

Lab 5: Introduction to Robotic Arm Kinematics

DATE: Feb 16th. 2023

Lecturer: *Queenie Qiu, John Raiti*. Student: *Shucheng Guo*

1 Learning Objectives

1. Learn the difference between arm motions in task-space and configuration-space.
2. Familiarize yourself with MoveIt! as a ROS component.
3. Implement high-level nodes through MoveIt! combined with RViz and Gazebo to move a Kinova Gen3lite arm and a Fetch robotics arm using forward and inverse kinematics.

2 Introducing Kinova Gen3lite

2.1 ROS Kortex Package Installation

ROS Kortex is the official ROS package to interact with Kortex and its related products. These are the instructions to run in a terminal to clone the `ros_kortex` repository and install the necessary ROS dependencies.

Note: The default branch for git clone is `develop` for ROS Noetic. If you are using other ROS distributions, please git clone to corresponding branch.

```
$ sudo apt install python3 python3-pip
$ sudo python3 -m pip install conan
$ conan config set general.revisions_enabled=1
$ conan profile new default --detect > /dev/null
$ conan profile update settings.compiler.libcxx=libstdc++11 default
$ cd catkin_ws/src
$ git clone https://github.com/Kinovarobotics/ros_kortex.git
$ cd ../
$ rosdep install --from-paths src --ignore-src -y
```

Then, to build and source the workspace:

```
$ catkin_make
$ source devel/setup.bash
```

2.2 Spawning a Kinova Gen3lite robot in Gazebo

We will use the `ros_kortex` repository to work with the Kinova arm. In this metapackage, we will use the `kortex_gazebo` portion for simulation. Execute the following launch file that will bring up a gen3lite in Gazebo and open an RViZ window

```
$ roslaunch kortex_gazebo spawn_kortex_robot.launch arm:=gen3_lite z0:=0.8
```

The Gazebo window looks like:

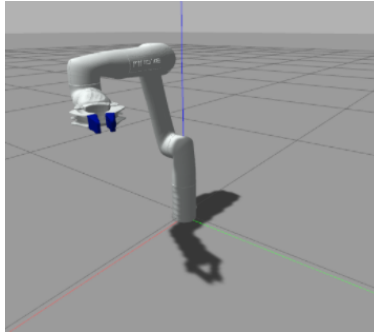


Figure 1: Spawning in Gazebo.

Note: Once the RViz and Gazebo windows have been opened, and the terminal shows two green messages (“You can start planning now!” and “The Kortex driver has been initialized correctly!”) you can add a Robot model and add a MotionPlanning component in Rviz.

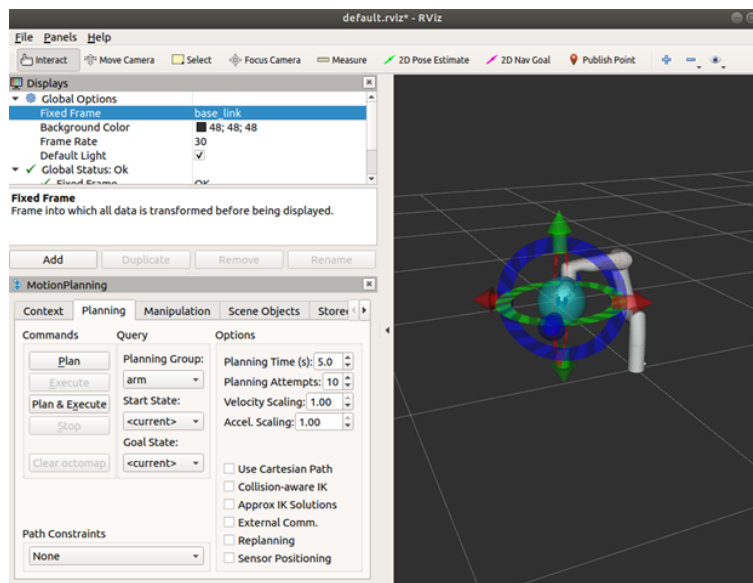


Figure 2: Kinova arm in Rviz.

2.3 Using the Interactive Markers to change the robot’s pose based on the task-space

You can use the different arrows and rings on the interactive marker, to change positions and orientations of the robot’s end-effector. As you use them, you will see an orange version of the robot arm with the proposed new position. Once you have reached a desired position, use the buttons “Plan” and “Execute”(in the MotionPlanning Panel) to make the robot in Gazebo move to the new proposed position. Some useful tools for you to track the robot’s motions are:

1. Check the values of the joints:

```
$ rostopic echo -n 1 /my_gen3_lite/joint_states
```

2. Outputs the pose of the robot's end-effector with respect to the world (in this case Gazebo's origin):

```
$ rosrn tf tf_echo /world /end_effector_link
```

3. You may also consider using the base link as the reference:

```
$ rosrn tf tf_echo /base_link <END-EFFECTOR LINK>
```

If you want to change the state of the gripper, you need to change the Planning Group (in the MotionPlanning Panel) from arm to gripper, and you can select a new position.

Deliverables:

1. Add a TF component in RViz and enable only the base_link and the end_effector_link. Disable the Motion Planning component to temporarily remove the interactive markers. Add a screenshot of the RViZ view.

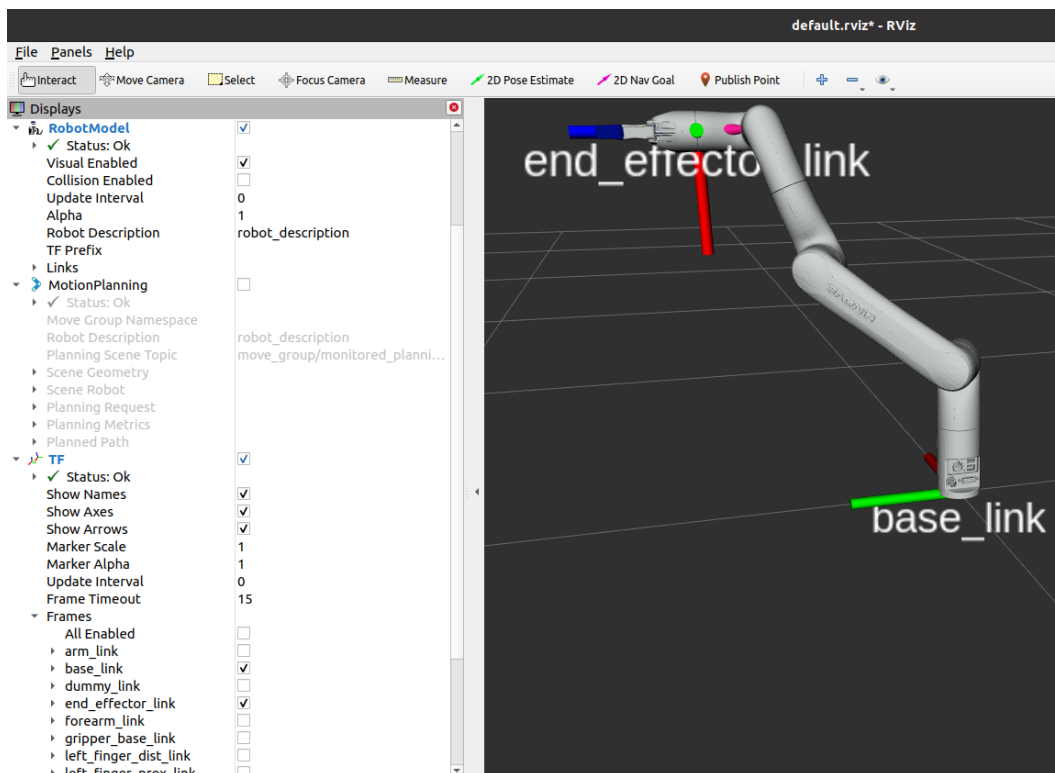


Figure 3: RViZ with Kinova.

2. Use `rqt_tf_tree` to determine the kinematic chain between the two reference frames (`base_link` and `end_effector_link`).
3. Turn on the Motion Planning component again. The "Goal State" in the MotionPlanning panel allows you to set goals to some predefined positions. Inspect it and report what are the reported robot's joints values and pose (wrt the world) when you send the `gen3lite` to the following preset positions:
 - a. Home
 - b. Retract
 - c. Select a robot pose that you would choose if you had to pick something from the table. Add a screenshot of the resulting pose of the arm in **Gazebo**.

2.4 Using the Joints tab in MotionPlanning to change the robot's pose on the configuration-space

In the MotionPlanning panel, find the "Joints" tab. You will be able to change the values of the different joint angles. Explore the configuration space, determine which joints move in which direction when a positive or a negative angle is given to each joint.

Deliverables:

1. Configure the joint angles and move to your desired position. Add a screenshot of the `rviz` window(including joint values and the robot visualization), use the same tools described in the previous subsection to explore the values for the pose and the joint angles.
2. Explain the differences you observed when controlling the robot's position using either the interactive markers or the joint angles.
3. Can you change all joint angles from the `RViZ` window? If not, explain what degree of freedom is not available and how that impacts the poses you can reach with the robot.

3 Introducing Fetch Robot

3.1 Fetch Packages Installation

Following the instructions to install packages from from Fetch Robotics.

```
$ cd catkin_ws/src
$ git clone -b ros1 https://github.com/fetchrobotics/fetch_ros.git
$ git clone -b gazebo11 https://github.com/fetchrobotics/fetch_gazebo.git
$ git clone -b ros1 https://github.com/fetchrobotics/fetch_msgs.git
$ git clone -b ros1 https://github.com/fetchrobotics/power_msgs.git
$ git clone -b ros1 https://github.com/fetchrobotics/robot_controllers.git
$ cd ../
$ rosdep install --from-paths src --ignore-src -y
(Ignore the error: Unable to locate package ros-noetic-simple-grasping)
$ sudo apt install ros-noetic-rgbd-launch
```

Note:The branch "ros1" and "gazebo11" is developed for ROS Noetic. If you are using other ROS distributions, please git clone to corresponding branch.

3.2 Fetch Robot Manipulation

We will go through a similar procedure as the Kinova Gen3lite, familiarizing ourselves with the arm components of the Fetch robot. Execute the following commands in different terminals to spawn a Gazebo simulated Fetch robot, activate the motion capabilities and open an RViZ window

```
$ roslaunch fetch_gazebo simulation.launch
$ roslaunch fetch_moveit_config move_group.launch
$ rosrun rviz rviz
```

You will need to add the Robot Model and the MotionPlanning components in RViZ once it opens. Check the `rqt_tf_tree` to determine the name of the end-effector's link to run the command:

```
$ rosrun tf tf_echo /odom <END-EFFECTOR LINK>
```

You may also consider using the start of the arm chain as the reference:

```
$ rosrun tf tf_echo /<FIRST ARM JOINT> <END-EFFECTOR LINK>
```

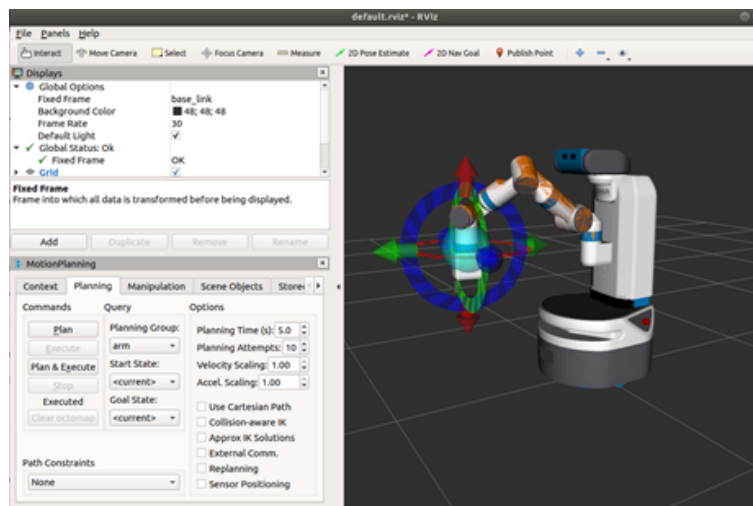


Figure 4: Fetch in Rviz.

Deliverables:

1. Add a TF component in RViz and enable only the `base_link`, the first joint of the arm, and the `end_effector_link`. Disable the Motion Planning component to remove temporarily the interactive markers. Add a screenshot of the RViZ view.
2. Use `rqt_tf_tree` to determine the kinematic chain between the two reference frames.
3. Unlike the Kinova arm, the MotionPlanning for Fetch does not have preset poses for the arm.

- (a) Show a screenshot of the resulting pose when all joints in the arm are set to 0 degrees.
 - (b) Report the joint angle values of all Fetch's arm joints on a pose that would allow Fetch to pick up an object from the floor.
4. Explain the differences you observed when controlling the robot's position using either the interactive markers or the joint angles.
 5. Can you change all joint angles from the RViZ window? If not, explain what degree of freedom is not available and how that impacts the poses you can reach with the robot.

4 Using Python to control Fetch robots

Now that we have familiarized the robot, we will create nodes that will generate motions on Fetch robot. One will move the robot arm by assigning values to joints angle, applying **forward kinematics** to reach to the target position and orientation of the end-effector in Cartesian space. Another node will apply **inverse kinematics** which calculate and change the joint angles based on the given target pose of the end-effector. You will use working code provided on canvas as a guide. You may need to modify the code to finish the deliverable.

Notes: You can visit the Fetch documentation for additional tutorials.

Deliverables:

1. Please identify which Python file corresponds to each type of robot kinematics?
2. Capture videos of the robot executing movements using both programs, separately. Upload them to Canvas with your writeup.
3. Modify the code that performs forward kinematics and move the arm to a pose with your desired input values. Compare the resulting angle values (rostopic echo /joint_states) to your inputs.
4. Modify the code that performs inverse kinematics and move the arm to a pose with your desired input values. Then compare the resulting pose (roslaunch tf tf_echo /<link1> /<link2>) to your inputs.
5. Discuss what type of robot kinematics was faster to use? Which one was easier to use?
6. What were the advantages and disadvantages of using RViz or generated code to control the robot's motion? Please provide at least two thoughts.