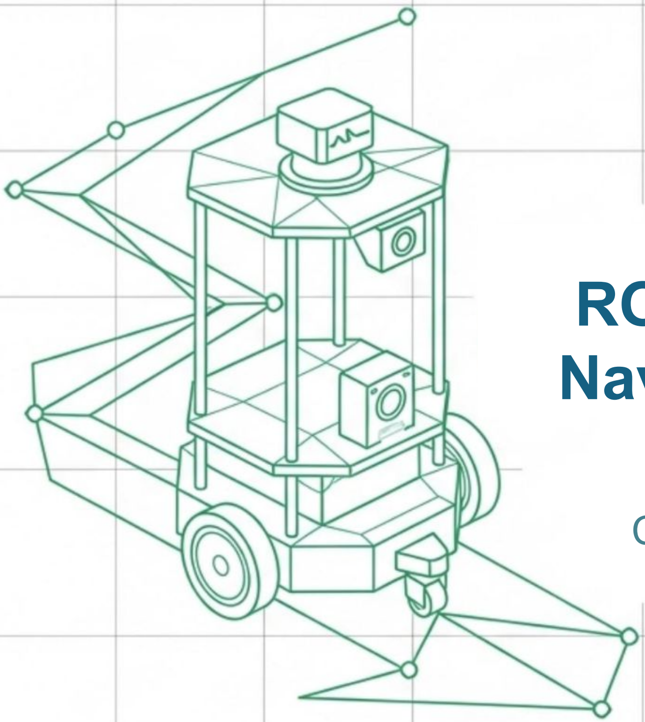
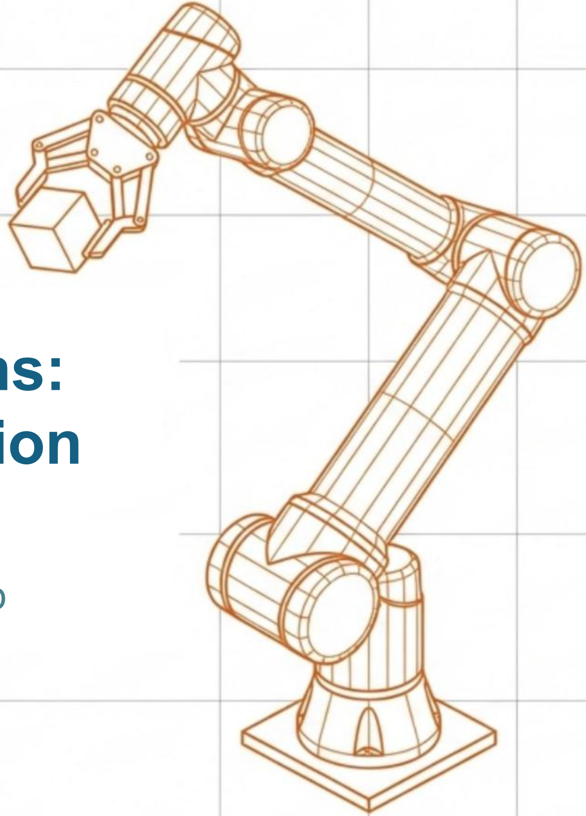
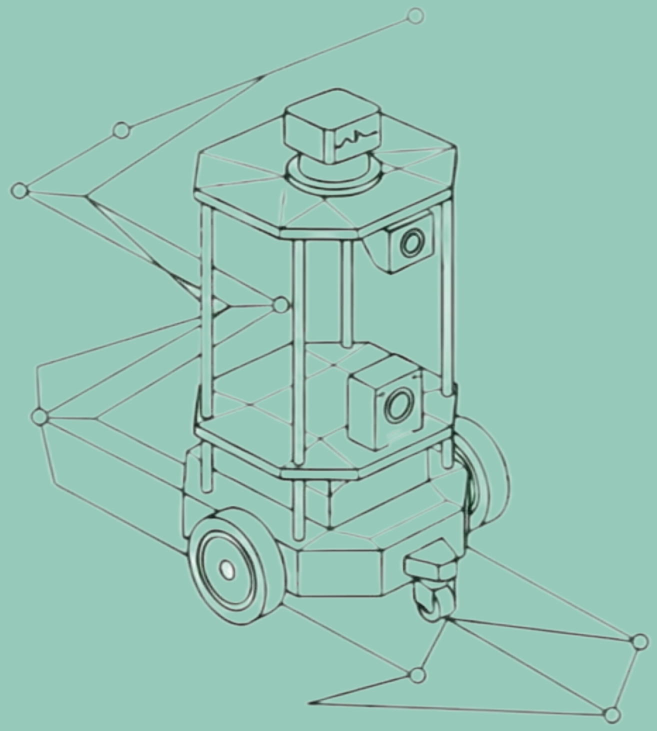


ROS2 Robotics Systems: Navigation & Manipulation Implementation

Comprehensive Report on IAS Lab Simulated Environments

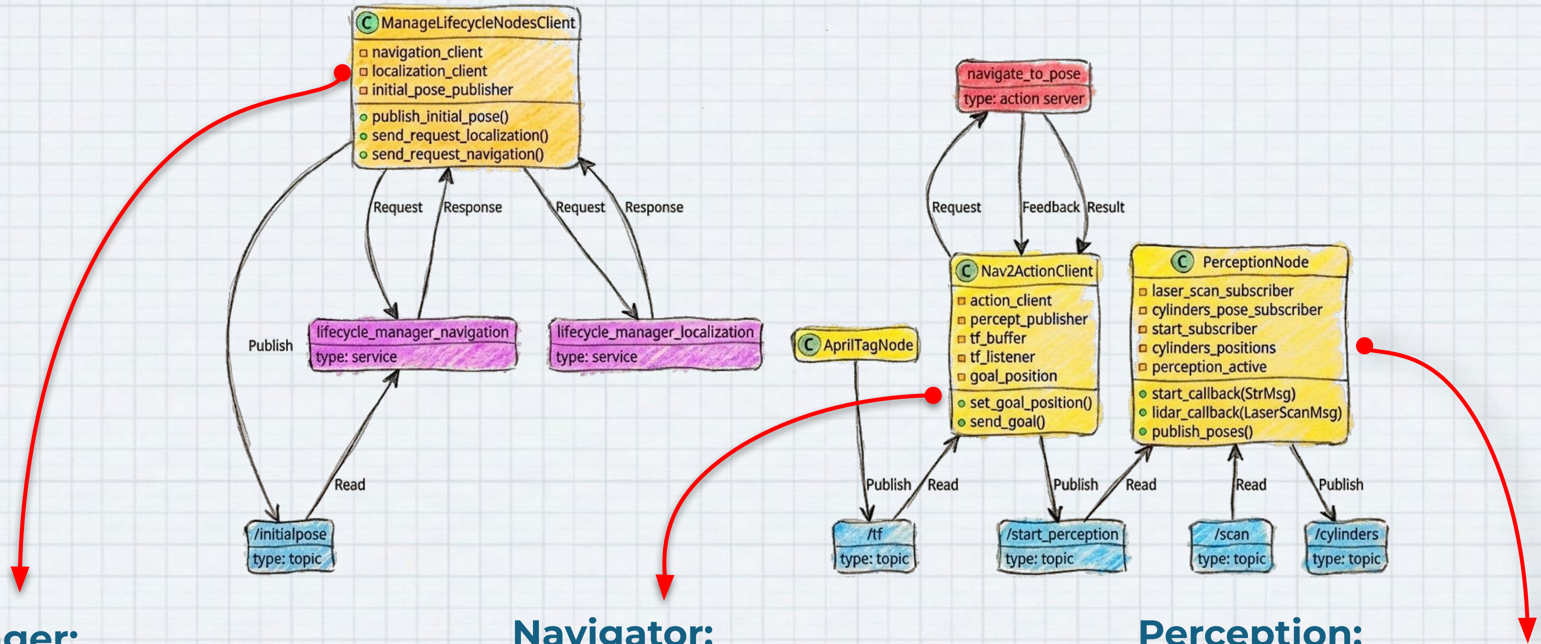


Alessandro Chinello, Mattia Scantamburlo, Luca Piai
University of Padova



Assignment 1:
Navigation, Position, Detection

Architecture: The Navigation Pipeline



Manager:

- Interfaces with localization and navigation
- Publishes on *\initial_pose*

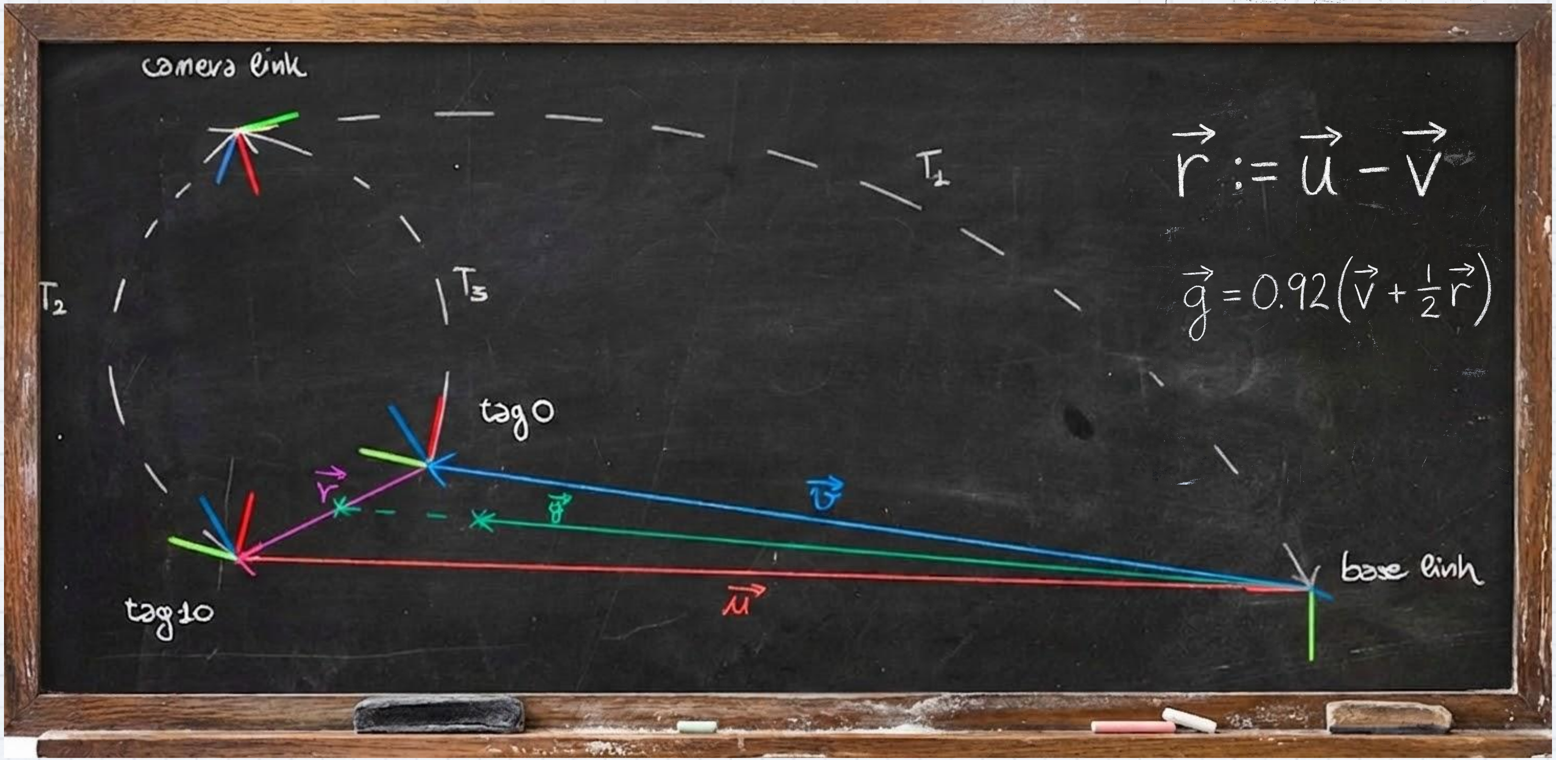
Navigator:

- Calculate target position
- publish on *\start_perception* topic

Perception:

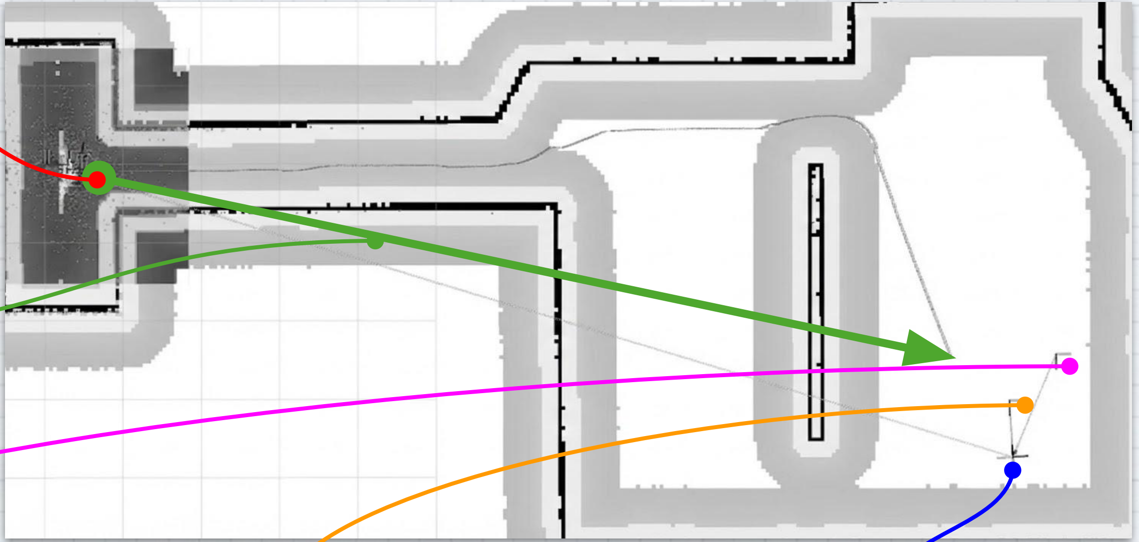
- receive data from *\scan*
- detects cylinders
- publish coordinates of detections

Final position computation



Navigation & Localization

- Initial pose
- Vector (g)



- April Tag 10
- April Tag 0
- Camera

Perception: From LiDAR Scans to Object Detections

Raw Data Conversion

Map raw data coming from LIDAR to pixel of an image

Line removal

Apply HoughLines to remove straight lines (walls)

Data Enhancement

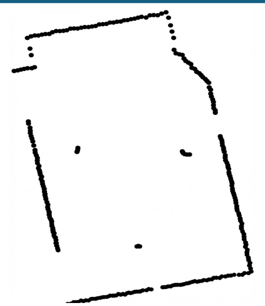
Apply Gaussian Blur to sparse LiDAR points

Circle detection

Apply HoughCircles to identify circles centers from blurred points

The Detection Pipeline

Input Image



The image representing the raw data from *\scan* topic

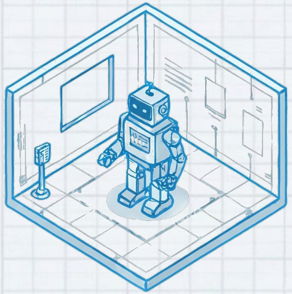
Gaussian smoothing & Straight line removal

Result of removing straight lines and applying Gaussian filter

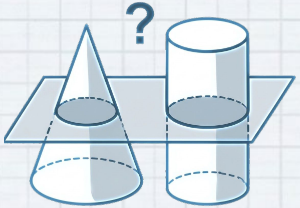
Circle detection

Finally detect circles, draw centers and publish in \cylinders

General problems that we have tackled

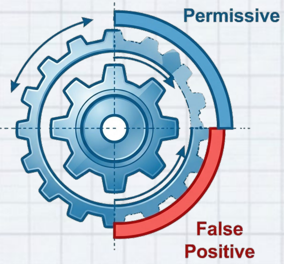


Environment Specificity: The current solution is **highly tailored** to the test environment, limiting generalizability.

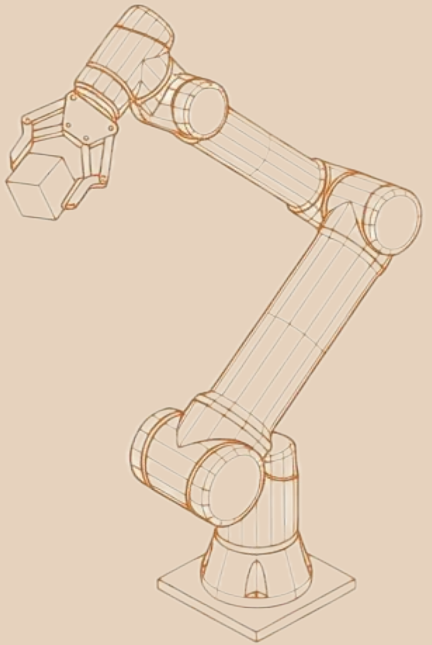


Dimensional Constraints: Relying on **2D LiDAR** prevents the distinction between different 3D shapes with similar circular cross-sections (e.g., cones vs. cylinders).

Data Sparsity: Limited available data points required a **high tolerance** in the detection algorithm.



Algorithm Sensitivity: **HoughCircle** function was configured to be "permissive" to catch partial arcs. **Consequence:** Increased risk of false positives (e.g., misidentifying parts of large cones as cylinders).



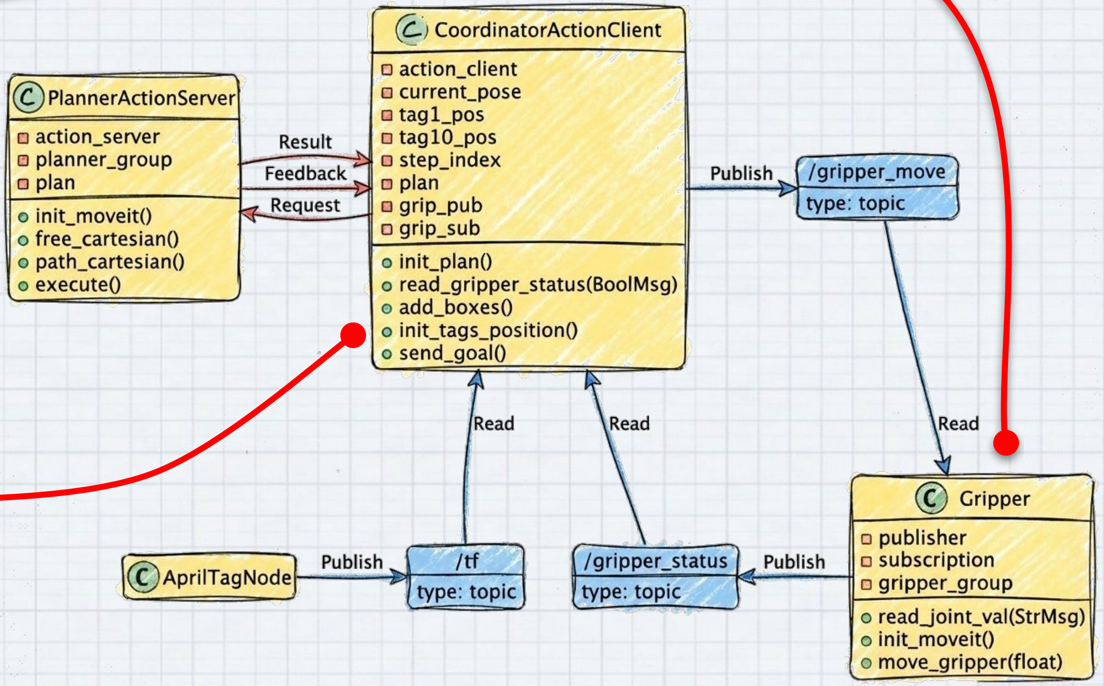
Assignment 2: Pick and place

Architecture: The pick and place pipeline

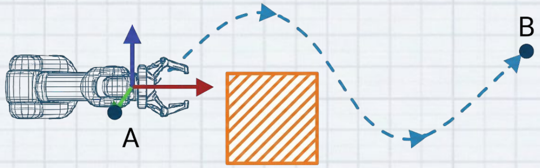
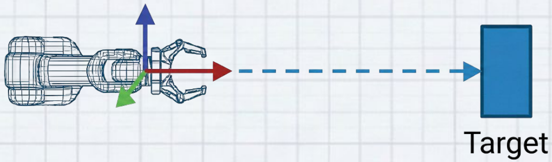
Gripper Node:
open and closes the gripper

PlannerActionServer:
computes and executes the plan to reach the given target pose

CoordinatorActionClient:
responsible for the high-level sequencing of the pick-and-place task.



Comparison between cartesian and free path

Free Cartesian Motion (PTP)	**Use case:** The planner automatically finds the best path to the goal, constrained only by environmental obstacles; used for general, flexible movements.	**Visual concept:** 
Linear Trajectory	**Use case:** The planner computes a linear trajectory starting from the current position to the target one. Used in general for simple movements like approach and retreat.	**Visual concept:** 

Coordinator node

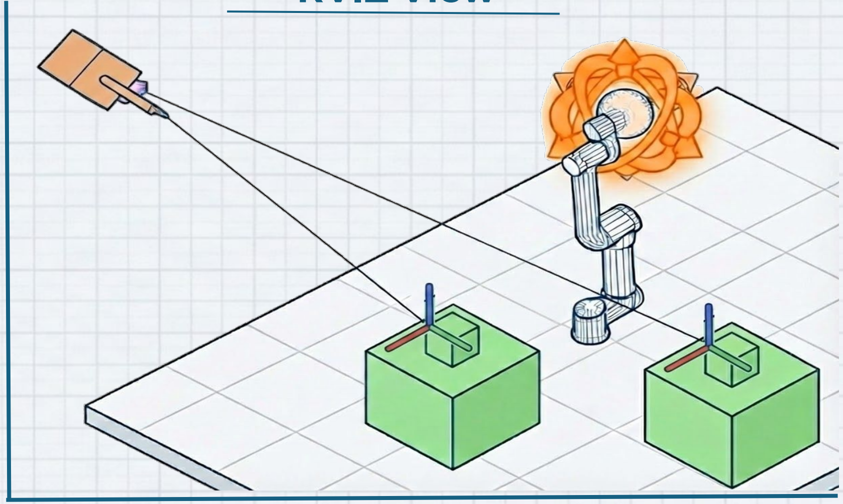
1. Setup Phase

- **Localization:** Stores target positions via AprilTag detection (*/tf*).
- **Collision Mapping:** Adds virtual obstacles for safe, collision-aware planning.

2. Task Execution

- **Sequencing:** Breaks the mission into a series of end-effector poses.
- **Action Control:** Syncs goal requests between the **Planner** and **Gripper**.

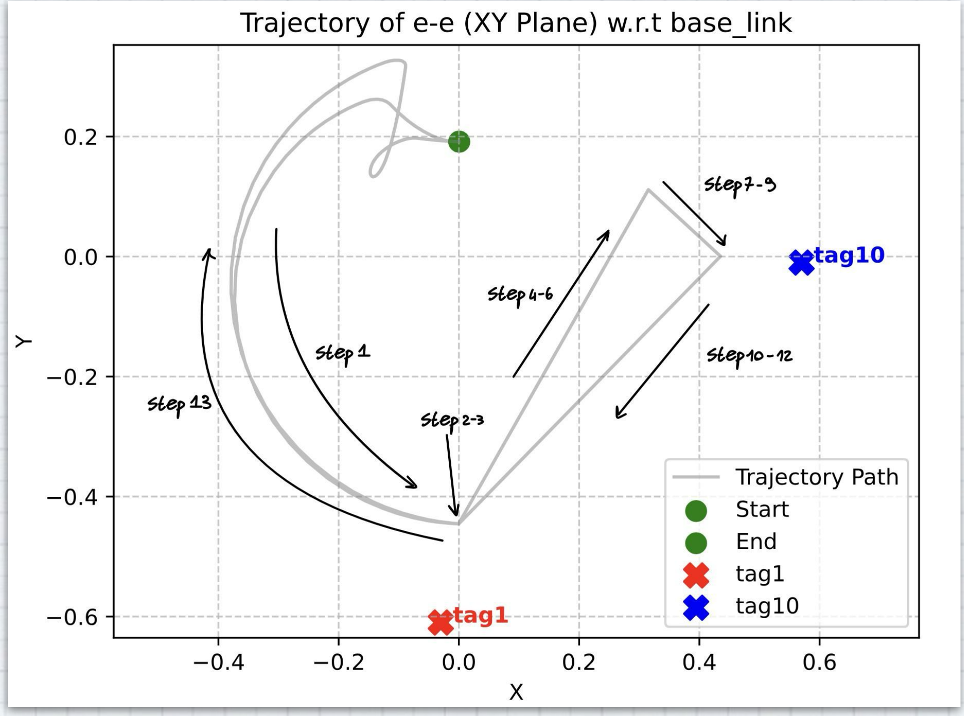
RVIZ View



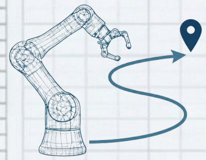
- **The Central Orchestrator:** A high-level unit that sequences the pick-and-place task by managing communication between the Planner and the Gripper.

The 13 Steps & The End-Effector Trajectory

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

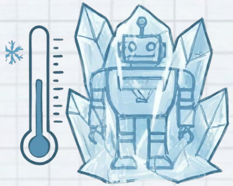


General problems that we have tackled



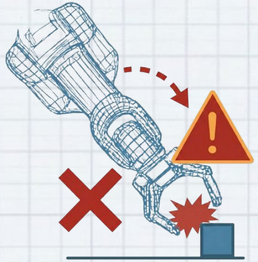
Planner Latency (Slow Start/End): Occurs during unconstrained Cartesian motion requests.

The PTP (Point-To-Point) solver requires significant time/CPU to find a valid path within collision constraints.



Simulation Freezing: Occurs during "Close Gripper" commands after tag detection.

Linked to **CPU bottlenecks** (performance stabilizes on higher-end 12-core hardware).



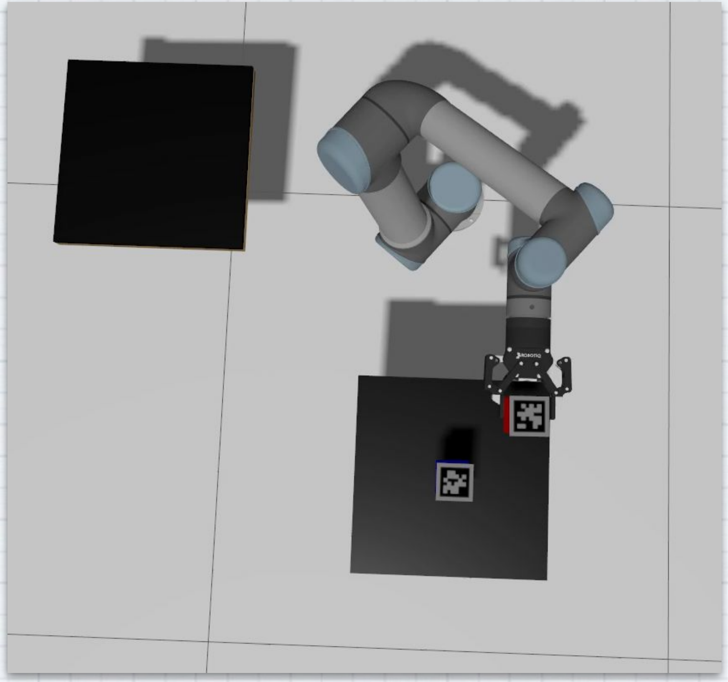
Trajectory Completion Issues: Linear trajectories to dropping points occasionally fail to reach 100% completion.

Side-Grasping Risk: Since objects are grasped from the side (to prevent slipping), incomplete paths can lead to collisions with adjacent tag (see next slide).

Note: this issue is rare and difficult to reproduce..

Possible collision with Tag 10

Dropping Tag 1



Picking Tag 10