# ROBT403 Laboratory Work 3

***Index Terms*— ROS, Inverse Kinematics, Forward Kinematics, 3 DOF planar manipulator**

## I. Introduction

In this report, we document the process of configuring and controlling a robotic system using MoveIt and ROS, focusing on tasks that demonstrate motion planning and execution. The goal was to create and customize a MoveIt package for a specific robot, simulate its behavior in Gazebo, and implement code to move the end-effector along specified paths. The tasks involved setting up the MoveIt library, moving the end-effector by defined distances along the X-axis, and programming the robot to trace shapes such as rectangles. These exercises provided hands-on experience with robotic motion planning, simulation, and programming within the ROS framework.

## II. Theory

For robotic manipulators, inverse kinematics (IK) translates a desired end-effector pose (position and orientation) in Cartesian space into joint angles. For manipulators with more than three degrees of freedom, calculating these joint angles becomes increasingly complex due to the additional freedom and redundancy in possible joint configurations.

In 3-DOF systems, solving IK analytically is often possible since there are fewer joints and constraints. Hence, we were able to solve the kinematics equations using algebraic in the previous laboratory work. It involved 3-DOF manipulator description with trigonometric relationships that are typically always solvable using direct mathematical formulas.

Conversely, in 5-DOF systems, the increased joint number makes analytical solutions impractical or infeasible. MoveIt uses iterative, numerical approaches (such as Jacobian-based methods) to solve for joint angles, especially when additional constraints like joint limits or collision avoidance are considered. Therefore, we utilized Rviz (moveit) to calculate the path and execute it Gazebo simulation through built-in kinematics solvers that support 5DOF configurations. With Moveit's Kinematics Solver Plugin (such as KDL), we found feasible joint solutions to achieve a desired pose by iterating over possible configurations. This iterative solution differs from the more straightforward algebraic methods suitable for 3-DOF systems.

Additionally, it was different with Cartesian Path Planning as well. In MoveIt, the *computeCartesianPath* function enables precise Cartesian control of the end-effector's trajectory. This function allows MoveIt to interpolate between points along a Cartesian path, ensuring smooth, continuous end-effector movement—critical for tasks requiring precise control in the Cartesian space, such as assembly or complex motion tasks.

Finally, while for 3-DOF systems, redundancy is rarely an issue because of the fewer number of possible configurations, 5-DOF robots need additional decision criteria to select an optimal joint configuration. MoveIt addresses this with optimization plugins, which allow the selection of configurations based on constraints like minimizing energy consumption, staying within joint limits, or avoiding obstacles. These criteria are defined in MoveIt's planning scene and IK solver configurations.

## III. Methodology

### A. Task 1

In task 1, the primary objective was to configure the MoveIt library to enable robot control and simulation. This was achieved by creating a custom MoveIt package named moveit-aidana using the MoveIt Setup Assistant. The setup process involved defining the robot's URDF model, configuring its planning groups, setting up kinematic solvers, and generating necessary configuration files for MoveIt integration. This step-by-step approach ensured that the MoveIt package was tailored to the specific robot model and its capabilities.

To verify the configuration and run the simulation, we used the command "roslaunch gazebo-robot moveit-gazebo.launch" to launch the Gazebo simulator with the robot model. Additionally, the command "roslaunch moveit-aidana moveit-planning-execution.launch" was executed to initiate MoveIt in RViz, which enabled planning and execution of the robot's movements. Through this process, we were able to control the robot in RViz and simulate its behavior in Gazebo, using the "Plan and Execute" feature to move the robot to desired positions.

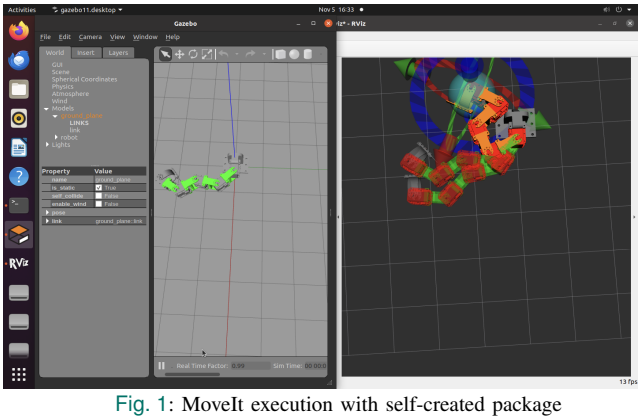

Fig. 1: MoveIt execution with self-created package

### B. Task 2

In task 2, the objective was to create a ROS node using MoveIt that moves the robot's end-effector by 1.4 units along the x-axis, as visualized in RViz. The provided code accomplishes this task by utilizing the MoveIt MoveGroupInterface to plan and execute the movement. The task began with initializing the ROS node and setting up the move-group for the planning group named "move-aidana", which corresponds to the specific robot configuration in the MoveIt setup.

In Task 2, a miscommunication in the lab manual initially prevented successful movement of the robot in the Gazebo simulation. The manual instructed creating a controllers_real.yaml file in the moveit_zhanat/config directory and copying the configuration from the controllers_real.yaml file found in your_workspace/src/snake-noetic/moveit_arm/config. The copied content included details specific to a different robot group and controller structure, which were incompatible with our simulation setup.

However, after further examination, it was determined that the appropriate configuration should actually follow the structure found in the controllers.yaml file within snake-noetic, moveit_arm and then inside of config folder. This file contained the correct controller list and joint definitions that aligned with the current Gazebo and MoveIt simulation requirements. By updating controllers_real.yaml to reflect the correct setup, communication between MoveIt and Gazebo was restored, enabling the robot to move successfully in the simulation environment.

The code first captured the current pose of the end-effector using move-group.getCurrentPose(). This pose was stored as the current-pose, and a target-pose was defined by modifying the x position to move the end-effector 1.4 units backward along the x-axis (by subtracting 1.4 from the current x-coordinate). The move-group.setApproximateJointValueTarget(target-pose) method was used to set the target pose for the robot's end-effector, and move-group.move() executed the motion command.

To ensure the move was completed successfully, a loop was implemented where the code continually checked the current pose against the target pose with a tolerance of 0.01 units. If the difference between the current x-coordinate and the target x-coordinate was within this tolerance, the loop terminated, indicating that the end-effector reached the desired position. The loop was controlled by ros::Rate loop-rate(50) to maintain consistent pose checking and smooth execution.

This approach highlighted the fundamental principles of using MoveIt for basic Cartesian path planning and movement, demonstrating how to adjust and execute end-effector movements along a single axis. The task required modifying the initial script by adjusting the movement value, showcasing how to adapt existing MoveIt-based scripts for different motion requirements.

### C. Task 3

The task involved using the MoveIt framework within the ROS (Robot Operating System) to command a robot's end-effector to follow a rectangular trajectory. The objective was to control the robot's movement step-by-step to draw a rectangle by planning and executing movements between specified waypoints in Cartesian space. The process began with initializing the ROS node and creating a MoveGroupInterface object for the planning group "move-aidana", which is configured in the MoveIt setup for the specific robot. The initial pose of the end-effector was captured using move-group.getCurrentPose(), and this pose was used as a reference for defining subsequent waypoints.

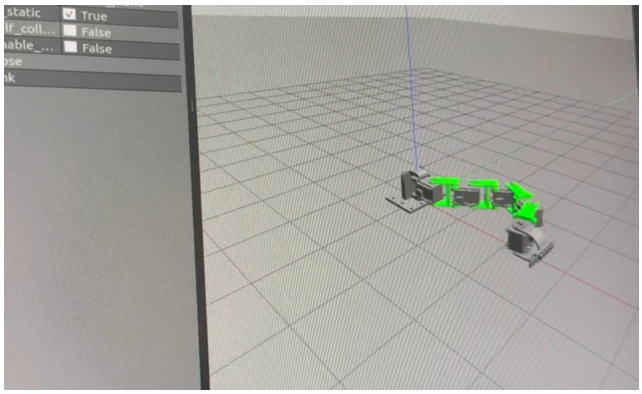

Fig. 2: Robot moving by 1.4 units in a straight line

The code directed the end-effector to move through four main waypoints to complete the rectangle: moving left along the negative x-axis by 1.4 meters, moving down along the negative y-axis by 0.7 meters, shifting right along the positive x-axis by 0.7 meters, and finally moving up along the positive y-axis to return to the starting y-coordinate. For each step, move-group.setApproximateJointValueTarget(target-pose) was used to set the target pose, and move-group.move() executed the movement. Each movement was verified by continuously checking the current pose to ensure that the end-effector reached the target within a 0.01-meter tolerance. The process repeated in a loop controlled by ros::Rate loop-rate(50), ensuring smooth and regular position updates.

This method proved effective for creating simple paths by defining and executing waypoint-based movements while checking pose accuracy. It illustrated how Cartesian path planning in MoveIt can be used for straightforward shapes like rectangles. The approach emphasized precision by incorporating pose verification at each step, thus minimizing potential cumulative errors. This task demonstrated MoveIt's capability to handle path planning and execution for simple geometric paths and laid the groundwork for expanding to more complex paths by adding further waypoints or adjustments.

### D. Execution and Testing

For task 1, run "roslaunch gazebo_robot moveit_gazebo.launch" and "roslaunch moveit_aidana moveit_planning_execution.launch" in seperate terminals to control robot in RViz. Next, for task 2 run "rosrun lab4 test_moveit" to move the end-effector by 1.4 units and "rosrun lab4 test_rectangle" for task 3 to draw a rectangle with the end-effector. The video of the execution can be found at https://youtu.be/ilK6y-SxkeI?si=a1Oz422DMIwhTE3Z and files at https://github.com/asemqr/RoboticsLabs/tree/LAB4

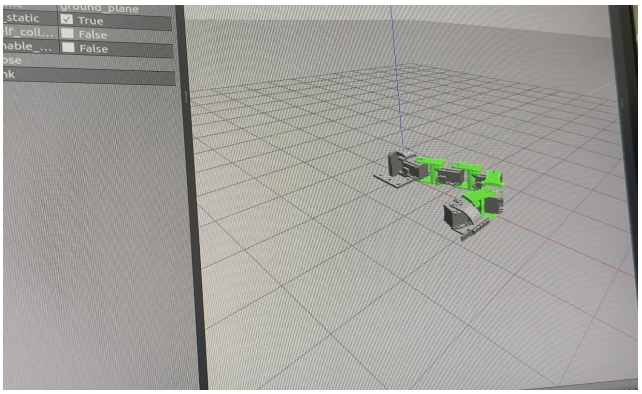


Fig. 3: Robot drawing a rectangle