HW4: Manipulation with ROS 2

Mobile Manipulation 2026 - ShanghaiTech University

Due Date: Refer to the course website

Attention

Before starting this assignment, ensure you have **ROS 2 Jazzy** installed on Ubuntu 24.04, together with **Gazebo Harmonic** and **MoveIt 2**, and a basic understanding of ROS 2 nodes, topics, services, actions, and TF. This is an **individual assignment**. See the provided `README.md` for environment setup, build and launch commands.

Manipulation Overview

The assignment consists of three manipulation tasks on a simulated Franka Emika Panda arm. All three tasks use the same manipulator and the same table; only the arrangement on the table differs.

#	Objective	Scene preset
1	Pick and place, no obstacle	`basic`
2	Pick and place, with a wall	`obstacle`
3	Peg insertion	`insert`

Table 1: The three tasks and their scene presets.

We provide the scene, the robot, the MoveIt configuration, a gripper wrapper, and the hint card in the *Hint Card* section. We do **not** provide any task implementation, framework, or template: program structure, node decomposition, and solution approach are part of the assignment.

Your Task

Assignment package source: [`https://robotics.shanghaitech.edu.cn/gitlab/moma-2026/resources/hw4`]

Common Information

Scene Definition

The single source of truth for the workcell is

`src/arm_scene/config/workcell.yaml`

It contains the dimensions and positions of the table, tray, wall, fixture, block, and peg. Read it before writing any code. Do not attempt to infer dimensions from RViz.

Inspect Gazebo, not RViz

	Shows
Gazebo	The physical simulation — the actual result
RViz	The planning model — what MoveIt expects to happen

Table 2: What each visualisation shows.

The two do not always agree, and several of the intended difficulties of this assignment lie precisely in that difference. Judge the outcome in Gazebo.

Provided Interfaces

Interface	Description
`/arm_controller/follow_joint_trajectory`	Joint trajectory controller (ros2_control)
`/gripper_cmd`	Gripper `GripperCommand` action, provided by `arm_gripper_adapter` (see its own `README.md`)
MoveIt services and actions	`/compute_ik`, `/move_action`, `/compute_cartesian_path`, `/apply_planning_scene`, ...

Table 3: Provided interfaces.

These are conveniences, not a framework. Using them is optional.

Task 1 — Pick and Place

Objective. Pick up the block on the left side of the table and place it in the tray on the right.

Scene. A block on the $+y$ side of the table; a shallow tray on the $-y$ side. Dimensions and positions are given in `workcell.yaml`.

Criterion. The block rests stably in the tray: not on the table, not on the tray wall, not leaning against it.

Task 2 — Pick and Place with an Obstacle

Objective. As Task 1, with a wall standing between the pick and the place locations.

Scene. Task 1 plus a wall. Its dimensions, position, and in particular its height and horizontal extent are given in `workcell.yaml`. Together they determine which routes are available.

Criterion. The block rests stably in the tray, and during the motion the wall is neither displaced nor contacted.

Task 3 — Insertion

Objective. Pick up the peg standing on the table and insert it into the square hole of the fixture.

Scene. A fixture with a square hole, and a peg standing next to it. The hole is larger than the peg by approximately 1 mm per side, and its entrance is chamfered at $45°$. Dimensions are given in `workcell.yaml`.

Criterion. The peg stands in the hole: not leaning on the rim, not fallen beside the fixture.

Hint Card

The items below concern the environment rather than the tasks. They are provided because they would otherwise consume a disproportionate amount of time. Attempt the tasks first and consult this section when blocked. The items state the direction only; the method remains part of the assignment.

Robot

1. **The initial pose of the arm is not the zero configuration.** If the arm appears fully extended and vertical, that is not the initial pose but a degraded one: a singular configuration whose forearm intersects objects on the table. The symptom resembles a controller fault — commands appear to have no effect. Do not attempt to correct it; it is configured. If it occurs, something has displaced the arm.

2. **Grasp height should not be derived from geometry alone.** The physics engine applies an additional collision thickness of approximately 10 mm per side to unmodified mesh geometry, so contact occurs before the geometric surfaces meet. A grasp height computed geometrically may cause the fingers or the palm to contact the table early. This has been corrected for the fingers; the palm and the arm links still carry it.

3. **A grasped object must be attached to the planning scene.** Otherwise the planner is unaware that the gripper is holding anything and will produce paths that pass through obstacles, while the simulation collides with them. The result is a plan that reports success and an execution that fails. In MoveIt this operation is called *attach*.

4. **The gripper must face downward when grasping.** The orientation quaternion must not be a pure yaw rotation, which leaves the palm facing upward. The correct form is a $180°$ rotation about x, followed by the yaw rotation about z. The failure is subtle: inverse kinematics without collision checking succeeds, while inverse kinematics with collision checking reports `-31`.

Planner

5. **Identical motions may produce different trajectories on successive runs.** This is expected: the default planner (RRTConnect) returns the first solution it finds and does not optimize. In one measurement the peak tool height of the same transfer differed by more than 40 cm between two runs. A single run is not sufficient evidence for any conclusion.

6. **Placing an object exactly flush against a surface will fail.** `compute_cartesian_path` evaluates inverse kinematics point by point; the final points are rejected as collisions and the path stops short. Leave a small clearance and release the object.

7. **The planner routing over an obstacle is normal behaviour.** Joint-space planning can raise the tool to any reachable height, so an obstacle taller than the carry height only blocks straight-line motion. To constrain the route, either make it geometrically impossible or require a sufficient clearance margin. The planner optimizes for collision avoidance and cost; it does not follow intent.

Submission

Submit to your homework repo in a folder named HW4.
Submit your code and a report:

1. Your programs, building successfully with `colcon build`.

2. A short report: the approach taken for each task, the difficulties encountered, and how you determined that the task had succeeded.

3. An observation of each task in Gazebo (a short video is fine).

4. A short note on your use of AI for this assignment: which services and software you used, how much work it saved you, and a few example prompts.

A numerical report is not required.

Good luck, and happy coding!