# AURORA ROBOTICS CORE WORKSHOP 1.0

# INTRODUCTION TO ROS2

# GENERAL CLASS RULES

- All mics should be muted
- I am always looking at the chat box so feel free drop questions anytime
- Feel free to use the “raise your hand” option anytime
- I recommend you have a note (could be digital), I tend to drop random knowledge casually

All slides, code and materials will be shared in the training repo. Feel free to use and share but do not modify

(I am very good with lawsuits)


Oya na, no wam. We go see why e tough

# WHAT IS ROS2?

- ROS = Robot Operating System, not actually an OS, it’s a framework.
- ROS2 is the next generation, built for performance, reliability, and real-time robotics.
- Think of it as the middleware that helps robots talk, think, and move in coordination.

It provides:

- Communication (Nodes ↔ Topics ↔ Services ↔ Actions)
- Tools for simulation, visualization, and debugging
- Standardization across robot types

# REAL-WORLD APPLICATIONS

- Industrial manipulators (UR5, Franka, Panda, ABB)
- Marine robotic manipulatoars (BlueROV, Reach Alpha 5)
- Autonomous lab arms (Pick & Place)
- Digital Twins in simulation (Gazebo / Isaac Sim)

# WHY ROS2?

| ROS (Classic) | ROS2 (Modern) |
| --- | --- |
| Built on Python 2 | Python 3 & C++17 |
| Not real-time | Real-time capable |
| Single-host | Multi-machine, DDS-based |
| ROS Master required | Decentralized (no master) |
| TCP-based | DDS middleware (fast, flexible) |

ROS2 lets multiple robots (land, air, and underwater) collaborate smoothly.

# ROS2 BUILDING BLOCKS

- Node: Small executable (e.g., arm_controller, camera_reader)
- Topic: Message highway between nodes (e.g., /joint_states)
- Message: The package of data (e.g., joint angles)
- Service: Request/response (e.g., move arm to position)
- Action: Long-running task (e.g., move arm slowly to a target)
- Parameter: Runtime configurable value (e.g., arm speed = 0.5 rad/s)

# ROS2 BUILDING BLOCKS

## What Is a Node?

A node is just a small program that does one job inside your robot. Every sensor, motor controller, or AI module runs as its own node.

Example

- camera_node 📷 – takes pictures
- arm_controller 💪 – moves the robot’s arm
- path_planner 🧭 – figures out where to go

Think of It Like Each node is a little worker in a factory. One senses, another decides, another moves – all talk to each other over walkie-talkies (topics).Nodes keep the robot modular. If one crashes, others still work. It’s teamwork, not a single giant program.

# ROS2 BUILDING BLOCKS

What Is a Topic?

- A topic is a named communication channel where nodes send or receive data.
  - Nodes that publish put messages onto the topic.
  - Nodes that subscribe listen and react.

- A radio frequency called /joint_states.The arm controller broadcasts joint angles; RViz listens and shows motion.

- Topics let multiple parts of a robot share live data, sensor readings, motor commands, camera images, without knowing each other directly.

# ROS2 BUILDING BLOCKS

## What Are Messages?

A message is the actual data packet sent over a topic. Every message has a defined type, like a blueprint describing what fields it contains.

Example
std_msgs/msg/String → text
std_msgs/msg/Float64 → a single number
sensor_msgs/msg/Image → camera frame
sensor_msgs/msg/LaserScan → lidar data

Letters inside an envelope. The topic is the postal route; the message is what’s written inside.Messages ensure both the sender and receiver agree on the data’s structure – no confusion, no mix-ups.

# ROS2 BUILDING BLOCKS

## Services

A service is a one-time request → response conversation between nodes.

Example:

- /reset_encoder → asks a motor node to reset its counter
- /spawn_turtle → adds a new turtle in simulation

Calling customer service: you ask a question, wait, and get an answer back.

Use services for tasks that happen occasionally, not continuously, like calibration or saving data.

# ROS2 BUILDING BLOCKS

## Actions

An action is for long-running tasks where you want feedback along the way.

Example: move a robot arm from start → goal in 10 seconds.

### Think of It Like

Ordering pizza online 🍕 – you get updates: “baking,” “on the way,” “delivered.”

### Why It Matters

Actions let nodes send progress updates, pause, or cancel operations (great for navigation or manipulation).

# ROS2 BUILDING BLOCKS

## Parameters

Parameters are configurable values stored inside a node.
 They control behavior without changing code.

Example
max_speed = 0.5
camera_exposure = 100

You can tune robot behavior live:

```
ros2 param list
ros2 param get /arm_controller max_speed
ros2 param set /arm_controller max_speed 0.8
```

# ROS2 BUILDING BLOCKS

Running a ROS 2 Node

```
ros2 run demo_nodes_cpp talker
```

What’s Happening

- ros2 run means “Run a program (node) that lives inside a package.”
- demo_nodes_cpp is the package name.
- talker is the node executable inside that package.
- This starts a node that publishes messages (text) over a topic named /chatter.
- Think of It Like: You’ve just turned on a radio station that starts broadcasting messages every second.
- This is how any robot component starts, you launch small nodes that do one job (talk, listen, move, or sense).

# ROS2 BUILDING BLOCKS

Listening to a Node

```
ros2 run demo_nodes_cpp listener
```

What’s Happening

- Another node starts, called listener.
- It subscribes to the same topic /chatter.
- It prints whatever the talker is sending.

- Think of It Like turning on a radio that’s tuned to the same frequency as the talker, now you can hear what’s being broadcast.
- This is how communication works in ROS 2: one node publishes, another subscribes. Together they form the basis for all robot communication, sensors sending data, motors receiving commands, etc.

# ROS2 BUILDING BLOCKS

Listing All Topics

```
ros2 topic list
```

What’s Happening

- This shows every topic that currently exists in your running ROS 2 system.
- Topics are like named message channels between nodes.

- Think of It Like A list of all the “radio frequencies” being used right now.

- It helps you see who’s talking about what – maybe /joint_states for your arm or /camera/image_raw for your camera.

# ROS2 BUILDING BLOCKS

Peeking Inside a Topic

```
ros2 topic echo /chatter
```

What’s Happening

- This listens directly to the /chatter topic and prints every message it receives.
- It doesn’t run a new node; it just “taps” into the existing conversation.

- Think of It Like Putting your ear close to the walkie-talkie and hearing the actual messages passing through.
- When debugging robots, this command lets you see live data: temperatures, positions, sensor readings, etc.

# ROS2 BUILDING BLOCKS

Listing All Nodes

```
ros2 node list
```

What’s Happening

- Shows all currently running nodes in your ROS 2 system.
- Each node is like a small independent program with a specific purpose.

- Think of It Like Checking who’s online in your robot network “talker”, “listener”, “joint_controller”, etc.
- You use it to confirm your robot’s brain parts are alive and communicating.

# ROS2 BUILDING BLOCKS

Listing All Services

```
ros2 service list
```

What’s Happening

- Displays all available services – special request/response channels.
- A service lets one node ask another to do something once (like “take a photo now” or “reset motor”).
- Think of It Like Calling customer service: you ask a question, wait, and get an answer back.
- Understanding services prepares you for robot actions that aren’t continuous streams, like calibration, resetting sensors, or saving data.

# ROS2 IN ACTION (CONCEPT OVERVIEW)
# A ROBOT ARM WITH 6 JOINTS.

| Node | Function | Communicates via |
| --- | --- | --- |
| joint_publisher | Reads angles from encoders | /joint_states topic |
| trajectory_planner | Plans smooth motion | /trajectory_cmd topic |
| motor_controller | Converts command to motor torque | /motor_cmd topic |
| gripper_service | Opens/closes gripper | /gripper_control service |

# TASK 3

Create a workspace

```
mkdir -p ros2_class1_ws/src
cd ros2_class2_ws
```

we will start learning with some already available demos

```
git clone -b humble --depth 1 https://github.com/ros-controls/ros2_control_demos src/ros2_control_demos
```

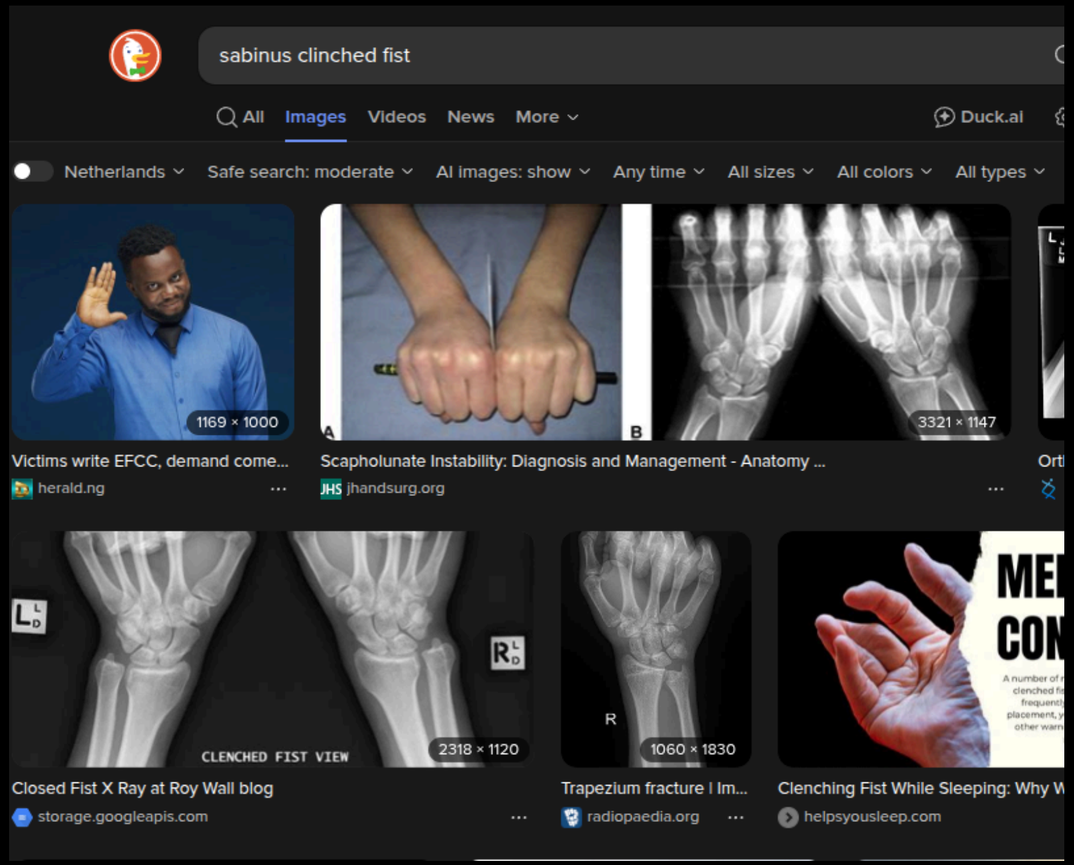
Build the packages

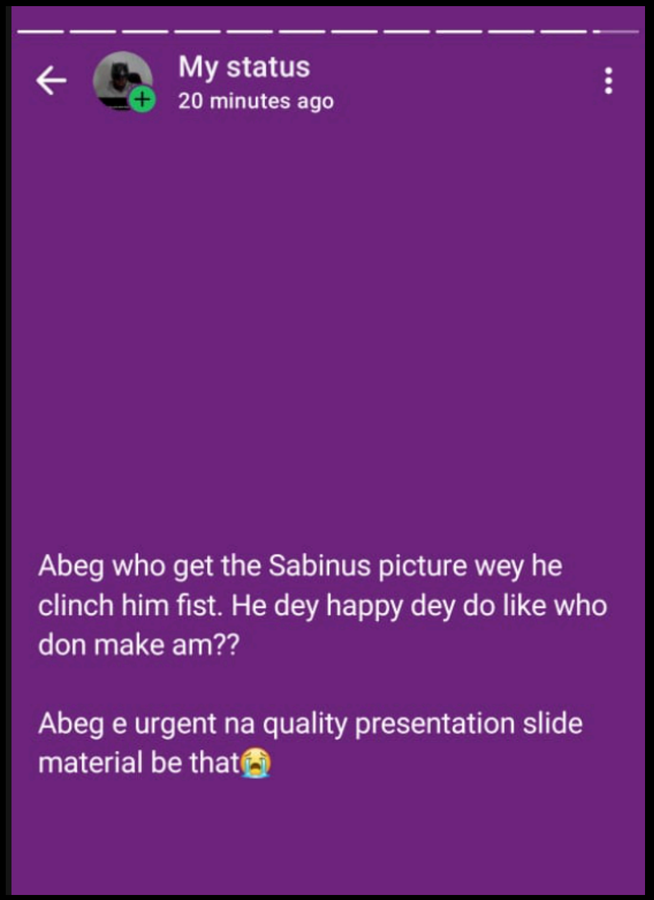
```
colcon build
```

# By now you should have some errors

# WHAT'S NEXT?

First appreciate me for the sabinus sticker, I put in alot of effort to find it




NOW MOVE ON TO THE NEXT SLIDE

# WHAT WE'RE TRYING TO DO IN THE TASK

- Imagine ROS 2 as a big workshop for robots, a place where all your robot's parts (arms, sensors, wheels, etc.) can talk to each other using code.

- Before we start building things in this big workshop, it is a nice idea to create our own personal workbench, a space on your computer where your robot projects will live.

Hence:

Create a workspace

```
mkdir -p ros2_class1_ws/src
cd ros2_class2_ws
```

# WHAT WE'RE TRYING TO DO IN THE TASK

- `mkdir -p ros2_class1_ws/src`
- `cd ros2_class1_ws`

- What this means:
- `mkdir` just means "make a new folder."
- `-p` tells it to make any missing parent folders too.
- `ros2_class1_ws` will be your ROS 2 workspace (a special folder where you keep your robot projects).
- Inside it, we make another folder called `src` – that's where the actual source code (packages) will go.

- `cd` just means "go inside that folder."

- Think of it like: "We've just built an empty toolbox and opened it, ready to fill it with robot parts."

# WHAT WE'RE TRYING TO DO IN THE TASK

`git clone -b humble --depth 1 https://github.com/ros-controls/ros2_control_demos src/ros2_control_demos`

This gets some demo robots to play with

- What this means:
  - `git clone` means "download this project from GitHub."
  - We're downloading a ready-made collection of demo robots (arms, wheels, grippers, etc.) made by the ROS team.
  - The flag `-b humble` makes sure we get the version that matches ROS 2 Humble (your ROS 2 release).
  - `--depth 1` means "just get the latest copy, not the whole history" — so it downloads faster.
  - We put it inside `src/` because that's where all ROS 2 packages live.

- Think of it like: we're grabbing a box of free sample robots so we can learn by playing with them.

# WHAT WE'RE TRYING TO DO IN THE TASK

Build the Workspace

colcon build

What this means:

- colcon is the tool that builds and connects all the ROS 2 packages in your workspace.

- It looks inside the src folder, finds every package, compiles the code, and places the results in a new build/ and install/ folder.

After it finishes, it's like:
"We've assembled all the robot parts, screwed everything together, and now the robots are ready to run!" buttttttttt... next slide

# WHAT WE'RE TRYING TO DO IN THE TASK

When we run colcon build, we're basically saying:

"Hey computer, please take everything inside this robot project and put it together so it can run."

But… the project we downloaded (ros2_control_demos) doesn't live alone. It depends on other tools, like bolts and screws, that weren't installed yet on our computer.

Imagine It Like Building a Robot Kit

If you bought a robot kit and the instruction manual said:
"Attach the arm using a 4 mm Allen key,"
but you never owned an Allen key – you'd get stuck, right?
That's exactly what happens when we build without installing dependencies.

# WHAT ARE DEPENDENCIES

Dependencies are just other packages or libraries that our robot's code needs to work.

They contain things like:

- Functions to move motors
- Definitions of joint types
- Ready-made controller logic (e.g., PID loops, sensor readers)

When we try to build ros2_control_demos, it looks for these tools:

- ros2_control – the main framework for hardware + control
- ros2_controllers – pre-made controllers (joint, position, velocity)
- ros2_control_test_assets – example files for testing setups
- ros2_control_cmake – build instructions for the others

But since we haven't installed them yet, the computer says:
"I can't find the tool you're talking about!" and the build fails.

# WHAT DO YOU LEARN FROM THIS FAILURE?

1. That ROS packages depend on each other like modules in a big ecosystem.
2. That errors can be informative we can read the terminal to see what’s missing.
3. That fixing it is usually simple: install the missing packages

The solution?

- `sudo apt update`
- ```
  sudo apt install \
      ros-humble-ros2-control \
      ros-humble-ros2-controllers \
      ros-humble-ros2-control-test-assets \
      ros-humble-ros2-control-cmake
  ```
- Then rebuild


  ```

# E TOUGH ABI?

# WHAT'S NEXT?

```
source install/setup.bash
```

```
ros2 launch ros2_control_demo_example_1
view_robot.launch.py
```

# WHAT'S NEXT?

- Learn URDF (Robot Description Format)
- Build your first custom arm model
- Add control with ros2_control
- Visualize motion in Gazebo / RViz2
- Integrate with MoveIt2 for planning

“ROS2 isn’t just software and it’s definitely not an operating system it’s how robots communicate, collaborate, and come alive.”

# Thank you for learning