Cartesian Space Control of the Robot End-Effector: IK Package in MoveIt

Moldir Zabirova, Valeriya Kostyukova, Daryn Kenzhebek

I. MoveIt Library Configuration

MOVEIT library is a useful tool for the control of the robot in the Cartesian space. Calculation of the Inverse Kinematics of the joint angles is computationally expensive for a 5-DOFs planar robot, and MoveIt plugin provides an efficient way to plan, control, and navigate a robot in the Cartesian space.

In this laboratory we used the library for the robot's end-effector control, particularly, making the end-effector to draw different shapes and follow specific paths. We configured our own package and uploaded a URDF file to MoveIt. This file contains information about locations of the links and joints, the types of joints, and the shape of the links.

In MoveIt, we configured the necessary features of the snake robot such as self collisions, planning groups, and a KDL plugin, and saved them in the created package. We can finally launch the package in ROS and simulate the robor movement in RViz.

For the real robot, we established new controllers for our planning group in a new .yaml file and created a new .launch file to launch these controllers. We can finally run the .launch file for the Cartesian control. RViz helps to plan and execute different paths for the robot. But we can also control the movements from a script.

II. End-Effector Movement along X-Axis

To move the robot's end along the x-axis we created a node in C++ language. We specified the group we want to deal with (we had one group for the whole snake robot) and by retrieving the information about the current pose we changed the x-coordinate.

```
current_pose = move_group.getCurrentPose();
target_pose = current_pose;
target_pose.pose.position.x =
target_pose.pose.position.x - 1.4;
```

After including the necessary lines in the CMakeLists.txt (allowing to use C++, adding moveit ros planning interface, specifying additional locations of the header files, and including executables), we run the node (Fig. 1).

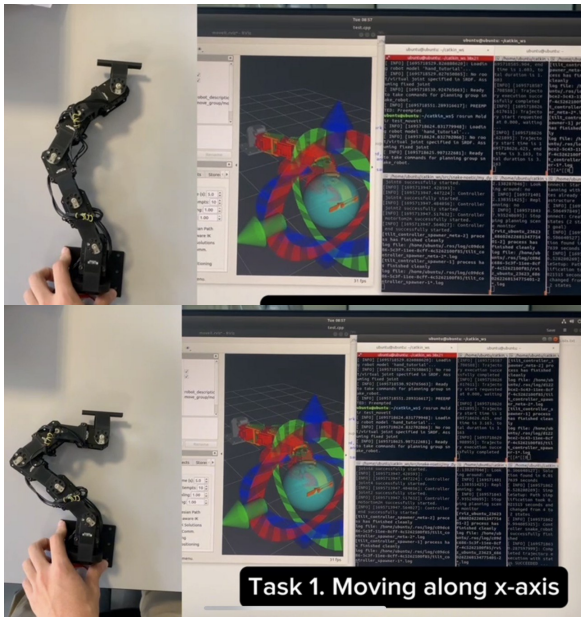

Fig. 1. Snake Robot Moves along X-Axis.

Fig. 2. Node for Drawing a Rectangular Path

III. End-Effector Drawing a Rectangle

For the end-effector drawing a rectangle we nested if-statement with x- and y-coordinates for each corner (Fig. 2).

After each adjustment, we set the approximate joint value target based on the updated position, initiated the movement, and retrieved the current pose of the robot. We checked whether the robot has reached the desired positions by comparing the absolute differences between the current and target coordinates, with each iteration moving the robot in a rectangle-like pattern. The exercise was executed in Gazebo simulator (Fig. 3).

IV. End-Effector Drawing a Circle

For a circular path, we used sine and cosine formulas for x- and y-coordinates in a while-loop (Fig. 4). Each time we added 5 degrees to the current orientation and calculated sine and cosine values describing the coordinated of the end-effector. The loop continues until the angle surpasses 370 degrees.

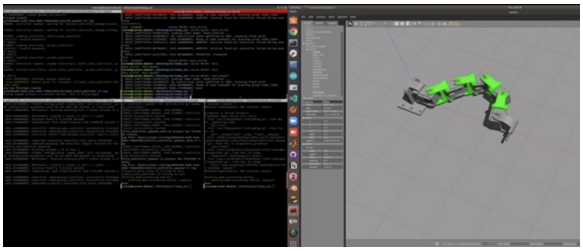

Fig. 3. Snake Robot in a Gazebo Simulator

Within each iteration, the robot's target position is updated accordingly, and the set trajectory is executed.

```
while (ros::ok()){
        move_group.setApproximateJointValueTarget(target_pose); // To calculate the trajectory
        move_group.move(); // Move the robot
        current_pose = move_group.getCurrentPose();
        double radius = 0.1;
        double angle = 5.0;
        double sine_y = 0.0;
        double cos_y = 0.0;
        double change = 0.0;
        double PI = 3.14;
        double center_x = current_pose.pose.position.x - radius;
        while (angle <= 370){
                sine_y = sin(angle*PI/180);
                                cos_y = cos(angle*PI/180);

                                if (angle >= 370){
                                        break;
                                }
                                target_pose.pose.position.y = sine_y;
                                target_pose.pose.position.x = center_x + cos_y;
                                move_group.setApproximateJointValueTarget(target_pose); // To calculate the trajectory
                                move_group.move(); // Move the robot
                                current_pose = move_group.getCurrentPose();
                                angle = angle + 5;
        }
        loop_rate.sleep();
}
```

Fig. 4. Node for Drawing a Circular Path

V. Conclusion

This laboratory project focused on the Cartesian space control of a snake robot's end-effector using the MoveIt library. By configuring the MoveIt library for the robot's end-effector control, we successfully employed it to plan, control, and navigate in the Cartesian space. We demonstrated the capabilities of Cartesian control by making the robot's end-effector draw different shapes and follow specific paths, including movement along the x-axis, drawing a rectangle, and tracing a circular path. The implementation involved coding in C++ and utilizing ROS tools like RViz for simulation. This project not only enhanced our understanding of robotic control in Cartesian space but also provided practical insights into configuring and utilizing the MoveIt library for efficient robot motion planning and execution.