6-DOF Robotic Arm with ROS 2 Control & PID-Based Trajectory Control

1. Introduction

The focus of this report is not limited to robot modeling or simulation, but specifically on:

- Research and comparison of available PID control methods in ROS 2 Jazzy
- Correct use of **JointTrajectoryController (JTC)** and **PID control via effort interfaces**
- Practical controller configuration and tuning for stable joint motion
- Validation of smooth, oscillation-free motion for all six joints

---

2. Task Overview

The task involved configuring a complete ROS 2 control pipeline for a 6-DOF robotic arm, integrating:

- URDF-based robot modeling
- ros2_control hardware abstraction
- Gazebo physics simulation
- Joint-level trajectory control with PID tuning

The final objective was to achieve **stable and smooth trajectory tracking** for all joints, verified through runtime observation and plotting of controller feedback.

---

3. Research on Control Methods (ROS 2 Jazzy)

3.1 Method 1: `control_toolbox` PID Library

The `control_toolbox` package provides the **core PID mathematics** used throughout ROS 2 controllers. It implements a classical PID structure without automatic mode

switching.

Control behavior depends solely on gain configuration:

- P only: I = 0, D = 0
- PI: D = 0
- PD: I = 0
- PID: P, I, and D all enabled

This library does not manage trajectories, hardware interfaces, or simulation. Its role is limited to computing control effort and is therefore **not sufficient as a standalone control solution** for a multi-joint manipulator.

3.2 Method 2: `ros2_control` with JointTrajectoryController

The recommended approach in ROS 2 Jazzy is the **ros2_control framework**, which separates responsibilities into:

- Hardware interfaces (state and command)
- Controller manager
- Controller plugins

The **JointTrajectoryController (JTC)** is responsible for:

- Interpolating joint trajectories
- Managing execution timing
- Computing tracking errors using feedback

According to the official `ros2_controllers` documentation:

- **PID gains are applied only when the command interface is `effort`**
- Position or velocity command interfaces perform trajectory following **without PID control**

Internally, when effort control is used, JTC invokes `Pid` to convert tracking error into torque commands.

---

4. Selected and Implemented Control Strategy

4.1 Controller Selection Rationale

The **JointTrajectoryController with effort command interface** was selected because it:

- Is fully supported and documented in ROS 2 Jazzy
- Allows per-joint PID tuning
- Integrates cleanly with Gazebo through `gz_ros2_control`
- Matches the control architecture described in ROS 2 control specifications

This decision is reflected in the updated PID section of the GitHub repository.

4.2 Controller Architecture (Conceptual Flow)

```
Trajectory Goal
   ↓
Trajectory Trajectory controller (JTC)
   ↓
Desired Position / Velocity
   ↓
State Feedback (position, velocity)
   ↓
Error Computation
   ↓
PID (control_toolbox::PidROS)
   ↓
Effort Command → Gazebo Joint
```

This architecture ensures deterministic behavior and stable control under simulation physics.

---

5. Controller Configuration Summary

The controller configuration follows the same structure as described in the ARM

reference PDF.

5.1 Controller Manager

- Update rate: 60 Hz
- Controllers loaded:
  - `joint_state_broadcaster`
  - `joint_trajectory_controller`

5.2 JointTrajectoryController Configuration

- Joints: 6 revolute arm joints
- Command interface: effort
- State interfaces: position, velocity
- PID gains defined per joint

Higher gains were assigned to base joints to compensate for gravity and inertia, while lower gains were used for wrist joints to avoid noise amplification.

---

6. Implementation and Testing Procedure

Phase 1: Robot and Control Description

- URDF defined with accurate inertial and collision parameters
- Transmissions added for all joints
- ros2_control tags included for Gazebo integration

Phase 2: Simulation Integration

- Gazebo launched using `ros_gz_sim`
- Robot spawned from URDF
- `gz_ros2_control` plugin used as hardware interface

Phase 3: Controller Activation

- Controllers loaded after robot spawn
- Joint states verified through `/joint_states`

Phase 4: Trajectory Execution and PID Tuning

- Joint trajectories commanded via `rqt_joint_trajectory_controller`
- PID gains adjusted iteratively
- Joint response monitored in Gazebo
- Reference vs feedback plotted using `rqt_plot`

---

7. Results, Validation, and Self-Evaluation

7.1 Controller-Level Validation Using `rqt_plot`

To validate the effectiveness of PID tuning and trajectory tracking, runtime data was visualized using **`rqt_plot`**. Instead of relying solely on `/joint_states`, signals published directly by the **JointTrajectoryController** were used, as they represent the controller's internal view of the system.

The following namespace was selected:

/arm_controller/controller_state

Within this topic, the parameters plotted were:

- `/arm_controller/controller_state/reference`
- `/arm_controller/controller_state/feedback`

These fields are published by the JointTrajectoryController and are time-synchronized with trajectory execution.

7.2 Rationale for Using `controller_state` Topics

The `/arm_controller/controller_state` topic was chosen for evaluation because:

- It reflects the **exact state used by JTC for control computation**
- It exposes both **reference** (desired trajectory state) and **feedback** (measured joint state)
- It avoids ambiguity introduced by external broadcasters

This makes it the most reliable source for assessing PID behavior and controller performance.

7.3 Relationship Between `reference`, `feedback`, and `/joint_states`

During evaluation, it was observed that:

/arm_controller/controller_state/reference ≈ /joint_states

This occurs because:

- The JointTrajectoryController publishes the current desired trajectory point as `reference`
- The `joint_state_broadcaster` publishes joint positions derived from the same Gazebo simulation state
- When trajectory tracking is accurate, the simulated joint positions closely match the reference

As a result, `reference` and `/joint_states` appear identical during steady-state motion.

7.4 Importance of `feedback` for PID Evaluation

Although `reference` and `/joint_states` may overlap during stable tracking, the `feedback` signal remains critical because:

- It is the signal used internally for **error computation** within JTC
- Any delay, disturbance, or oscillation appears first in `feedback`
- PID tuning effectiveness is evaluated by how quickly `feedback` converges to `reference`

For this reason, the primary evaluation plot used was:

reference vs feedback

7.5 Observed Results

Using the above plots, the following observations were made:

- Smooth convergence of feedback to reference after PID tuning
- No sustained oscillations or overshoot
- Improved transient response with appropriate derivative gain selection

These results confirm correct PID application through the JointTrajectoryController using an effort command interface.

---

8. Challenges Addressed

- Correct interpretation of JTC PID applicability
- Avoiding unsupported position-based PID configurations
- Ensuring stable physics interaction in Gazebo
- Coordinating controller startup timing

Each challenge was resolved by aligning implementation strictly with ROS 2 Jazzy documentation.

Gazebo Simulation Video:

https://meeting.zoho.in/meeting/videoprv?recordingId=2c7d136f95abd534b1a690d34f4f069a21d4341d158662436abe2e8526d68ccc&view=embed