6-DOF Robotic Arm with ROS2 Control & Gazebo Integration

Task Statement

Create a complete ROS2-controlled 6-DOF robotic arm simulation system that integrates URDF modeling, ROS2 Control, Gazebo physics simulation, and RViz visualization. The project involves building two ROS2 packages that work together to provide a fully functional robotic arm with position control capabilities.

Project Goal: Develop a simulation environment where a 6-DOF robotic arm can be controlled through ROS2 control topics, visualized in RViz, and simulated with proper physics in Gazebo.

Key Requirements

- Create `arm_description` package with proper URDF model
- Add `<transmission>` elements for each joint
- Configure ROS2 Control hardware interface
- Build `arm_gazebo` package for Gazebo integration
- Implement `gazebo_ros2_control` plugin
- Configure joint controllers (joint_state_broadcaster, position_trajectory_controller)
- Verify arm visualization in RViz
- Test joint control using ROS2 control CLI
- Ensure proper physics behavior in Gazebo

Step-by-Step Approach

Phase 1: Package Setup

1.1 Initial Environment Setup

```
# Source ROS2 Jazzy
source /opt/ros/jazzy/setup.bash
```

1.2 Create Workspace and Packages

```
# Create workspace
mkdir -p ~/arm_ws/src
cd ~/arm_ws/src

# Create arm_description package
ros2 pkg create --build-type ament_cmake --license Apache-2.0 arm_description

# Create arm_gazebo package
ros2 pkg create --build-type ament_python --license Apache-2.0 arm_gazebo
```

1.3 Install Required Dependencies

```
# Install ROS2 Control and Gazebo packages for Jazzy
sudo apt install ros-jazzy-controller-manager
sudo apt install ros-jazzy-gz-ros2-control
sudo apt install ros-jazzy-joint-trajectory-controller
sudo apt install ros-jazzy-rqt-joint-trajectory-controller

# Additional useful packages
sudo apt install ros-jazzy-joint-state-broadcaster
sudo apt install ros-jazzy-robot-state-publisher
sudo apt install ros-jazzy-rviz2
```

1.4 Build and Source the Workspace

```
# Navigate to workspace root
cd ~/arm_ws

# Build packages with symlink-install flag
colcon build --symlink-install

# Source the workspace
source install/setup.bash
```

1.5 Directory Structure

```
arm_description/
├── urdf/
│   ├── control
│       ├── arm_transmission.urdf.xacro
│       └── gazebo_sim_ros2_control.urdf.xacro
│   ├── mech
│       ├── adaptive_gripper.urdf.xacro
│       ├── arm_base.urdf.xacro
│       └── arm_define.urdf.xacro
│   ├── robots
│       └── arm_model.urdf.xacro
│       └── arm_define.urdf.xacro
├── meshes/
│   ├── 6dof_arm/visual
│   ├── adaptive_gripper
│   ├── d435
│   └── g_shape_base_v2_0
├── launch/
│   └── display.launch.py
├── rviz/
│   └── arm_6dof.rviz
├── CMakeLists.txt
└── package.xml

arm_gazebo/
├── config/
│   └── simple_controller.yaml
├── worlds/
│   └── empty_world.sdf
├── launch/
│   ├── 6dof_gazebo_controller.launch.py
├── setup.py
└── package.xml
```

Phase 2: URDF Model Creation

2.0 Base Model Reference

This project utilizes the **Elephant Robotics 6-DOF robotic arm URDF files** as the foundation for the robot model.

2.1 MECHANICAL SEGMENT - Links and Joints Definition

The robotic arm consists of **6 rotational links** plus an **adaptive gripper** end-effector.

Link Structure Overview

- **base_link**: Fixed base platform (ground connection)
- **link1**: Base rotation link (shoulder yaw)
- **link2**: Shoulder pitch link
- **link3**: Elbow pitch link
- **link4**: Wrist roll link
- **link5**: Wrist pitch link
- **link6**: Wrist yaw link (flange)
- **gripper_base**: Adaptive gripper assembly

2.2.1 Link Components

Each link in the URDF contains three critical elements:

A. Inertial Properties

```xml
<inertial>
  <origin xyz="0.0 0.0 0.05" rpy="0 0 0"/>
  <mass value="0.85"/>
  <inertia
    ixx="0.001458" ixy="0.0" ixz="0.0"
    iyy="0.001458" iyz="0.0"
    izz="0.002025"/>
</inertial>
```

Location: Found in `urdf/mech/arm_define.urdf.xacro`

B. Visual Properties

```xml
<visual>
  <origin xyz="0 0 0" rpy="0 0 0"/>
  <geometry>
    <mesh filename="package://arm_description/meshes/6dof_arm/visual/link2.dae" scale="0.001 0.001 0.001"/>
  </geometry>
  <material name="link_material">
    <color rgba="0.1 0.1 0.1 1.0"/>  <!-- Ambient: Dark gray base -->
  </material>
</visual>
```

Location: Found in `urdf/mech/arm_define.urdf.xacro`

Material Properties Explained:

Mesh Files: High-quality `.dae` format containing detailed 3D geometry exported from CAD software.

C. Collision Properties

```xml
<collision>
  <origin xyz="0 0 0.075" rpy="0 0 0"/>
  <geometry>
    <cylinder radius="0.035" length="0.15"/>
  </geometry>
</collision>
```

Location: Found in `urdf/mech/arm_define.urdf.xacro`

```
robot name is: arm_6dof
---------- Successfully Parsed XML ---------------
root Link: world has 1 child(ren)
    child(1):  base_link
        child(1):  link1
            child(1):  link2
                child(1):  link3
                    child(1):  link4
                        child(1):  link5
                            child(1):  link6
                                child(1):  link6_flange
                                    child(1):  gripper_base
                                        child(1):  gripper_left2
                                        child(2):  gripper_right2
                                        child(3):  gripper_right3
                                            child(1):  gripper_right1
                                        child(4):  gripper_left3
                                            child(1):  gripper_left1
```

2.2 CONTROL SEGMENT - ROS2 Control Configuration

This section bridges the mechanical robot description with the ROS2 Control framework, enabling software control of the hardware through standardized interfaces.

File Location: `urdf/control/arm_transmission.urdf.xacro`

2.2.1 Transmission Elements

Purpose: Transmission elements connect the physical joints to the control system's hardware interface.

Simple Transmission element which provide 1:1 direct drive is used

```xml
<!-- Example from arm_transmission.urdf.xacro -->
<transmission name="joint_1_trans">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="joint_1">
    <hardwareInterface>hardware_interface/PositionJointInterface</hardwareInterface>
  </joint>
  <actuator name="joint_1_motor">
    <mechanicalReduction>1</mechanicalReduction>
```

```
  </actuator>
</transmission>
```

2.2.2 Command and State Interfaces

In ROS2 Control, hardware components communicate with controllers through two interface types:

STATE INTERFACES (Read-Only) - Provide data to controllers:

- `position` - Current joint angle (radians)
- `velocity` - Current angular speed (rad/s)
- `effort` - Current torque (N·m)

COMMAND INTERFACES (Write-Only) - Receive control commands:

- `position` - Target joint angle
- `velocity` - Target angular speed
- `effort` - Target torque

2.2.3 Understanding ROS2 Control & Controller Manager

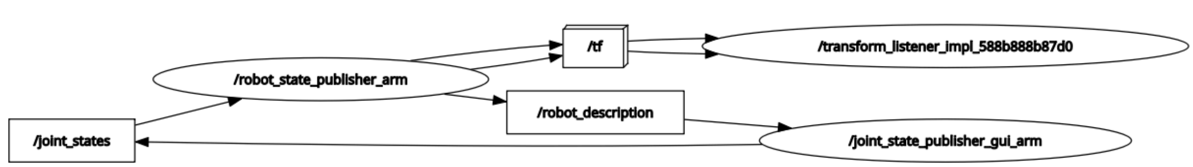
Reference: ROS2 Control Explained by Masum

1. **Hardware Interface** (Translator)
    - Knows how to communicate with specific motors and sensors
    - Makes the same code work with different robots
2. **Controller Manager** (Brain)
    - Manages multiple controllers without conflicts
    - Loads and unloads controllers dynamically
    - Switches between controllers smoothly (e.g., position → velocity control)
3. **Controllers** (Specialized Workers)
    - Each performs a specific control task
    - `joint_state_broadcaster`: Publishes joint states to `/joint_states` topic
    - `joint_trajectory_controller`: Follows position trajectories

Phase 3: RViz robot model visualize

Launch File Pseudocode:

1. Find package directory for 'arm_description'
2. Build path to URDF file
   → Joins: [package_path, 'urdf', 'robots', 'arm_model.urdf.xacro']
   → Result: .../arm_description/urdf/robots/arm_model.urdf.xacro
3. Build path to RViz config file
   → Joins: [package_path, 'rviz', 'config.rviz']
   → Result: .../arm_description/rviz/config.rviz
4. Process Xacro to URDF and convert to string
   → Command: 'xacro ' + urdf_path
   → Executes xacro processor
   → Output wrapped in ParameterValue(value_type=str)
5. Create robot_state_publisher node
   → Subscribes to /joint_states
   → Publishes TF transforms for all links
   → Parameters: robot_description (the processed URDF string)
6. Create joint_state_publisher_gui node
   → Provides GUI sliders for each joint
   → Publishes to /joint_states topic
   → Allows manual joint manipulation for testing
7. Create RViz2 node
   → Launches visualization window
   → Loads config file with pre-configured displays
   → Arguments: '-d' flag with config file path
8. Return LaunchDescription with all nodes
   → All nodes start together when launch file runs



Phase 4: Gazebo Simulation Package

The `arm_gazebo` package handles physics simulation and controller management. Unlike `arm_description` which only visualizes, The controllers.yaml file lives in the `arm_gazebo` package, not `arm_description`, because controller configuration is specific to the simulation/hardware platform, not the robot model itself.

4.1 Controller Configuration File

Purpose: Defines which controllers are available and how they're configured.

PSEUDOCODE: simple_controller.yaml

SECTION 1: Controller Manager Configuration

- update_rate: 100 Hz (control loop frequency)
- List of available controllers:
    - joint_state_broadcaster (type: JointStateBroadcaster)
    - joint_trajectory_controller (type: JointTrajectoryController)

SECTION 2: Trajectory Controller Configuration

- joints: [joint_1, joint_2, joint_3, joint_4, joint_5, joint_6,gripper_controller]
- command_interfaces: [position]
  → Controller WRITES position commands to hardware
- state_interfaces: [position, velocity]
  → Controller READS current position and velocity from hardware
- state_publish_rate: 50 Hz
  → How often to publish joint states
- action_monitor_rate: 20 Hz
  → How often to check trajectory goal progress
- constraints:
    - stopped_velocity_tolerance: 0.01 rad/s
    - goal_time: 0.0 (no time constraint)
      → Defines when trajectory is considered "complete"

4.2 Gazebo Launch File

Purpose: Starts Gazebo, spawns robot, loads controllers, and configures ROS2 Control.

Gazebo Launch

PSEUDOCODE : 6dof_gazebo_controller.launch.py

Step 1: Declare the arm_description package and define the URDF file

Step 2 : Launch the gazebo simulator launch file (ros_gz_sim/launch/gz_sim.launch.py)

Define the world file and declare the gz_args with simulation parameters

Step 3 : Create a node 'robot_state_publisher' to publisher /tf (transform) for all the robot link

Subscribes to /joint_states, publishes /tf and /tf_static

Step 4 : Spawn the robot using /robot_description

Step 5 : Define the bridge to maintain sync with Gazebo and ROS nodes

Step 6 : Using controller_manager create the broadcaster and controller

Loads the arm_controller (JointTrajectoryController)
Reads configuration from simple_controller.yaml

Step 7 : Apply delay before lading the broadcaster for the robot to exist in gazebo. Apply the same on controller to load once the joints feedback is obtained

Step 8 : Provide the rqt_gui package to perform the trajectory on the joints and gripper

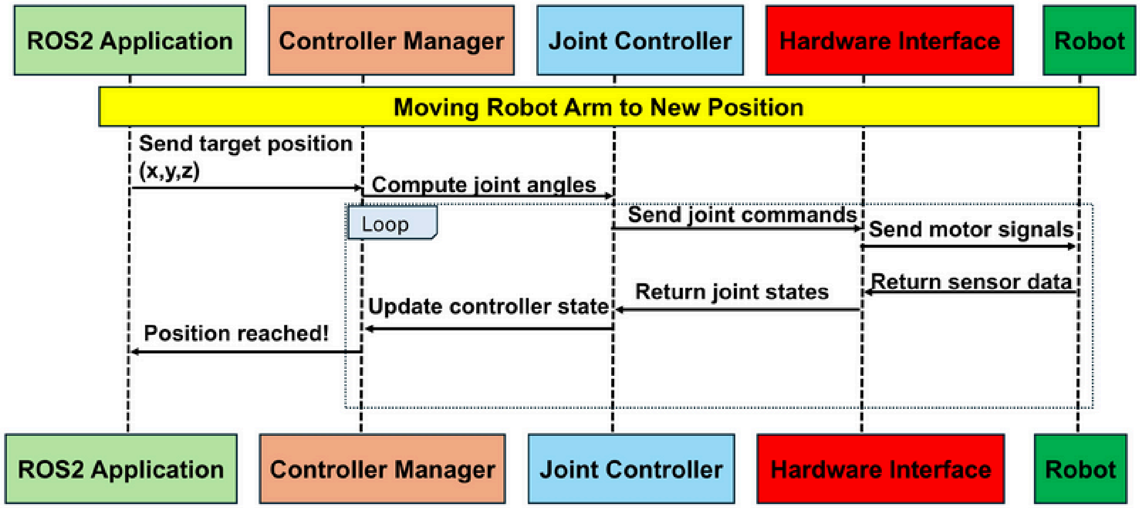
Step 9 : Return using LaunchDescription

The gz_ros2_control Plugin: The Hidden Bridge

`gz_ros2_control` is a Gazebo system plugin that acts as the critical bridge between:

- **Gazebo's physics simulation** (where your robot lives)
- **ROS2 Control framework** (where your controllers live)

How it works:[1][2]



The plugin must be declared in your URDF file (typically in `gazebo_sim_ros2_control.urdf.xacro`):

```
<gazebo>
  <plugin filename="gz_ros2_control-system" name="gz_ros2_control::GazeboSimROS2ControlPlugin">
    <parameters>$(find arm_gazebo)/config/simple_controller.yaml</parameters>
  </plugin>
</gazebo>
```

Phase 5: Trajectory Control and Testing

5.1 Using rqt_joint_trajectory_controller

The `rqt_joint_trajectory_controller` is a **graphical interface** for sending trajectory commands to your robot's joint trajectory controller. It's part of the ROS2 Control ecosystem and provides an intuitive way to test joint movements.

What it does:

- Provides sliders for each joint to set target positions
- Sends trajectory goals to the `joint_trajectory_controller`
- Shows controller status and feedback
- Useful for testing individual joint movements and validating controller configuration

Launch with rqt_gui:

```
# Method 1: Launch directly (if included in launch file)
ros2 launch arm_gazebo 6dof_gazebo_controller.launch.py

# Method 2: Launch separately in a new terminal
ros2 run rqt_joint_trajectory_controller rqt_joint_trajectory_controller

# Method 3: Through rqt_gui plugin
ros2 run rqt_gui rqt_gui
# Then: Plugins → Robot Tools → Joint Trajectory Controller
```

Usage:

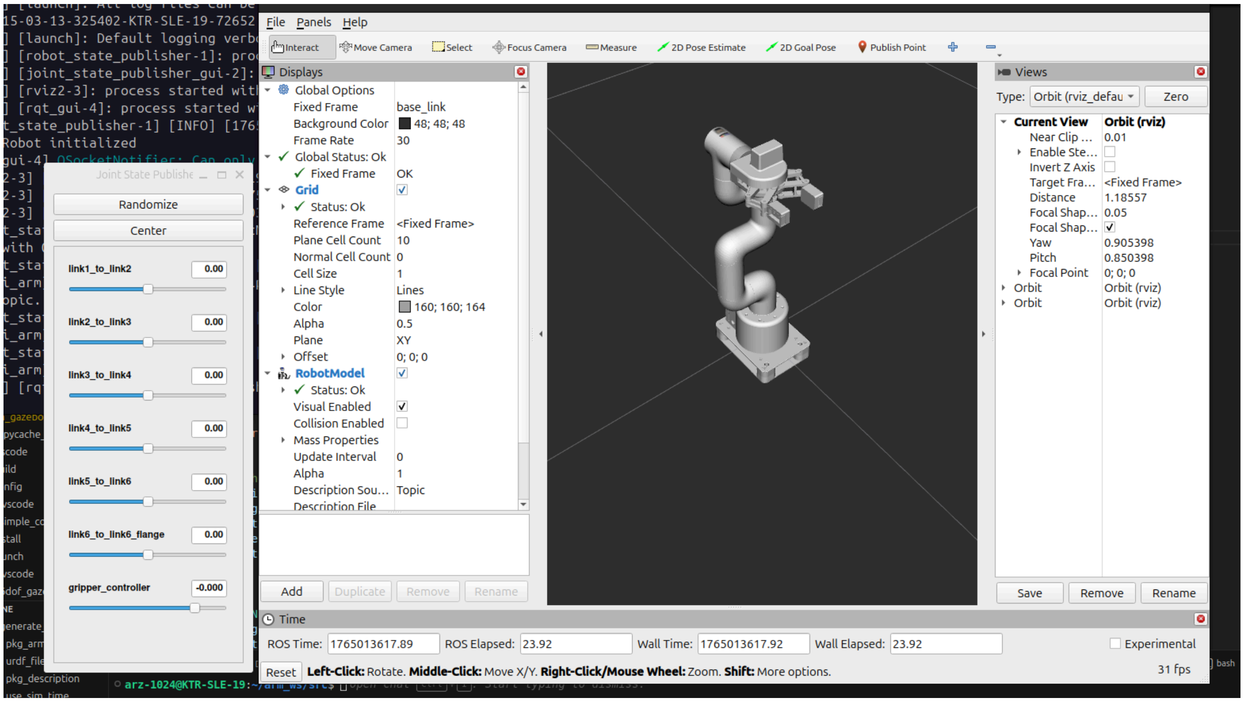
1. Select the controller manager namespace (usually `/controller_manager`)
2. Select the controller name (`arm_controller`)
3. Use sliders to set desired joint positions
4. Monitor the execution in Gazebo

Alternative: Command Line Testing

Phase 6: Build and Test

6.1 Test in RViz

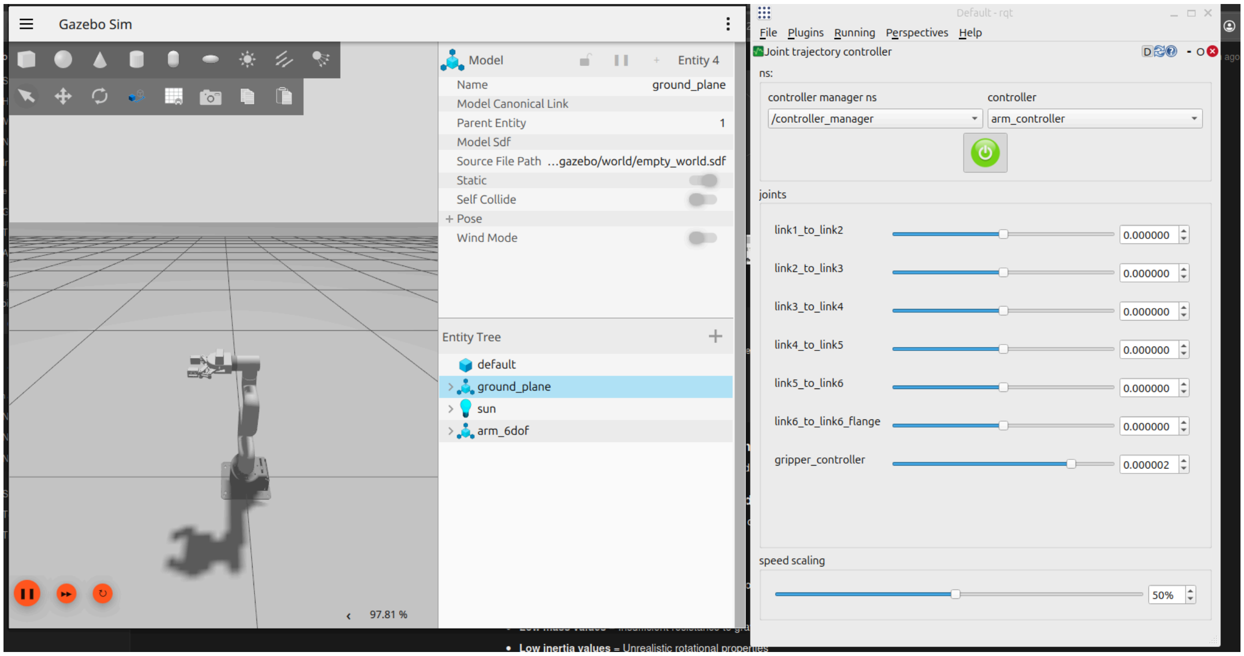
```
ros2 launch arm_description 6dof_rviz.launch.py
```



6.2 Test in Gazebo

```
# Terminal 1: Launch the gazebo file
ros2 launch arm_gazebo 6dof_gazebo_controller.launch.py

# Terminal 2: Check controllers
ros2 control list_controllers
```



Phase 7: Common Errors and Debugging

This section documents common issues encountered during development and their solutions.

Error 1: Robot Collapsing to the Ground

Problem: Robot spawns in Gazebo but immediately crumples or falls through the ground.

Symptoms:

- Joints fold inward or bend unexpectedly
- Robot appears unstable and collapses under its own weight

Root Causes:

- **Low mass values** = Insufficient resistance to gravity
- **Low inertia values** = Unrealistic rotational properties
- **Missing inertial properties** for links

Solution:

```xml
<!-- Ensure each link has proper inertial properties -->
<inertial>
  <origin xyz="0.0 0.0 0.05" rpy="0 0 0"/>
  <mass value="0.85"/>  <!-- Realistic mass in kg -->
  <inertia
   ixx="0.001458" ixy="0.0" ixz="0.0"
   iyy="0.001458" iyz="0.0"
```

```
  izz="0.002025"/>
</inertial>
```

Error 2: Robot Disappears Below the Ground

Problem: Robot falls through the ground plane in Gazebo.

Root Cause:

- Missing `<collision>` elements in URDF
- Collision geometry defines the robot's physical boundary for physics calculations
- Without collision, Gazebo doesn't know the robot's physical shape

Solution:

```
<!-- Add collision element to each link -->
<collision>
  <origin xyz="0 0 0.075" rpy="0 0 0"/>
  <geometry>
    <cylinder radius="0.035" length="0.15"/>
  </geometry>
</collision>
```

Error 3: gz_ros2 Package Not Found

Problem: Launch fails with error about missing `gz_ros2` package.

Root Cause:

- Used **Gazebo Ignition** supporting package
- Package names changed in newer ROS2/Gazebo versions

Solution: ***ros_gz_sim*** package is used instead

Error 4: Controllers Not Activating

Problem: Controllers show as `inactive` even after launching.

```
$ ros2 control list_controllers
arm_controller        joint_trajectory_controller/JointTrajectoryController  inactive
joint_state_broadcaster joint_state_broadcaster/JointStateBroadcaster        inactive
```

Debugging Approach:

Step 1: Test with Mock Components

Mock components help debug the controller and launch file integration **without** considering hardware interface issues.

Added to URDF temporarily:

```xml
<ros2_control name="GazeboSystem" type="system">
  <hardware>
    <plugin>mock_components/GenericSystem</plugin>
  </hardware>
  <!-- ... joint definitions ... -->
</ros2_control>
```

Work with mock components, the issue is with the Gazebo hardware interface, not controller configuration.

If no interfaces appear, the hardware plugin isn't loading.

Step 2: Common Root Causes

A. Plugin Not Found

```
Caught exception: N9pluginlib20LibraryLoadExceptionE
According to the loaded plugin descriptions the class
gz_ros2_control/GazeboSimSystem does not exist
```

Solution: Check that `gz_ros2_control` is installed:

```
sudo apt install ros-jazzy-gz-ros2-control
```

B. Duplicate Controller Manager

Problem: Launching a second controller manager node conflicts with Gazebo's built-in one.

Bad approach:

```
# DON'T DO THIS - creates duplicate controller manager
control_node = Node(
    package='controller_manager',
    executable='ros2_control_node',
    parameters=[{'robot_description': robot_description_content},
              robot_controllers],
```

```
    output='both'
)
```

Solution: Remove the standalone `controller_manager` node. The `gz_ros2_control` plugin automatically starts a controller manager inside Gazebo.

C. Wrong Gazebo Launch Method

Problem: Using `ExecuteProcess` to launch Gazebo doesn't establish ROS2 integration.

This launches Gazebo standalone (like running `gz sim` in terminal) with **no ROS2 bridge**.

```
# USE THIS - proper ROS2 integration
gazebo_launch = IncludeLaunchDescription(
    PythonLaunchDescriptionSource(
        [PathJoinSubstitution([
            FindPackageShare('ros_gz_sim'),
            'launch',
            'gz_sim.launch.py'
        ])]
    ),
    launch_arguments={'gz_args': gz_args}.items()
)
```

Step 3: Timing Issues

Problem: Controllers try to load before robot exists in Gazebo.

Solution: Add delays using `RegisterEventHandler`:

```
# Wait for robot to spawn before loading controllers
delay_joint_state_broadcaster = RegisterEventHandler(
    event_handler=OnProcessExit(
        target_action=spawn_entity,
        on_exit=[joint_state_broadcaster_spawner],
    )
)

delay_arm_controller = RegisterEventHandler(
    event_handler=OnProcessExit(
        target_action=joint_state_broadcaster_spawner,
        on_exit=[arm_controller_spawner],
```

```
    )
)
```

Verification Checklist

Controllers active:

```
$ ros2 control list_controllers
arm_controller          joint_trajectory_controller/JointTrajectoryController  active
joint_state_broadcaster joint_state_broadcaster/JointStateBroadcaster          active
```

Useful Commands Reference

```
# Check controller status
ros2 control list_controllers
ros2 control list_hardware_interfaces
ros2 control list_hardware_components

# View joint states
ros2 topic echo /joint_states

# Debug URDF
xacro ~/arm_ws/src/arm_description/urdf/robots/arm_model.urdf.xacro > /tmp/robot.urdf.
xacro
check_urdf /tmp/robot.urdf.xacro

# Debug YAML file
yamllint simple_controller.yaml
```

OUTPUT

https://meeting.zoho.in/meeting/public/videoprv?recordingId=690e544e17cef4ac7817e4744520b548eea39ce35376554e6ccf50e792148e12&x-meeting-org=60014996923