NAZARBAYEV UNIVERSITY

Tutorial
“NU” text in space using Staubli

In this tutorial, we will make a Scara Staubli robot write a text “NU” in space. The approach will be using MoveIt to control the arm of the robot and give it cartesian coordinates to make a writing. It is easier, as you do not need to calculate the inverse kinematics at every step.

Step 1:

Set up the environment with ROS melodic. Download or ask for the ros package of the given robot. You will need MoveIt, so you should have it on your computer. You can download it using *“sudo apt install ros-<your-ros-distribution>-moveit”.* In the end, go to your workspace and write “catkin_make”.

Step 2:

After downloading the prerequisites, you should write a code either in python or c++ to control the arm of the robot. I did it in python code. You can write and set up your code wherever you want. I wrote it in the “my_pcl_tutorial/src” folder and named the file as “text.py”. Before running this node you should first compile it using “chmod +x text.py” if using python. Then, make changes in the “CMakefile.txt” by adding this file into the python scripts.

Step 3:

Write a code. Here, is the example code:

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import sys
import rospy
import moveit_commander
import geometry_msgs.msg
from math import radians

def main():
    # Initialize the MoveIt commander and ROS node
    moveit_commander.roscpp_initialize(sys.argv)
    rospy.init_node('write_nu', anonymous=True)

    # Create a MoveGroupCommander for the robot's planning group
    group_name = "arm"  # Replace with your actual planning group name
    move_group = moveit_commander.MoveGroupCommander(group_name)

    # Set up planning parameters
    move_group.set_max_velocity_scaling_factor(0.1)  # Reduce velocity for smoother execution
    move_group.set_max_acceleration_scaling_factor(0.1)
```

```python
# Define waypoints for "N" and "U"
waypoints = []

# Current pose
start_pose = move_group.get_current_pose().pose

# "N" waypoints
n_start = geometry_msgs.msg.Pose()
n_start.orientation = start_pose.orientation
n_start.position.x = 0.3
n_start.position.y = 0.0
n_start.position.z = 0.2
waypoints.append(n_start)

n_up = geometry_msgs.msg.Pose()
n_up.orientation = n_start.orientation
n_up.position.x = 0.3
n_up.position.y = 0.1
n_up.position.z = 0.2
waypoints.append(n_up)

n_diag = geometry_msgs.msg.Pose()
n_diag.orientation = n_up.orientation
n_diag.position.x = 0.35
n_diag.position.y = 0.0
n_diag.position.z = 0.2
waypoints.append(n_diag)

n_end = geometry_msgs.msg.Pose()
n_end.orientation = n_diag.orientation
n_end.position.x = 0.35
n_end.position.y = 0.1
n_end.position.z = 0.2
waypoints.append(n_end)

# Transition between "N" and "U"
transition = geometry_msgs.msg.Pose()
transition.orientation = n_end.orientation
transition.position.x = 0.35
transition.position.y = 0.1
transition.position.z = 0.3  # Lift to avoid collision
waypoints.append(transition)

# "U" waypoints
u_start = geometry_msgs.msg.Pose()
u_start.orientation = transition.orientation
u_start.position.x = 0.35
u_start.position.y = 0.1
u_start.position.z = 0.2
waypoints.append(u_start)
```

```
    u_down = geometry_msgs.msg.Pose()
    u_down.orientation = u_start.orientation
    u_down.position.x = 0.35
    u_down.position.y = 0.0
    u_down.position.z = 0.2
    waypoints.append(u_down)

    u_bottom = geometry_msgs.msg.Pose()
    u_bottom.orientation = u_down.orientation
    u_bottom.position.x = 0.375
    u_bottom.position.y = 0.0
    u_bottom.position.z = 0.2
    waypoints.append(u_bottom)

    u_up = geometry_msgs.msg.Pose()
    u_up.orientation = u_bottom.orientation
    u_up.position.x = 0.375
    u_up.position.y = 0.1
    u_up.position.z = 0.2
    waypoints.append(u_up)

    # Compute the Cartesian path for "N" and "U"
    (plan, fraction) = move_group.compute_cartesian_path(
        waypoints,   # Waypoints to follow
        0.01,        # eef_step
        0.0          # jump_threshold
    )

    rospy.loginfo("Planned trajectory fraction: %f" % fraction)

    # Execute the planned trajectory if sufficiently complete
    if fraction > 0.9:
        move_group.execute(plan, wait=True)
        rospy.loginfo("Successfully executed 'N' and 'U' trajectory.")
    else:
        rospy.logerr("Failed to compute a valid trajectory for 'N' and 'U'.")

    # Shutdown MoveIt
    moveit_commander.roscpp_shutdown()

if __name__ == '__main__':
    try:
        main()
    except rospy.ROSInterruptException:
        pass
```

As you can see it uses “moveit_commander” to control the robot. However after running this code there might be some issues. First, remember that the workspace has a radius of 0.4m. Thus, if you exceed this number it won’t run properly. Next, there is a problem of GOAL_TOLERANCE_VIOLATED, meaning that the robot cannot reach the given points in the specified tolerances. To solve this issue, you need to change the “ros_controllers.yaml” file in the “scara_auto/src/staubli_experimental/staubli_rs40b_moveit/config” folder. You should add goal_time and other constraints. Also, as the robot has some oscillations and overshoots you may need to adjust pid constant accordingly. Example change looks like this:

```yaml
arm_controller:
  type: effort_controllers/JointTrajectoryController
  joints:
    - joint_1
    - joint_2
    - joint_3
    - joint_4
  constraints:
    goal_time: 3.0
    stopped_velocity_tolerance: 0.1
    joints:
      joint_1: {trajectory: 0.05, goal: 0.1}
      joint_2: {trajectory: 0.05, goal: 0.1}
      joint_3: {trajectory: 0.05, goal: 0.1}
      joint_4: {trajectory: 0.05, goal: 0.1}
  gains:
    joint_1:
      p: 50
      d: 10
      i: 0.01
      i_clamp: 1
    joint_2:
      p: 50
      d: 10
      i: 0.01
      i_clamp: 1
    joint_3:
      p: 50
      d: 10
      i: 0.01
      i_clamp: 1
    joint_4:
      p: 50
      d: 10
      i: 0.01
      i_clamp: 1
gripper_position:
  type: effort_controllers/JointTrajectoryController
  joints:
    - gripper
  goal_time: 3.0
  stopped_velocity_tolerance: 0.1
  joints:
    gripper: {trajectory: 0.05, goal: 0.1}
  gains:
    gripper:
      p: 50
      d: 10
      i: 0.01
      i_clamp: 1
```

Finally, to run the code and look in the RVIZ and as well as the Gazebo you should use the MoveIt folder. So, in the first window run the “roslaunch staubli_rs40b_moveit demo_gazebo.launch”. In the other launch the code itself, ‘rosrun my_pcl_tutorial text.py“. It should show you that the robot executed 100% of the route. Next, observe the rviz and gazebo if the robot is moving the right way.

----------THE END----------