CynLr Report

Visual-Guided Dual-Arm Synchronization with Decentralized Control

Oussama Jaffal

December 2025

Abstract

This report presents a multi-robot simulation framework built in ROS 2 using Gazebo (classic) and MoveIt 2 for motion planning and execution. The system supports heterogeneous manipulators (e.g., UR5 and Franka Panda), namespaced deployments, and task-level primitives (circular motion, Lissajous curve motion, visual tracking, and grasp-by-attachment). We describe the architecture, software components, configuration and launch strategy, and provide validation through tasks executed in simulation.

1 Introduction

Robotic manipulation research and development often requires reproducible simulation pipelines that integrate perception, planning, control, and task logic. The objective of this project is to build a modular simulation environment that can (i) spawn multiple robot arms in a shared world, (ii) run independent motion stacks per robot, and (iii) execute manipulation tasks such as reaching, tracking, and grasping.

Contributions.

- A multi-robot ROS 2+Gazebo simulation and launch framework with serial spawning and per-robot namespacing.
- Task primitives for motion execution: a circular motion, and a Lissajous curve motion.
- A tracking controller that maps perception feedback to end-effector motion in a constrained plane.
- A grasping method in simulation using link attachment once proximity/contact conditions are met.

2 System Overview

2.1 Software Stack

ROS 2 (Foxy); **Gazebo Classic**; **MoveIt 2**: planning pipeline (OMPL), kinematics, trajectory execution; **ros2_control**: joint trajectory control via controller manager (namespaced).

2.2 Robot Models and Namespacing

Each robot instance is spawned under a dedicated namespace (e.g., `/arm1`, `/arm2`). The framework ensures independent controller managers and MoveIt 2 `move_group` nodes per robot. For Gazebo and TF integration, topic remapping conventions are used (e.g., `/tf` $\rightarrow$ `tf` inside a namespace).

2.3 World and Objects

The simulation world includes a ground plane, a graspable cube, UR5 and Framka Emika Panda arms. Object properties include dimensions, inertial parameters, friction, and visual materials.

3 Architecture and Components

3.1 Launch and Configuration

The launch system:

- starts Gazebo server/client with the selected world,
- spawns robots serially to avoid controller-manager namespace conflicts,
- loads controllers (`joint_state_broadcaster`, `arm_controller`),
- starts one MoveIt 2 `move_group` per robot instance,
- starts task nodes/actions (circular motion, Lissajous curve motion, tracking, grasping).

3.2 Motion Execution Primitives

Circular motion. This primitive traces a circular Cartesian path around the current end-effector pose. It is implemented following the `MoveL` action-server design from the IFRA ROS 2 + MoveIt framework, reusing the same waypoint-based planning workflow (Cartesian waypoint generation → `computeCartesianPath` → time-parameterized execution).

Lissajous curve motion. Similarly, this primitive executes a Lissajous Cartesian trajectory using the same `MoveL`-inspired action structure and waypoint planning pipeline, differing only in the parametric waypoint generation.

3.3 Tracking Controller

Detection module

The detection is done using a color filter that only shows the color of the cube which is yellow, and then converts the centroid of the detected object into a normalized coordinates system where the center of the frame is (0,0). Which means the target for the tracking controller is to go to (0,0) each time within a small error margin that can be modified.

Tracking Controller

A controller subscribes to a perception topic (e.g., `/detected_ball`) and drives the end-effector in a constrained plane (e.g., YZ) to reduce image-plane error. The mapping is done as follows :

$$\Delta y = -k_y\, e_x, \qquad \Delta z = -k_z\, e_y, \tag{1}$$

3.4 Grasp-by-Attachment in Simulation

To emulate grasping reliably in simulation, a link attachment mechanism is used:

- the arm approaches an object to a pre-grasp pose,
- a proximity condition triggers an `AttachLink` service call,
- the object becomes rigidly attached to the robot link.

4 Task Definitions

This project targets the following tasks (representative set):

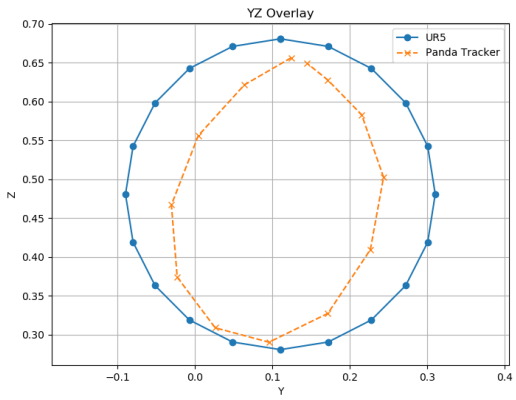
1. **Multi-robot spawn and control**: spawn heterogeneous arms in one world and command each independently.
2. **Reach-to-pose**: move the end-effector to a desired pose or offset, validating kinematics and planning.
3. **Pick (attach) and transport**: reach cube-adjacent pose, attach on proximity, then move object.
4. **Motion execution**: trace a circle or a Lissajous curve motion with controllable resolution and timing.
5. **Vision-guided tracking**: reduce visual error by planar end-effector motion using perception feedback.

5 Experiments and Results

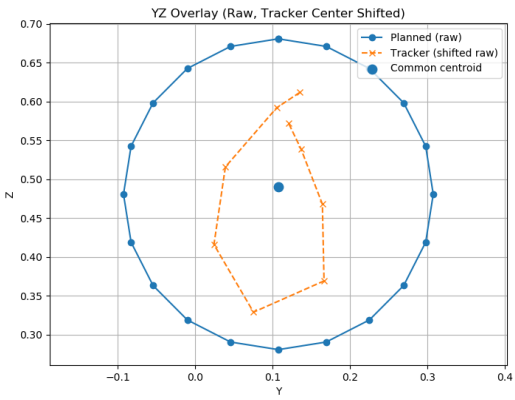
This section reports two trajectory-execution experiments in simulation: (i) a circular end-effector path and (ii) a Lissajous end-effector path. For each experiment, we compare planned Cartesian waypoints from UR5 arm against the tracking end-effector trajectory from the Panda arm, both motions are done using the same number of waypoints for the motion execution of the UR5 arm: 20 waypoints.

5.1 Experiment 1: Circular Trajectory Tracking

Figures 1a–1b summarize the circular trajectory experiment. The left panel shows the planned vs tracked YZ overlay with speed of motors 0.5 for the UR5, and the right panel shows the same with a speed of 1.0 for the UR5. (speed 1.0 means maximum joint velocity/acceleration limits)



(a) YZ overlay (speed 0.5 for UR5 joints).

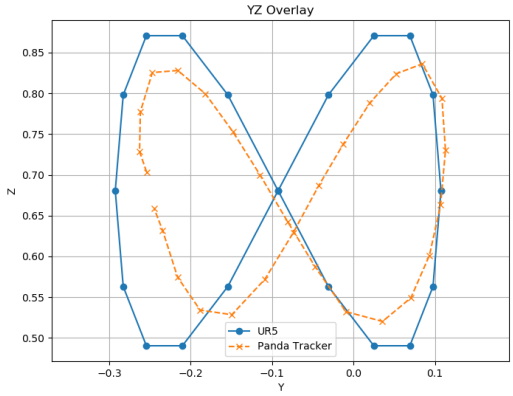


(b) YZ overlay (speed 1.0 for UR5 joints).

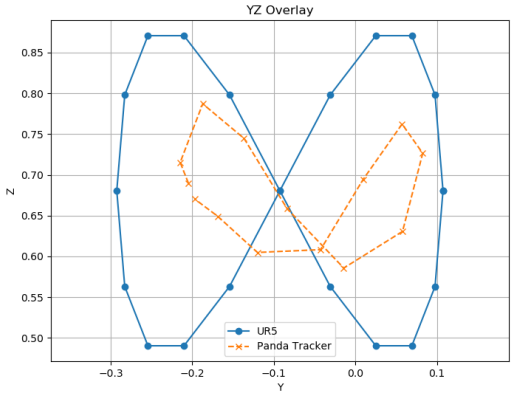
5.2 Experiment 2: Lissajous Trajectory Tracking

Figures 2a–2b–3a summarize the Lissajous tracking experiment. The top row presents the YZ overlay and the time-series plot, left with a speed of 0.2, right with a speed of 0.5. The

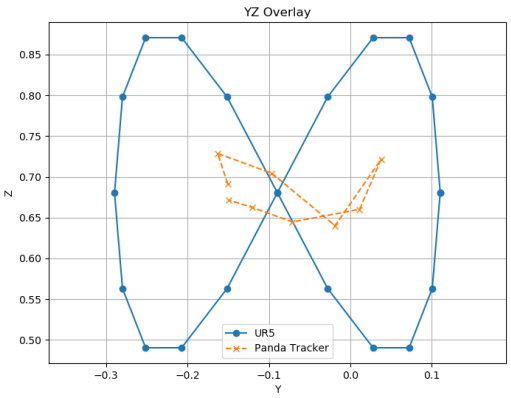
bottom row reports the tracking with a speed of 1.0. (speed 1.0 means maximum joint velocity/acceleration limits for the UR5 arm)



(a) YZ overlay (speed 0.2 for UR5 joints).



(b) YZ overlay (speed 0.5 for UR5 joints).



(a) YZ overlay (speed 1.0 for UR5 joints).

5.3 Discussion

Across both the circular and Lissajous experiments, speeds below 0.5 perform stable tracking that preserves the intended motion, even if the executed trajectory contains slightly fewer effective waypoints at higher speeds near 0.5. In contrast, at 1.0 the tracking becomes significantly less reliable: deviations increase for the circle case, and for the Lissajous trajectory the UR5 execution is notably more challenging and nearly fails to reproduce the planned pattern.

6 Limitations

- The grasping is done using proximity of end effector to the object grasped. A better way was to use the grasping hand of the UR5 and add the controllers of the fingers.
- The tracking controller for Panda arm could be tweaked even further to insure a solid performance even with high speeds + adding trajectory smoothing.
- Due to some internal issue, I had to add a different Panda arm in the simulation which is spawned outside the map. Otherwise the controller for the joints of the Panda arm is not responding.

Acknowledgements

- **IFRA-Cranfield ROS2_RobotSimulation:** action-server structure and MoveIt 2 integration used as the baseline for Cartesian motion primitives.
  `https://github.com/IFRA-Cranfield/ros2_RobotSimulation/tree/foxy`
- **multi_robot_arm:** multi-robot Gazebo/ROS 2 simulation adopted and extended for use of different robotic arms.
  `https://github.com/arshadlab/multi_robot_arm`
- **ball_tracker:** vision/tracking framework.
  `https://github.com/joshnewans/ball_tracker`
- **Gazebo grasping / link attachment:** attachment-based grasping approach.
  `https://github.com/IFRA-Cranfield/IFRA_LinkAttacher`