Fanuc CRX-10iA Robot Arm with MoveIt2 in ROS 2

1. Introduction

This report documents the design, implementation, and evaluation of a robotic manipulation system based on the Fanuc CRX-10iA collaborative robot arm using ROS 2 Humble, Gazebo, MoveIt2, and ros2_control. The work was carried out as part of two assigned tasks:

1. **Single-arm motion planning with a Robotiq gripper using MoveIt2**
2. **Dual-arm coordinated motion using a custom Cartesian controller**

2. Software and Tools Used

- **Operating System: Ubuntu (ROS 2 Humble)**
- **Simulation: Gazebo (Classic)**
- **Motion Planning: MoveIt2**
- **Control Framework: ros2_control, JointTrajectoryController**
- **Kinematics Library: Orocos KDL (KDL Kinematics & Dynamics)**
- **Visualization: RViz2**
- **Programming Languages: C++, Python (launch and utility scripts)**

3. Task 1 – Single Arm Motion Planning with Robotiq Gripper

3.1 Objective

The goal of Task 1 was to simulate a single Fanuc CRX-10iA robot arm equipped with a Robotiq gripper, and to enable motion planning and execution using MoveIt2. This included:

- **Accurate URDF/Xacro modeling**
- **ros2_control-based actuation in Gazebo**
- **MoveIt2 planning and execution**
- **Gripper actuation via action interfaces**

3.2 System Setup

Gazebo Simulation

The robot arm was launched in Gazebo using a custom launch file:

ros2 launch ur_description_pkg gazebo.launch.py

The simulation includes:

- **CRX-10iA arm URDF with inertial, collision, and visual properties**
- **ros2_control hardware interface using gazebo_ros2_control**
- **JointTrajectoryController for arm joints**

MoveIt2 Integration

MoveIt2 was configured using a custom fanuc_moveit_config package:

ros2 launch fanuc_moveit_config moveit.launch.py

Key aspects of the MoveIt2 setup:

- **Separate planning groups for arm and gripper**
- **Proper SRDF configuration for joint groups and end-effector**
- **Integration with ros2_control for trajectory execution**

3.3 Motion Execution

A joint-space trajectory can be sent directly using a ROS 2 action:

```
ros2 action send_goal \
  /arm_cont/follow_joint_trajectory \
  control_msgs/action/FollowJointTrajectory \
  "{ trajectory: { joint_names: ['joint_1','joint_2','joint_3','joint_4','joint_5','joint_6'], points: [{
positions: [0.0, -0.5, 0.5, 0.0, 0.3, 0.0], time_from_start: { sec: 3, nanosec: 0 } }] } }"
```

A custom C++ node was developed using the MoveIt2 C++ API:

ros2 run robot_moveit simple_moveit_interface

This node demonstrates:

- **Planning to target joint or pose goals**

- **Executing trajectories via MoveIt2**
- **Integration with RViz for visualization**

3.4 Gripper Control

The Robotiq gripper is controlled using a GripperCommand action:

```
ros2 action send_goal /robotiq_gripper_controller/gripper_cmd \
control_msgs/action/GripperCommand \
"{command: {position: 0.4, max_effort: 100.0}}"
```

Special care was required to correctly model and control the gripper due to mimic joints, which introduced several integration challenges (discussed later).

4. Task 2 – Dual Arm Motion Planning with Custom Cartesian Controller

4.1 Objective

The objective of Task 2 was to enable synchronized motion of two CRX-10iA robot arms mounted on a common base structure. The task explicitly required simultaneous motion, which exposed limitations in standard MoveIt2 multi-group planning.

4.2 Dual-Arm System Design

Mechanical and URDF Design

- **A single URDF/Xacro file was created containing:**
  - **A custom triangular prism–like mounting structure (45 DEGREES)**
  - **Two CRX-10iA robot arms mounted on opposite faces**
- **Careful placement of base frames was required to ensure:**
  - **Correct kinematic chains**
  - **Proper TF tree structure**

This step involved significant trial and error, especially in aligning the robot base links with the mounting geometry.

4.3 Initial MoveIt2-Based Strategy (and Its Limitations)

The initial approach was:

- **Define two MoveIt planning groups, one per arm**
- **Plan motions independently for each arm**
- **Attempt to execute motions simultaneously**

Observed behavior:

- **Even with separate move groups, MoveIt2 executed motions sequentially, not synchronously**
- **One arm completed its motion before the second arm started**

Conclusion:
MoveIt2, in its default configuration, does not guarantee time-synchronized execution across multiple planning groups. Achieving true synchronization would require complex trajectory merging or a higher-level coordinator.

4.4 Final Strategy – Custom Cartesian Controller

To overcome the synchronization issue, a new strategy was adopted.

Key Design Decisions

- **Bypass MoveIt2 for execution (but keep it for visualization/debugging if needed)**
- **Implement a custom Cartesian controller node**
- **Use KDL for inverse kinematics computation**
- **Command both arms using a single ROS 2 node and callback**

4.5 Controller Architecture

The Cartesian controller node performs the following steps:

1. **Subscribe to a single topic /desired_pose**

Receive a 14-element array:
Format:
[x_l, y_l, z_l, qx_l, qy_l, qz_l, qw_l,
x_r, y_r, z_r, qx_r, qy_r, qz_r, qw_r]

2. **Split the message into left and right arm pose goals**
3. **Compute inverse kinematics for each arm using KDL**
4. **Construct joint trajectories for both arms**
5. **Publish trajectories within the same callback, ensuring synchronized motion**

4.6 Running the Dual-Arm System

Launch Gazebo Simulation
ros2 launch ur_description_pkg dual_gazebo.launch.py

Run Cartesian Controller Node
ros2 run robot_moveit cartesian_controller

Send Goal Poses

Arbitrary Goal:

```
ros2 topic pub --once /desired_pose std_msgs/msg/Float64MultiArray \
"{data: [0.2, 1.149, 0.28, -0.66, -0.52, -0.02, 0.52,
      -0.46, 1.179, 0.338, -0.7, 0.42, 0.113, 0.55]}"
```

Home Configuration:

```
ros2 topic pub --once /desired_pose std_msgs/msg/Float64MultiArray \
"{data: [1.23, -0.15, 0.44, 0.91, -0.0, 0.39, -0.0,
      -1.27, -0.15, 0.44, 0.91, -0.0, -0.39, 0.0]}"
```

5. Challenges Faced and Solutions

5.1 Robotiq Gripper and Mimic Joints

- **MoveIt2 and ros2_control had difficulty handling mimic joints**
- **Gazebo ros2_control plugins had to be modified**
- **Separate planning groups were created for the arm and gripper**

Outcome:
Reliable gripper planning and execution achieved.

5.2 Dual-Arm Synchronization

Problem:

- **MoveIt2 executed dual-arm motions sequentially**

Solution:

- **Designed a custom Cartesian controller**
- **Used a single callback to command both arms**
- **Published trajectories simultaneously**

Outcome:
True synchronized motion of both robot arms.

5.3 URDF Complexity

- **Designing a shared mounting structure was non-trivial**
- **Small errors in transforms caused large kinematic issues**

Solution:

- **Incremental URDF testing**
- **Visual validation in RViz and Gazebo**

6. Results and Demonstration

Task 1 – Single Arm with Gripper

Video demonstration:

- **https://drive.google.com/file/d/1Y_UGj7euWUwUPLbSSBUvMNqmEwy80Kb3/view**

Task 2 – Dual Arm Cartesian Control

Video demonstration:

- **https://drive.google.com/file/d/1KiJ8Zgyk8HmjuDJhHmEUsnXrvC-3C7LA/view**

7. Conclusion

This project demonstrates a complete robotic manipulation pipeline in ROS 2, from URDF modeling and simulation to motion planning and custom control. While MoveIt2 is powerful for single-robot planning, this work highlights its limitations for tightly synchronized multi-robot systems. By

designing a custom Cartesian controller, reliable and deterministic dual-arm coordination was achieved.

The project strengthened understanding of:

- **ros2_control internals**
- **MoveIt2 execution behavior**
- **Kinematics-based control strategies**
- **System-level robotics integration**