UNIVERSITÀ DEGLI STUDI DI PADOVA

ROS2 PICK & PLACE

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

Second Assignment Report

Alessando Chinello [†], Mattia Scantamburlo [†], Luca Piai [†] [*]

February 4, 2026

1 Objectives

The objective of this assignment is to perform a pick-and-place task within the IAS lab simulation. Following the detection of AprilTags via a dedicated camera, the UR5 robotic arm is controlled to swap the objects' position.

2 Code structure

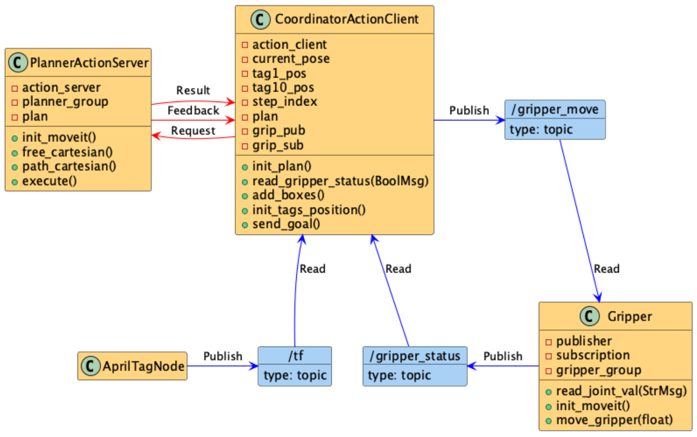

FIGURE 1: Simplified class diagram.

In the following we briefly describe each implemented node. Any reader can access the full code at our GitHub.

2.1 PlannerActionServer

This node implements the core logic for arm motion planning and execution within the *MoveIt 2* framework. It acts as an *action server*. The *action client* needs to specify the **target pose** that it wants to reach and how it wants to reach it (move_type parameter). The server gives feedback about the current end-effector pose and it provides the final one when the goal is completed.

Planning Modes The client needs to specify move_type parameter in the the goal request. This parameter selects one of following two planning modes.

- **Free Cartesian Motion**: implements the standard point-to-point (PTP) movements. The planner tries to find a path to reach the target pose. The trajectory is not specified by the client, the only constraints on the motion are given by the surrounding environment.

- **Linear Cartesian Path**: specifically designed for precise phases such as object *approach* or *retreat*. In this case the planner computes a linear interpolation in Cartesian space between the current end-effector pose and the target.

Regardless the mode, once the planning phase is done, the node immediately execute the requested motion.

2.2 Gripper

This node manages the actuation of the Robotiq 2-finger gripper through a subscriber/publisher architecture integrated with the *MoveIt 2* planning interface. This component has a single responsibility: opening and closing the gripper. More specifically, it reads the radian value (published at `/gripper_move`) that specifies the target position of the gripper joint. When the motion is completed, the status (*true* or *false*) is published at `/gripper_status` topic.

2.3 CoordinatorActionClient

The `CoordinatorActionClient` node serves as the central orchestration unit of the system, responsible for the high-level sequencing of the pick-and-place task.

Scene Configuration At startup, the node performs an initialization phase in which two operations are performed:

1. **AprilTags Localization**: `AprliTagNode` detects the position and publishes them into `/tf` topic. The coordinator node stores the two positions.

2. **Adding Collision Objects**: four collision boxes are added in the position occupied by the tables and parallelepipeds below the AprilTags.

The presence of the collision boxes, ensure that all subsequent motion requests sent to the Planner node are computed within a collision-aware environment. Figure 2 shows the scene after performing the two operations.

[*][†]Department of Information Engineering, University of Padova, email:
alessandro.chinello.2@studenti.unipd.it,
mattia.scantamburlo.1@studenti.unipd.it,
luca.piai.1@studenti.unipd.it

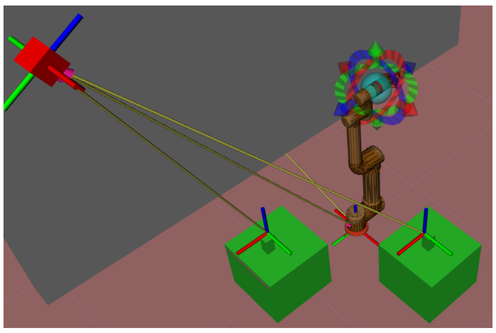

FIGURE 2: Starting scene configuration from rviz.

Solving the task step by step The pick and place task is defined as a sequence of steps: poses that the end-effector needs to reach. Each step is an *action goal* that the `CoordinatorActionClient` sends to the `PlannerActionServer`. Between these steps, the Coordinator communicates with the `Gripper` node whenever required.

3 Execution

3.1 High level view

In this subsection we present a sequence diagram that explains the interactions between nodes in our system. As we

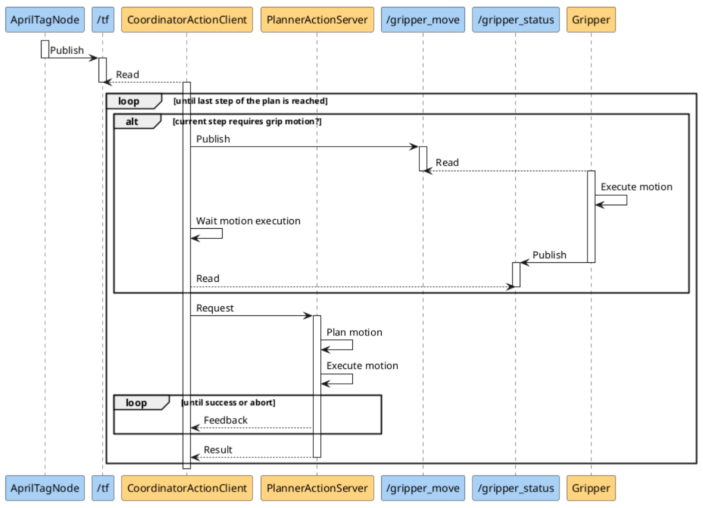

FIGURE 3: Sequence diagram.

can observe from Figure 3, the Coordinator node manages the communication with the Gripper and the Planner in an iterative way. At each iteration, the Coordinator first checks whether the current step requires a gripper motion. If so, the target value for the gripper joint is published on the appropriate topic, and the Coordinator waits until the execution is completed. It then initiates communication with the Planner by sending a request; the next iteration does not begin until the corresponding result is received.

3.2 When the goal is aborted

The action goal requested by the Coordinator node, could be aborted for two reasons:

1. The Planner node is not able to find a feasible joints' trajectory that solves the PTP problem for the specified target point.
2. The Planner computes a linear trajectory that is associated to a percentage[1] lower than 85%.

By design, the Planner tries to solve the PTP problem at most 10 times. If all attempts fail, the Planner aborts the goal. Instead, when the client requests a linear Cartesian path, then only one computation is made before aborting (since the failure in this case only depends on the target position).

From the Coordinator's point of view, the logic is straightforward: regardless of the reason for the abort, the same goal is resent. Each step can be requested at most three times; after that, the Coordinator moves on to the next step.

3.3 Brief description of the steps

The task of pick and place and safe swap is decomposed in 13 steps:

- **Step 1**: approach tag1 (the red one).
- **Step 2-3**: grab tag1.
- **Step 4-6**: move towards the other table and drop tag1.
- **Step 7-9**: approach tag10 and grab it.
- **Step 10-12**: move towards the empty table and drop tag10.
- **Step 13**: return to starting position.

The first and the last steps use a PTP motion, while all intermediate steps are based on linear trajectories. In the list we've grouped intermediate steps to keep things compact; however those steps are simple motion (mainly along one axis) performed with respect to the current position.

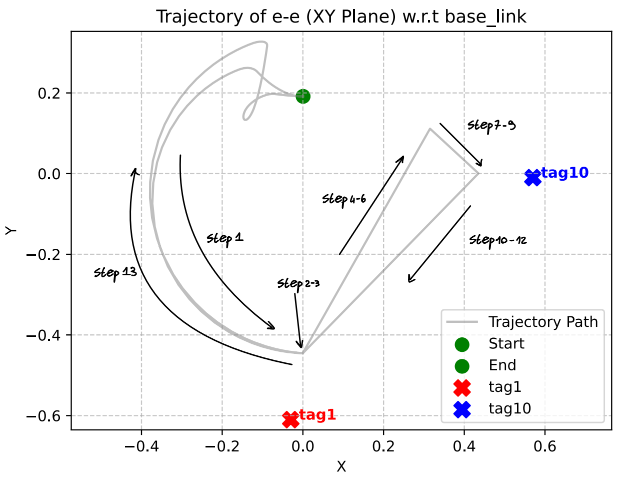

FIGURE 4: Steps 1-13: result of a simulation.

4 Results

The simple pipeline that we've developed is able to solve the pick and place and safe swap task. However our solution has some weak points and in this section we want to focus on these.

[1] The percentage indicates how much of the requested trajectory was successfully followed.

The slow start/end issue This problem is related to the Planner, more specifically when the client request a free Cartesian motion. In our solution we're not giving any constraints on the joints' value, the only constraints are the collision objects added in the scene. As a result, the Planner may spend some amount of time attempting to solve the PTP problem. The required computation time also depends on the available CPU resources.

Closing gripper may cause freezing When running the simulation it may happen that the arm freezes for a few seconds after grabbing one of the AprilTags. Based on our experience, this issue appears to be related to insufficient CPU performance, as it does not occur when running the simulation on a 12-core machine.

Dropping point of tag1 In this step, the Coordinator requests a linear trajectory from the current position—located above the detected red AprilTag—to the designated dropping point. If the percentage of the path computed by the Planner is below 100%, the arm may collide with tag1 while attempting to grasp the blue one. This happens because we chose to grasp objects from the side rather than from above: grasping from above is not feasible in the simulation since the objects tend to slowly slip out of the gripper. This issue may prevent the task from being completed; however, we did not find a fix, as it occurs rarely and was therefore difficult to reliably reproduce and analyze.

5 How we organized the work

As for the previous assignment, the development process was done collaboratively in Discord sessions, where all three members of this group were present and actively working. All the decisions, implementations and choices were discussed together. The workload was divided equally among us. Still the commits on the GitHub repository were often pushed by the member who happened to be writing the code at the moment. For the approach, we adopted a method we are very familiar with: a pseudo-agile workflow, in which we iteratively refine the solution together.

6 Conclusions

In this report we've presented a working solution to the proposed task. The solution that we've developed can be used as a starting point for further improvements.

As an example, the Coordinator could be improved to better handle abort signals sent by the Planner. If a goal is aborted, the Coordinator could define and execute an alternative plan (a *plan B*) for that step, rather than simply skipping it. Additionally, the slow start and end behavior could be addressed by providing more appropriate constraints to the Planner.

7 Usage of AI

In the early stages, large language models (Google Gemini) were primarily employed to obtain code examples related to the MoveIt library. Additionally, the project logo was created using Sora. Finally, an LLM was used to review and improve the grammar of this document.

8 Links

All the required links:

- GitHub repository [2].
- Video[3].

Emails:

- `alessandro.chinello.2@studenti.unipd.it`
- `mattia.scantamburlo.1@studenti.unipd.it`
- `luca.piai.1@studenti.unipd.it`

[2]`https://github.com/Ultimi-Sumiti/Assignment_2`

[3]`https://drive.google.com/file/d/1_Fhm85_CfsbAiJNpVfZhpgZhbcuu7b_Z/view?usp=drive_link`