ROBT 403 LAB 4: Moveit Library for Cartesian Pose Control

Dinmukhamet Murat

***Abstract*—This report presents the configuration and implementation of the MoveIt library for Cartesian pose control of a robotic manipulator. The aim is to set up a real robot and its virtual counterpart using MoveIt and control its end-effector using Cartesian commands. This includes configuring MoveIt, establishing joint trajectory controllers, and executing custom movements along Cartesian coordinates. Tasks include moving the end-effector along the X-axis, drawing a rectangle, and implementing optional movements like a circle.**

***Index Terms*—3-DOF Planar Manipulator, Cartesian Pose Control, Gazebo**

I. Introduction

The MoveIt library, a popular motion planning framework for ROS, provides a suite of tools for developing and executing complex motion plans, specifically enabling capabilities like inverse kinematics, trajectory planning, and end-effector control. In this lab, the focus is on leveraging the MoveIt library to configure a robotic arm for Cartesian pose control, where direct manipulation of the end-effector's position and orientation in space is essential for performing specific tasks.

The setup process entails integrating essential ROS packages, configuring URDF (Unified Robot Description Format) files, and establishing control parameters suitable for position-based controllers rather than effort-based ones. Through configuring real and simulated controllers, including the setup of JointTrajectoryControllers, the lab enables the robot to operate in both virtual and real scenarios.

II. System setup

The robotic system used for the lab consists of a 5-DoF planar manipulator, where only three joints are active. We utilized ROS Melodic for communication and control, along with necessary ROS packages such as dynamixel_motor and moveit_arm for controlling the robot.

A. Environment Preparation

Set up the ROS workspace and download the required packages. Clone necessary GitHub repositories:

git clone https://github.com/arebgun/dynamixel_motor.git
git clone https://github.com/KNurlanZ/snake-noetic.git

B. MoveIt Configuration and Setup Assistant

The MoveIt Setup Assistant was runned to create a new MoveIt configuration package with loading moveit_robot.urdf.xacro. According to the instructions self-collisions, planning group named move_dimash and pre-defined initial pose were generated. The final package was named moveit_dimash and check with the following command: roslaunch moveit_dimash demo.launch

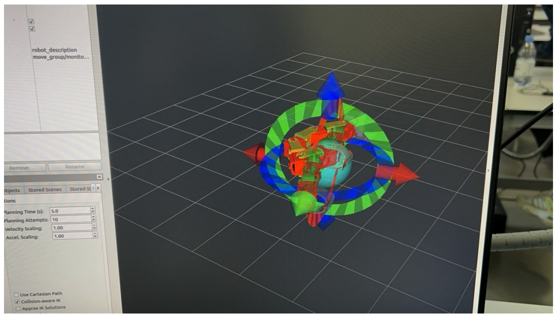


Fig. 1: Rviz of real robot

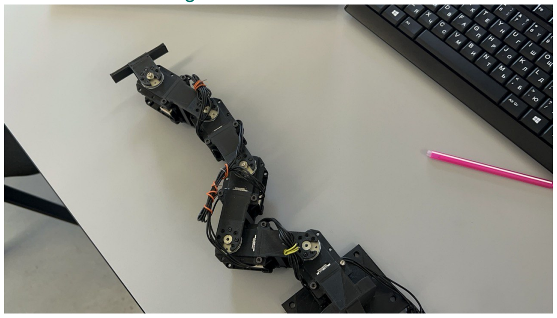
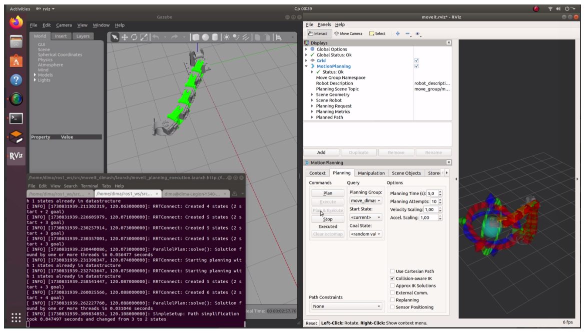


Fig. 2: Movement of real robot

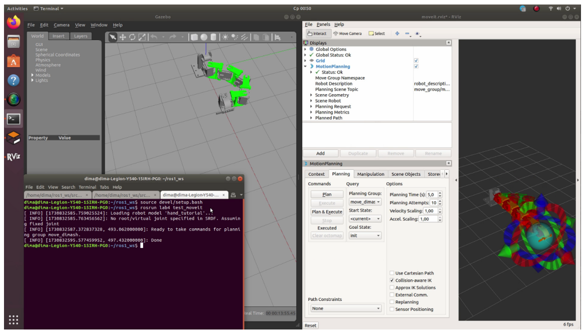
III. Task1: ROS MoveIt configuration for Planar Robot

After the completion of the package, first real controllers were connected with MoveIt with new yaml file named: trajectory_controllers_dimash.yaml. Name of the group was also added to new launch file. In moveit_dimash joint names yaml file and controllers yaml file were created according to the instructions. The following launch files were also updated in order to run newly created package: moveit_planning_execution, moveit_rviz.



```
/home/dima/ros1_ws/src/moveit_dimash/launch/moveit_planning_execution.launch http://l...
h 1 states already in datastructure
[ INFO] [1730831939.211302319, 120.063000000]: RRTConnect: Created 4 states (2 s
tart + 2 goal)
[ INFO] [1730831939.226605979, 120.071000000]: RRTConnect: Created 5 states (2 s
tart + 3 goal)
[ INFO] [1730831939.230257914, 120.073000000]: RRTConnect: Created 5 states (2 s
tart + 3 goal)
[ INFO] [1730831939.230357001, 120.073000000]: RRTConnect: Created 5 states (2 s
tart + 3 goal)
[ INFO] [1730831939.230440086, 120.073000000]: ParallelPlan::solve(): Solution f
ound by one or more threads in 0.056477 seconds
[ INFO] [1730831939.231398347, 120.074000000]: RRTConnect: Starting planning wit
h 1 states already in datastructure
[ INFO] [1730831939.232743647, 120.075000000]: RRTConnect: Starting planning wit
h 1 states already in datastructure
[ INFO] [1730831939.258541447, 120.087000000]: RRTConnect: Created 5 states (2 s
tart + 3 goal)
[ INFO] [1730831939.260025566, 120.088000000]: RRTConnect: Created 6 states (2 s
tart + 4 goal)
[ INFO] [1730831939.262227760, 120.088000000]: ParallelPlan::solve(): Solution f
ound by one or more threads in 0.031046 seconds
[ INFO] [1730831939.309834853, 120.108000000]: SimpleSetup: Path simplification
took 0.047497 seconds and changed from 3 to 2 states
```

Fig. 3: Simulation in Gazebo



```
dima@dima-Legion-Y540-15IRH-PG0:~/ros1_ws$ source devel/setup.bash
dima@dima-Legion-Y540-15IRH-PG0:~/ros1_ws$ rosrun lab4 test_moveit
[ INFO] [1730832585.759025524]: Loading robot model 'hand_tutorial'...
[ INFO] [1730832585.763456562]: No root/virtual joint specified in SRDF. Assumin
g fixed joint
[ INFO] [1730832587.372837328, 493.062000000]: Ready to take commands for planni
ng group move_dimash.
[ INFO] [1730832595.577459952, 497.432000000]: Done
dima@dima-Legion-Y540-15IRH-PG0:~/ros1_ws$
```

Fig. 4: Movement along X axis

IV. Task2: Moving the end-effector along X axis

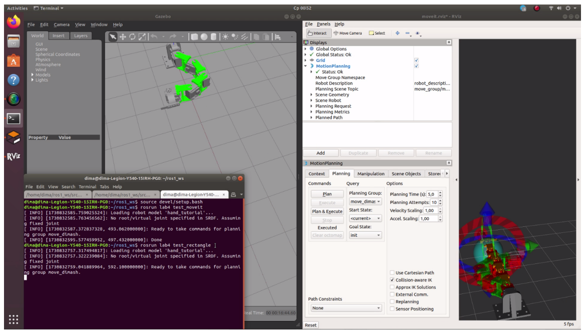
New catkin package was created to move the robot from the script. Therefore, a ROS node in C++ was created to move the end-effector by 1.4 units along the X-axis using cartesian control in RVIZ.

- **Initialization**: The program starts by including the required MoveIt headers, which enable access to functionalities for robot control and planning. These include 'planning_scene_interface' for setting up collision objects and 'move_group_interface' for managing joint configurations and executing movements.
- **Setting up ROS and MoveIt**: ROS initialization is performed with '`ros::init`' to create a node and an asynchronous spinner to handle callbacks. This setup ensures continuous execution and processing of background tasks during motion planning.
- **Planning Group Definition**: The planning group 'move_dimash' is specified for controlling the robot's manipulable joints. By creating a 'MoveGroupInterface' object with this group, we can set and update target poses directly, without manually specifying each joint.
- **Defining the Target Pose**: The current pose of the robot's end-effector is first retrieved. We set the initial target position by offsetting the X-axis by -1.4 units, which directs the end-effector backward along this axis. The code then sets this modified position as the new target.
- **Motion Execution Using a Loop**: A 'while' loop is used to move the robot incrementally towards the target pose. Inside the loop:
  - The target pose is set with an approximate joint configuration.
  - The 'move_group.move()' command is called to execute each small step in the movement.
  - A check on the end-effector's current X position ensures that the target has been reached with a tolerance of 0.01 units.

  This process guarantees precise motion control, iterating until the robot reaches the defined position.
- **Completion and Shutdown**: Once the target position is achieved, the program outputs a completion message. Finally, '`ros::shutdown()`' is called to safely terminate the node.

V. Task 3: Draw a rectangular shape with end-effector

- **Initialization**: The program starts by including the essential MoveIt headers, enabling interaction with MoveIt functionalities for motion planning and robot control:
  - 'planning_scene_interface' for managing planning scenes and collision objects.
  - 'move_group_interface' for controlling robot joints and executing motions.
  - 'moveit_msgs' for exchanging robot state and trajectory information.
- **Setting up ROS and MoveIt**: The ROS system is initialized using '`ros::init`', and a ROS node handle is created. An asynchronous spinner is then started to handle ROS callbacks during motion planning and execution:
  - '`ros::init`' initializes the ROS system with the node name 'move_group_interface_tutorial'.
  - '`ros::AsyncSpinner(0)`' ensures that callbacks can occur while the main thread processes motion.
- **Defining the Planning Group**: A planning group, 'move_dimash', is defined for controlling the robot's joints involved in manipulation. A 'MoveGroupInterface' object is created to handle planning and motion tasks for this group:
  - The 'MoveGroupInterface' allows easy manipulation of the robot's joint states and execution of planned motions.
- **Setting the Initial Target Pose**: The current pose of the robot's end-effector is retrieved, and a target pose is defined by shifting the position along the X and Y axes to trace the rectangle's shape:
  - The initial target position is set by adjusting the X and Y coordinates of the current pose.
  - The first target move reduces the X-coordinate by 1.4 units to move the end-effector along the X-axis.
- **Motion Execution for Rectangular Path**: The code moves the robot in a rectangular path by sequentially adjusting the X and Y coordinates of the target pose. A 'while' loop is used to execute these motions incrementally, checking whether each new target pose has been reached:
  - The first movement involves moving the end-effector backward along the X-axis.



```
dima@dima-Legion-Y540-15IRH-PG0:~/ros1_ws$ source devel/setup.bash
dima@dima-Legion-Y540-15IRH-PG0:~/ros1_ws$ rosrun lab4 test_moveit
[ INFO] [1730832585.759025524]: Loading robot model 'hand_tutorial'...
[ INFO] [1730832585.763456562]: No root/virtual joint specified in SRDF. Assumin
g fixed joint
[ INFO] [1730832587.372837328, 493.062000000]: Ready to take commands for planni
ng group move_dimash.
[ INFO] [1730832595.577459952, 497.432000000]: Done
dima@dima-Legion-Y540-15IRH-PG0:~/ros1_ws$ rosrun lab4 test_rectangle
[ INFO] [1730832757.317494817]: Loading robot model 'hand_tutorial'...
[ INFO] [1730832757.322239084]: No root/virtual joint specified in SRDF. Assumin
g fixed joint
[ INFO] [1730832759.041889964, 592.100000000]: Ready to take commands for planni
ng group move_dimash.
```

Fig. 5: Movement in rectangular shape

  - The second movement moves the end-effector along the Y-axis by reducing the Y-coordinate by 0.7 units.
  - The next movement moves the end-effector forward along the X-axis by 0.7 units.
  - The final movement moves the end-effector forward along the Y-axis by 0.7 units to complete the rectangle.

  Each movement is checked with a tolerance condition (‘0.01‘ units) to ensure precise motion. After each move, the code verifies the new position and checks for completion.
- **Completion and Shutdown**: Once the rectangle’s path is completed, a message is displayed, and the ROS node is safely shut down:
  - The ‘`ROS_INFO("Movement is finished")`‘ prints a message confirming that the motion is complete.
  - The program terminates with ‘`ros::shutdown()`‘, safely ending the node’s execution.

VI. Discussion and Conclusion

This lab successfully implemented Cartesian pose control of a robotic manipulator using the MoveIt library, which facilitated the configuration and execution of movements in both real and simulated environments. The main objective of the lab was to control the end-effector’s position by moving it along the X-axis, drawing a rectangular path, and exploring additional motions such as a circle. By leveraging MoveIt’s powerful capabilities for inverse kinematics and trajectory planning, the lab provided valuable insight into robotic arm control through Cartesian commands. By configuring both real and simulated robots, the lab provided hands-on experience with robotic motion planning, including the use of inverse kinematics, trajectory planning, and end-effector control.