimal_subscriber]: I heard: 'Hello, world! 18'
[INFO] [1728548958.490745168] [minimal_subscriber]: I heard: 'Hello, world! 19'
[INFO] [1728548958.990885962] [minimal_subscriber]: I heard: 'Hello, world! 20'
[INFO] [1728548959.490975433] [minimal_subscriber]: I heard: 'Hello, world! 21'
[INFO] [1728548959.990922089] [minimal_subscriber]: I heard: 'Hello, world! 22'
[INFO] [1728548960.490643072] [minimal_subscriber]: I heard: 'Hello, world! 23'
[INFO] [1728548960.990447175] [minimal_subscriber]: I heard: 'Hello, world! 24'
[INFO] [1728548961.490548724] [minimal_subscriber]: I heard: 'Hello, world! 25'
[INFO] [1728548961.990531573] [minimal_subscriber]: I heard: 'Hello, world! 26'
[INFO] [1728548962.490541891] [minimal_subscriber]: I heard: 'Hello, world! 27'
[INFO] [1728548962.990649654] [minimal_subscriber]: I heard: 'Hello, world! 28'
[INFO] [1728548963.490885720] [minimal_subscriber]: I heard: 'Hello, world! 29'
[INFO] [1728548963.991300917] [minimal_subscriber]: I heard: 'Hello, world! 30'
[INFO] [1728548964.490791985] [minimal_subscriber]: I heard: 'Hello, world! 31'
[INFO] [1728548964.990591331] [minimal_subscriber]: I heard: 'Hello, world! 32'
[INFO] [1728548965.490538856] [minimal_subscriber]: I heard: 'Hello, world! 33'
[INFO] [1728548965.990514116] [minimal_subscriber]: I heard: 'Hello, world! 34'
[INFO] [1728548966.491094214] [minimal_subscriber]: I heard: 'Hello, world! 35'
[INFO] [1728548966.990590048] [minimal_subscriber]: I heard: 'Hello, world! 36'
[INFO] [1728548967.490229223] [minimal_subscriber]: I heard: 'Hello, world! 37'
[INFO] [1728548967.990427883] [minimal_subscriber]: I heard: 'Hello, world! 38'
[INFO] [1728548968.490712051] [minimal_subscriber]: I heard: 'Hello, world! 39'
```

```
[INFO] [1728548957.490237698] [minimal_publisher]: Publishing: 'Hello, world! 17'
[INFO] [1728548957.990132364] [minimal_publisher]: Publishing: 'Hello, world! 18'
[INFO] [1728548958.490224765] [minimal_publisher]: Publishing: 'Hello, world! 19'
[INFO] [1728548958.990356331] [minimal_publisher]: Publishing: 'Hello, world! 20'
[INFO] [1728548959.490466088] [minimal_publisher]: Publishing: 'Hello, world! 21'
[INFO] [1728548959.990361035] [minimal_publisher]: Publishing: 'Hello, world! 22'
[INFO] [1728548960.490170308] [minimal_publisher]: Publishing: 'Hello, world! 23'
[INFO] [1728548960.990239831] [minimal_publisher]: Publishing: 'Hello, world! 24'
[INFO] [1728548961.489995932] [minimal_publisher]: Publishing: 'Hello, world! 25'
[INFO] [1728548961.990089007] [minimal_publisher]: Publishing: 'Hello, world! 26'
[INFO] [1728548962.490132557] [minimal_publisher]: Publishing: 'Hello, world! 27'
[INFO] [1728548962.990036529] [minimal_publisher]: Publishing: 'Hello, world! 28'
[INFO] [1728548963.490310403] [minimal_publisher]: Publishing: 'Hello, world! 29'
S[INFO] [1728548963.990699001] [minimal_publisher]: Publishing: 'Hello, world! 30'
[INFO] [1728548964.490254855] [minimal_publisher]: Publishing: 'Hello, world! 31'
[INFO] [1728548964.990271779] [minimal_publisher]: Publishing: 'Hello, world! 32'
[INFO] [1728548965.489988282] [minimal_publisher]: Publishing: 'Hello, world! 33'
[INFO] [1728548965.990232541] [minimal_publisher]: Publishing: 'Hello, world! 34'
[INFO] [1728548966.490443756] [minimal_publisher]: Publishing: 'Hello, world! 35'
[INFO] [1728548966.990320936] [minimal_publisher]: Publishing: 'Hello, world! 36'
[INFO] [1728548967.489955662] [minimal_publisher]: Publishing: 'Hello, world! 37'
[INFO] [1728548967.990071090] [minimal_publisher]: Publishing: 'Hello, world! 38'
[INFO] [1728548968.490322445] [minimal_publisher]: Publishing: 'Hello, world! 39'
```

Run Publisher and Subscriber

Run the Publisher

```
$  ros2 run cpp_pubsub talker
```

Run the Subscriber

```
$  ros2 run cpp_pubsub listener
```

ROS Communication via Services

Concept Overview

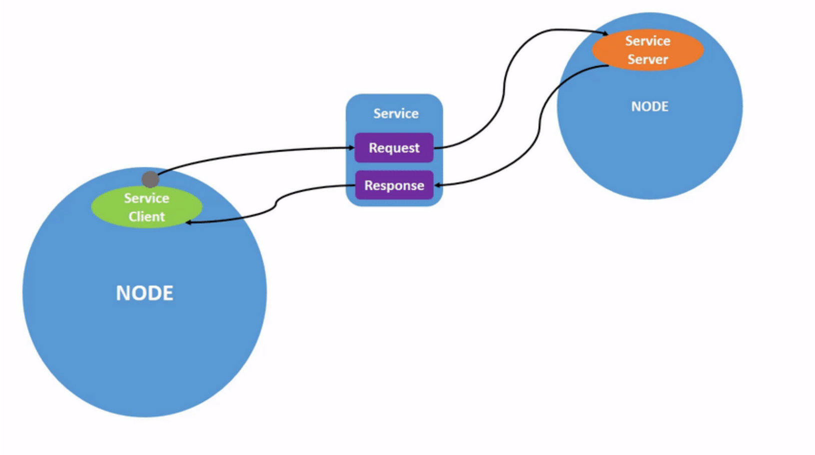
- **Services** provide a **call-and-response** communication model in ROS.
- Unlike **topics** (which continuously publish/subscribe), services are **invoked only when needed**.
- Used for **one-time queries or commands** where a response is required.

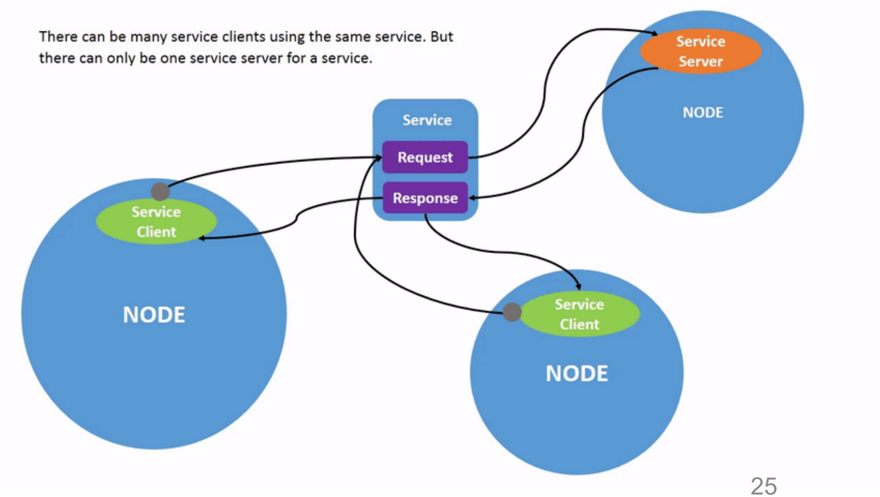
Service Architecture

- **Service Client**: Sends a **Request** message.
- **Service Server**: Processes the request and returns a **Response** message.
- Communication is **synchronous** — the client waits until the server replies.

ROS Communication via Services

Visual Diagram:




There can be many service clients using the same service. But there can only be one service server for a service.

Playing with Turtlesim

What is Turtlesim

- **Turtlesim** is a simple simulator included with **ROS 2**, often used for learning key ROS concepts such as:

Concept	**Description**
Node	The turtle acts as a node that communicates with other nodes using ROS messages.
Topic	Channels for data exchange (e.g., /turtle1/cmd_vel to control velocity).
Service	Request–response interaction (e.g., /spawn, /reset to create or reset turtles).
Message	Data format used in communication (e.g., Twist messages for movement).
Parameter	Stores configuration values such as window color or simulation settings.

- It provides a **2D virtual turtle** that can move around using ROS 2 commands.
- Ideal for **beginners** to practice robotics concepts without real hardware.

Install Turtlesim

Source ROS 2

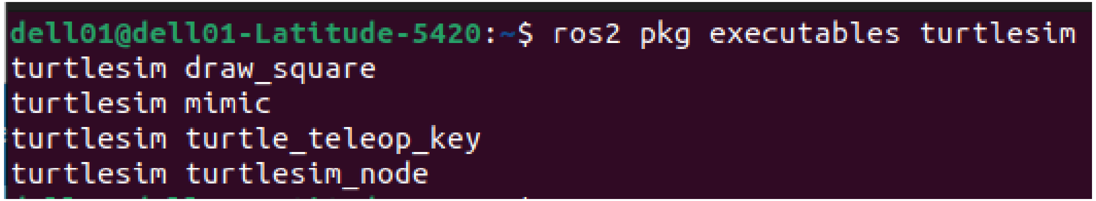
Always remember to **source ROS 2** in each new terminal before running any commands.

Install the Turtlesim Package

```
$  sudo apt update
$  sudo apt install ros-humble-turtlesim
```

Verify the Installation

```
$  ros2 pkg executables turtlesim
```



```
dell01@dell01-Latitude-5420:~$ ros2 pkg executables turtlesim
turtlesim draw_square
turtlesim mimic
turtlesim turtle_teleop_key
turtlesim turtlesim_node
```

Using Turtlesim

Start the Turtlesim Node

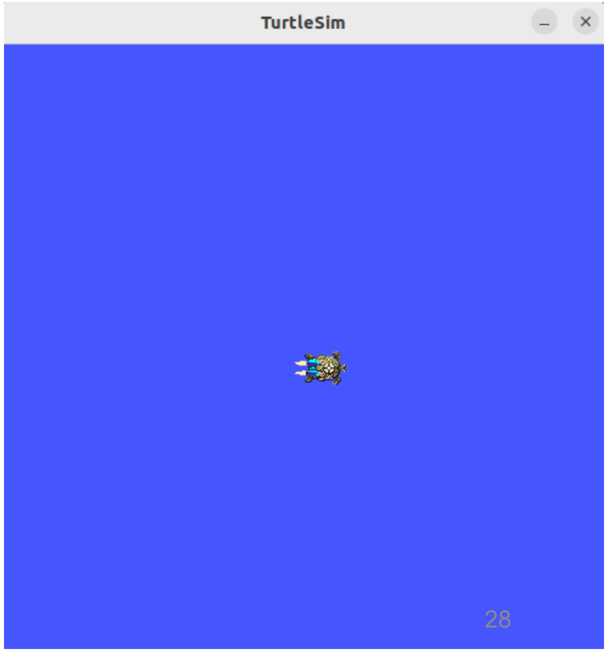
```
$  ros2 run turtlesim turtlesim_node
```

Control the Turtle

Open a new terminal, and run the teleoperation node:

```
$  ros2 run turtlesim turtle_teleop_key
```

- Use the arrow keys on your keyboard to control the turtle’s movement.
- Each node runs independently: the turtlesim_node displays the environment, while turtle_teleop_key sends velocity commands via ROS 2 topics.



Introducing rqt

What is rqt

- **rqt** is a graphical user interface (GUI) framework for **ROS 2**.
- It provides tools and plugins to **visualize** and **manage** ROS-based systems.
- Built using **Python** and **Qt**, it helps users interact with:
  - ROS **nodes**
  - **Topics**
  - **Services**
  - **Parameters**
    — all in a **visual** and **interactive** way.

Install and Run rqt

Install rqt and plugins

```
$  sudo apt update
$  sudo apt install '~nros-humble-rqt*'
```

Run rqt

```
$  rqt
```

Once launched, rqt allows you to explore nodes, topics, and parameters graphically.

Using rqt

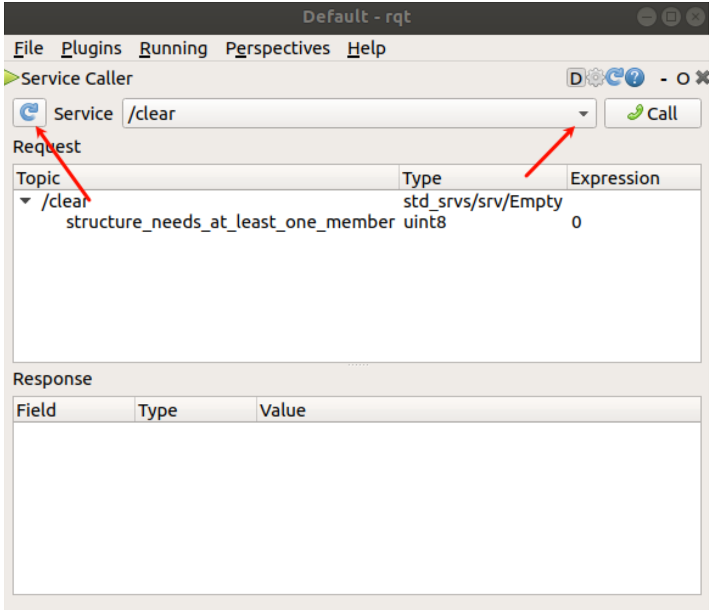
Accessing the Service Caller

1. Open **rqt**.
2. From the menu bar, select:

Plugins → Services → Service Caller

3. Click the **refresh** button (next to *Service*) to load all available ROS services.
4. Open the **Service** dropdown to view Turtlesim's available services.

The Service Caller interface lets you send requests directly to ROS 2 services.



Service /clear

Topic	Type	Expression
▾ /clear	std_srvs/srv/Empty	
structure_needs_at_least_one_member	uint8	0

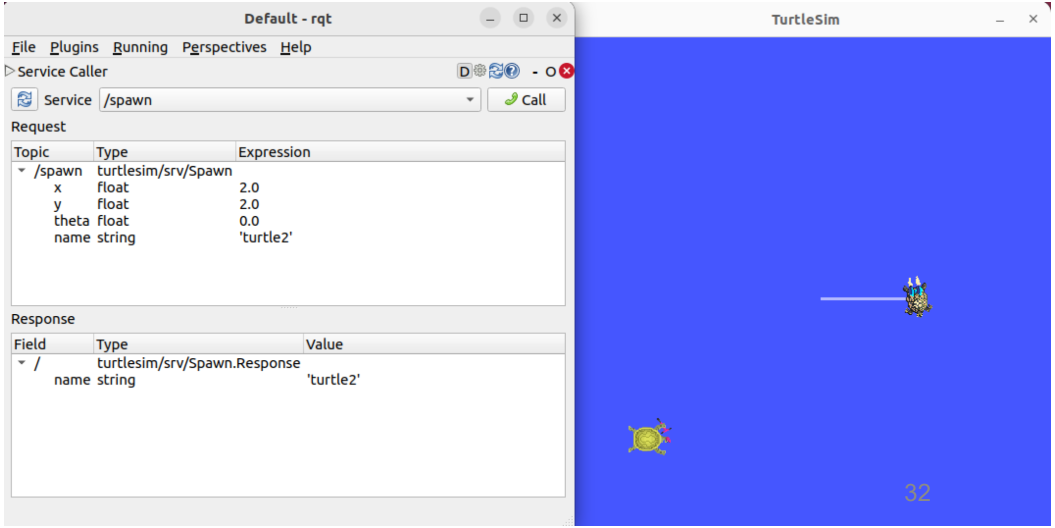
Field	Type	Value

Using rqt

Example: Using `/spawn` Service

- Select `/spawn` from the **Service** dropdown list.
- Enter:
  - **Name:** new turtle name (e.g., `turtle2`)
  - **Coordinates:** valid `(x, y)` values (e.g., `2, 2`)
- Click **Call** to spawn the new turtle.

If successful, a **new turtle** appears in the simulator window.



Service /spawn Call

Request

Topic	Type	Expression
/spawn	turtlesim/srv/Spawn	
x	float	2.0
y	float	2.0
theta	float	0.0
name	string	'turtle2'

Response

Field	Type	Value
/	turtlesim/srv/Spawn.Response	
name	string	'turtle2'

Using rqt

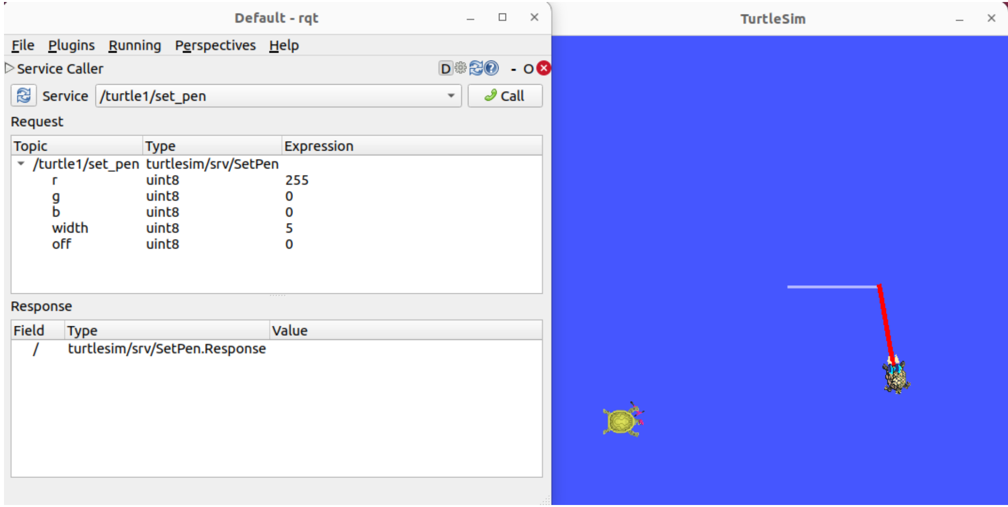
Try the `/set_pen` Service

1. In the **Service** dropdown, select:

```
/turtle1/set_pen
```

2. Set parameters for the pen:
   - r, g, b (0 – 255) → Pen color
   - width → Line thickness
   - off → Enable / disable drawing

3. Click Call.
   - Return to the terminal running turtle_teleop_key.
   - Move the turtle using arrow keys.



Service: /turtle1/set_pen

Request

Topic	Type	Expression
/turtle1/set_pen	turtlesim/srv/SetPen	
r	uint8	255
g	uint8	0
b	uint8	0
width	uint8	5
off	uint8	0

Response

Field	Type	Value
/	turtlesim/srv/SetPen.Response	

- If successful, turtle 1's trail color & line thickness will update.
- *The `/set_pen` service customizes your turtle's drawing style interactively*

Using rqt

Remapping and Controlling Multiple Turtles

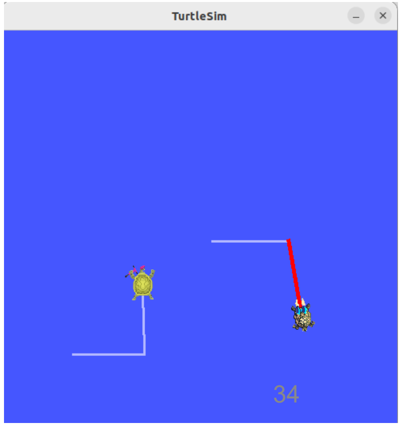
Why Remapping?

- Running another `turtle_teleop_key` normally still controls `turtle1`.
- You must **remap the topic** to control `turtle2`.

Remap the cmd_vel Topic

```
$   ros2 run turtlesim turtle_teleop_key --ros-args --remap turtle1/cmd_vel:=turtle2/cmd_vel
```

Now turtle2 can be controlled independently in this terminal.

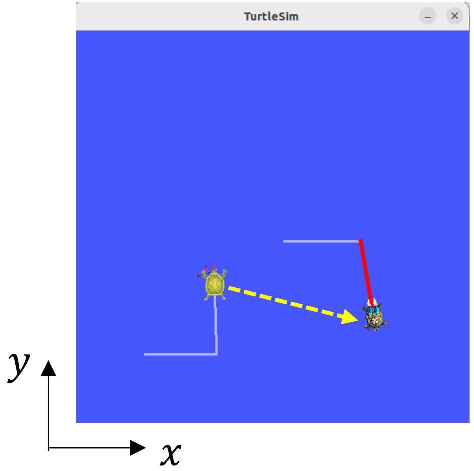


Close Turtlesim

Stop simulation:

- `Ctrl + C` in `turtlesim_node` terminal
- Type `q` in `turtle_teleop_key` terminals

Additional Task: Cooperative Motion



Overview

You've learned the basics of:

- ROS 2 Topics
- Turtlesim
- rqt

Now, let's combine them in an advanced task: **Cooperative motion of two turtles in Turtlesim.**

System Concept

- **Leader Turtle:** controlled manually using

```
$  ros2 turtle_teleop_key
```

- **Follower Turtle:** automatically tracks and follows the leader's position and direction.

This task demonstrates ROS 2's publish–subscribe mechanism for multi-node interaction.

Implementation Steps

Step 1 – Create the Workspace and Package

Let’s use the workspace you created in the previous tutorial, <WSName>_ws, for a new package turtle_coop. Make sure you are in the src folder before running the package creation command.

```
$  cd ~/<WSName>_ws/src
$  ros2 pkg create --build-type ament_cmake --license Apache-2.0 turtle_coop
```

Step 2 – Add Files

Place the following in your package:

- `turtle_coop.cpp`
- `CMakeLists.txt`
- `package.xml`

(Refer to Appendix C for template code and fill in TODO parts.)

Implementation Steps

Step 3 – Build the Package

Return to the root of your workspace and build only the turtle_coop package

```
$  cd ~/<WSName>_ws
$  colcon build --packages-select turtle_coop
```

If errors occur, clean your workspace and rebuild the package:

```
$  cd ~/<WSName>_ws
$  rm -rf build/ install/ log/
$  colcon build --packages-select turtle_coop
```

Step 4 – Source the Workspace

Open a new terminal, and from inside the <WSName>_ws directory, run the following command to source your workspace:

```
$  source install/local_setup.bash
```

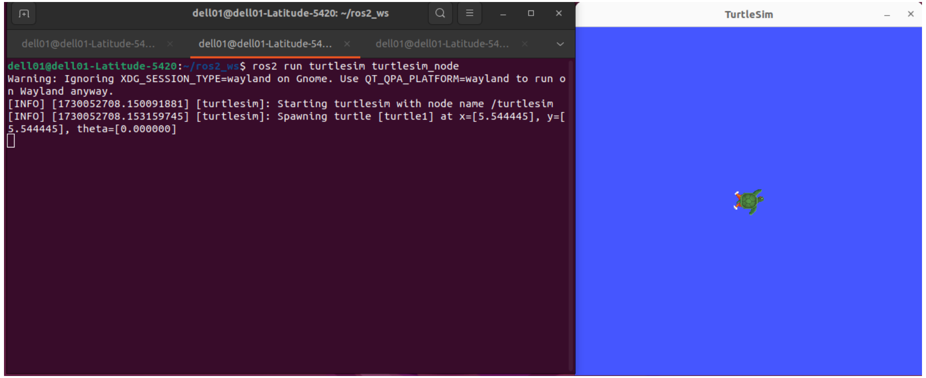
Do this EVERY time you open a new terminal.

Implementation Steps

Step 5 – Launch the Simulation

Terminal 1: Start Turtlesim

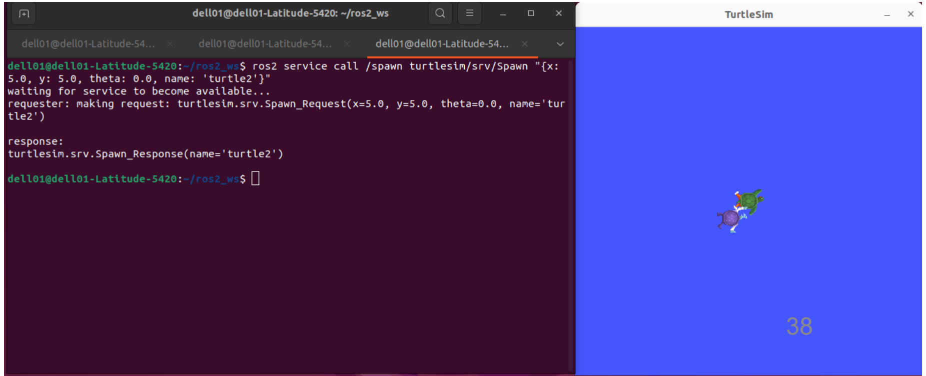
```
$  ros2 run turtlesim turtlesim_node
```


dell01@dell01-Latitude-5420: ~/ros2_ws
dell01@dell01-Latitude-54...
dell01@dell01-Latitude-54...
dell01@dell01-Latitude-54...
TurtleSim

```
dell01@dell01-Latitude-5420:~/ros2_ws$ ros2 run turtlesim turtlesim_node
Warning: Ignoring XDG_SESSION_TYPE=wayland on Gnome. Use QT_QPA_PLATFORM=wayland to run o
n Wayland anyway.
[INFO] [1730052708.150091881] [turtlesim]: Starting turtlesim with node name /turtlesim
[INFO] [1730052708.153159745] [turtlesim]: Spawning turtle [turtle1] at x=[5.544445], y=[
5.544445], theta=[0.000000]
```

Terminal 2: Spawn the second turtle

```
$  ros2 service call /spawn turtlesim/srv/Spawn "{x: 6.0, y: 5.0, name: 'turtle2'}"
```


dell01@dell01-Latitude-5420: ~/ros2_ws
dell01@dell01-Latitude-54...
dell01@dell01-Latitude-54...
dell01@dell01-Latitude-54...
TurtleSim

```
dell01@dell01-Latitude-5420:~/ros2_ws$ ros2 service call /spawn turtlesim/srv/Spawn "{x:
5.0, y: 5.0, theta: 0.0, name: 'turtle2'}"
waiting for service to become available...
requester: making request: turtlesim.srv.Spawn_Request(x=5.0, y=5.0, theta=0.0, name='tur
tle2')

response:
turtlesim.srv.Spawn_Response(name='turtle2')

dell01@dell01-Latitude-5420:~/ros2_ws$
```

Run and Observe the Cooperation

Step 6 – Control and Coordination

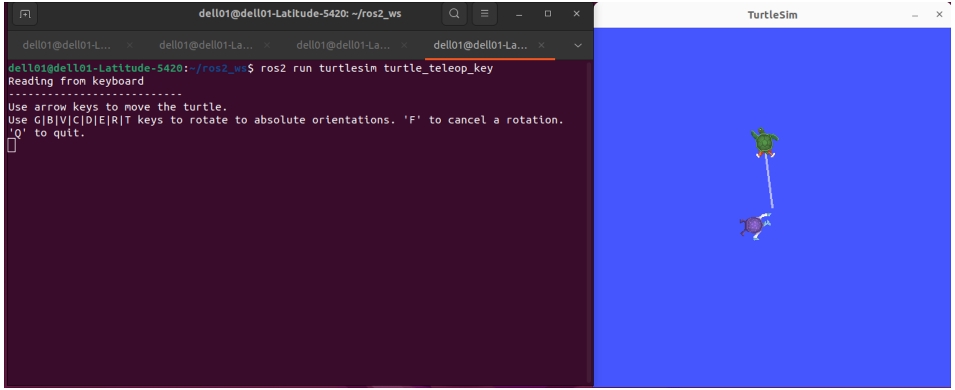
Terminal 3: Control the leader

```
$  ros2 run turtlesim turtle_teleop_key
```

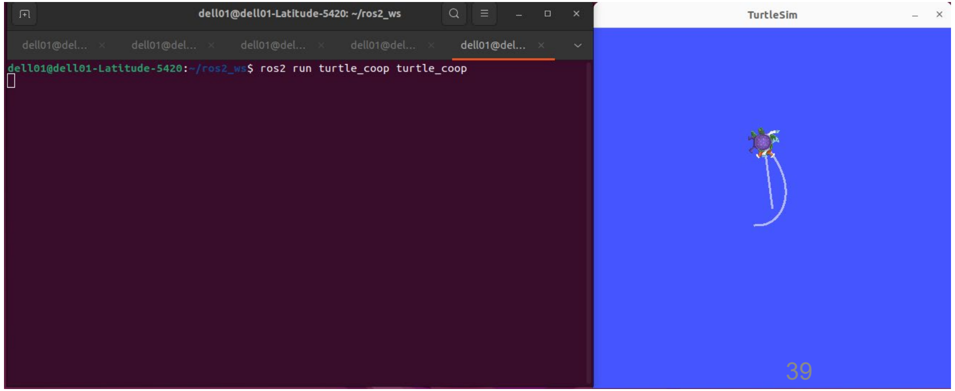
Terminal 4: Run the cooperative node

```
$  ros2 run turtle_coop turtle_coop
```

turtle1 moves via keyboard, turtle2 follows automatically.



```
dell01@dell01-Latitude-5420:~/ros2_ws$ ros2 run turtlesim turtle_teleop_key
Reading from keyboard
---------------------------
Use arrow keys to move the turtle.
Use G|B|V|C|D|E|R|T keys to rotate to absolute orientations. 'F' to cancel a rotation.
'Q' to quit.
```



```
dell01@dell01-Latitude-5420:~/ros2_ws$ ros2 run turtle_coop turtle_coop
```

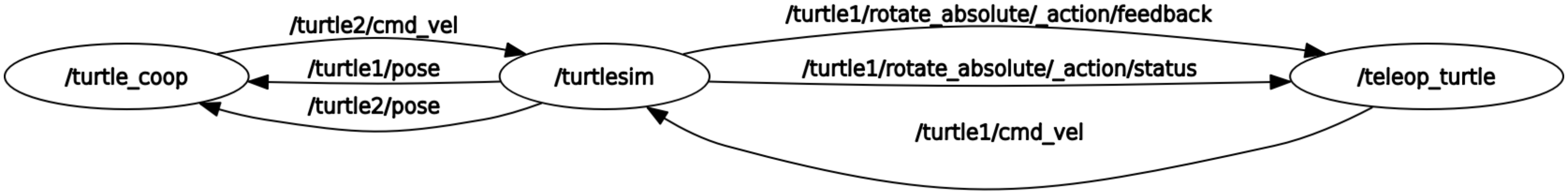
Run and Observe the Cooperation

Visualization

- The **rqt graph** shows message flow among:
  - `/turtle1/cmd_vel, /turtle2/cmd_vel`
  - `/turtle1/pose, /turtle2/pose`
  - `turtle_coop, turtle_teleop_key, turtlesim`

Illustrates real-time data exchange through ROS 2 topics.

```
$  rqt_graph
```



rqt graph for Turtle Cooperation

Improvements

Rotation Optimization

- The **follower turtle** sometimes takes a long (≈360°) rotation instead of the shortest path.
- Investigate and fix rotation direction logic.
- Think from a **trajectory planning** perspective:
    - How to ensure minimal rotation angle?
    - Could predictive heading correction improve response?

Improvements

Horizontal Overlap Issue

- Deviations occur when both turtles are **on the same horizontal line** or **overlap**.
- Possible mitigations:
  - Introduce a **tolerance threshold**
  - Apply **upper/lower bounds** on control
  - Add a **dead zone** to reduce oscillations

Improvements

Asymptotic Tracking

- Current tracking uses **proportional control**, leading to **asymptotic (slow) convergence**.
- Explore:
  - Nonlinear control (e.g., finite-time convergence)
  - Gain scheduling or adaptive feedback
  - Feedforward terms for faster response

Installing RX-150 Robotic Arm System

Installation Commands

```
$   sudo apt install curl

$   curl
    'https://raw.githubusercontent.com/Interbotix/interbotix_ros_manipulators/main/interbot
    ix_ros_xsarms/install/amd64/xsarm_amd64_install.sh' > ~/xsarm_amd64_install.sh

$   chmod +x ~/xsarm_amd64_install.sh

$   ~/xsarm_amd64_install.sh -d humble
```

During Installation

- No need to install the perception package: type ‘n’
- No need to install the MATLAB-ROS API: type ‘n’

Configure Environment Variables

```
$  echo "source ~/interbotix_ws/install/setup.bash" >> ~/.bashrc

$  source ~/.bashrc
```

💡 *These commands only need to be executed once.*

Verify Installation

Check Installed Interbotix ROS Packages

```
$  ros2 pkg list | grep interbotix
```

You should see packages such as:

```
interbotix_common_modules
interbotix_common_sim
interbotix_common_toolbox
interbotix_ros_xsarms
interbotix_ros_xsarms_examples
interbotix_ros_xseries
interbotix_tf_tools
interbotix_xs_driver
interbotix_xs_modules
interbotix_xs_msgs
interbotix_xs_ros_control
interbotix_xs_rviz
interbotix_xs_sdk
interbotix_xs_toolbox
interbotix_xsarm_control
interbotix_xsarm_descriptions
```

Run RX-150 in Gazebo

Launch the Simulation

```
$ ros2 launch interbotix_xsarm_descriptions xsarm_description.launch.py robot_model:=rx150 use_joint_pub_gui:=true
```

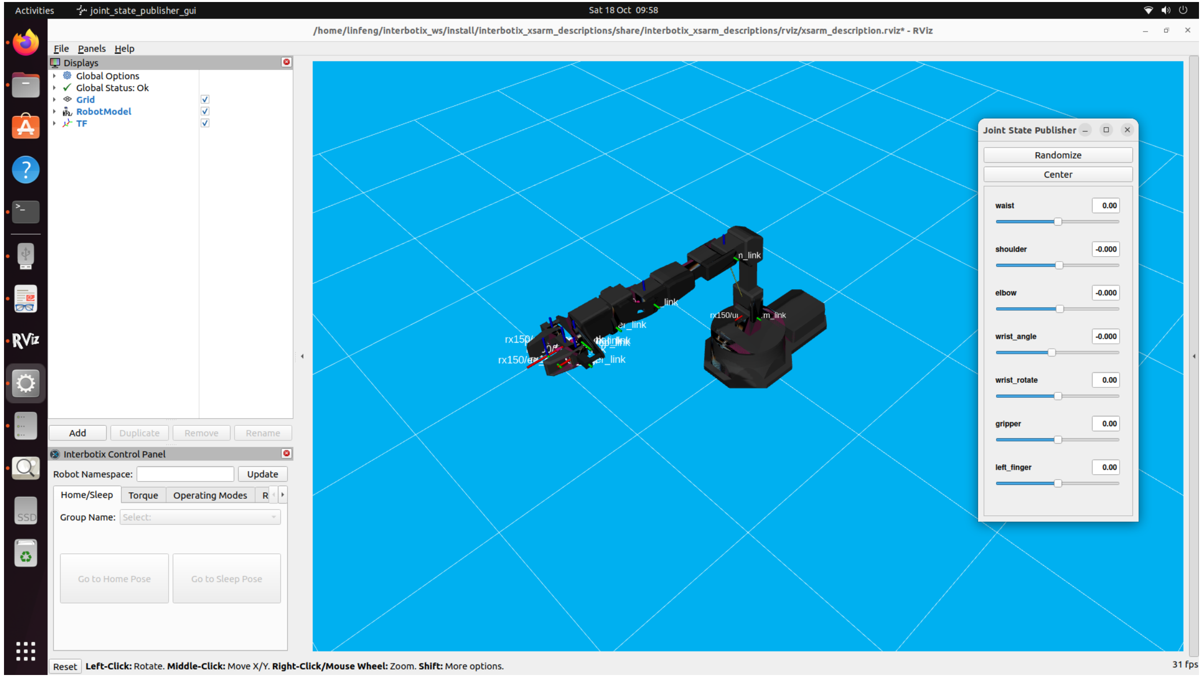
Use the Joint State Publisher

- **Randomize:** set all joints randomly within range
- **Center:** reset all joints to zero or neutral positions

Adjusting sliders updates joint angles published to `/joint_states`.

Run RX-150 in Gazebo

Graphical interface displays RX-150 arm and joint control panel for manual testing.



Activities joint_state_publisher_gui Sat 18 Oct 09:58

/home/linfeng/interbotix_ws/install/interbotix_xsarm_descriptions/share/interbotix_xsarm_descriptions/rviz/xsarm_description.rviz* - RViz

Displays
- Global Options
- Global Status: Ok
- Grid ☑
- RobotModel ☑
- TF ☑

Interbotix Control Panel

Robot Namespace:

Group Name: Select:

Joint State Publisher

waist	0.00
shoulder	-0.000
elbow	-0.000
wrist_angle	-0.000
wrist_rotate	0.00
gripper	0.00
left_finger	0.00

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

31 fps

https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros_interface/ros2/software_setup.html
https://docs.trossenrobotics.com/interbotix_xsarms_docs/ros_interface/ros2/quickstart.html

Summary

- Understood ROS 2 structure and workflow
- Practiced publisher–subscriber model
- Learned Turtlesim & rqt tools
- Implemented cooperative turtle control
- Installed RX-150 Robotic Arm System

THANK YOU