RBE 500 — Foundations of Robotics

Instructor: Siavash Farzan

Spring 2022

ROS Assignment 2: Forward Kinematics of Robot Manipulators

2.1 Objectives

By the end of this lab you should be able to:

- Solve the forward kinematics for a robotic manipulator
- Compare your own forward kinematics implementation to the functionality provided by ROS
- Make the Interbotix RX150 robot arm move to simple joint position goals in RVIZ
- View the state data published by RX150 using RVIZ

2.2 Problem Statement

In this assignment, we will start working on robot manipulators*. Specifically, we will write a script to solve the Forward Kinematics problem for a robot manipulator. Given joint angles and the dimension of the manipulator, the task is to compute the position and orientation of the end effector. You can use Denavit-Hartenberg (D-H) approach to derive Forward Kinematics. Several test cases will be provided for you to evaluate the performance of your program. The final script you submit should be able to pass all test cases.

2.3 Introduction to Interbotix Robot Arm

The ReactorX 150 Robot Arm belongs to a new family of arms from Interbotix. It is equipped with 4 links, 5 joints and a gripper. The arm has a 45cm horizontal reach from center of the base to gripper with a total span of 90cm. The working payload for it is 100g. The schematic and dimensions of the manipulator are shown in Figure 1. More information could be found at ReactorX 150 Manipulator.

*Parts of this assignment are adapted from the course materials by Professor K. Karydis at UC Riverside.

2.4 Set Up the Robot Manipulator

- First let's download the ROS packages for the robot arm.

```
cd ~/rbe500_ros/src
git clone https://github.com/Interbotix/interbotix_ros_arms.git
```

- We can install the dependencies by the following commands.

```
cd ~/rbe500_ros
rosdep update
rosdep install --from-paths src --ignore-src -r -y
```

- We need to install additional ROS packages before building the Interbotix ROS package:

```
sudo apt-get update
sudo apt-get upgrade
sudo apt update
sudo apt install ros-noetic-joint-state-publisher-gui
sudo apt install ros-noetic-controller-manager
sudo apt install ros-noetic-effort-controllers
sudo apt install ros-noetic-gazebo-ros-control
sudo apt install ros-noetic-joint-state-controller
sudo apt install ros-noetic-joint-trajectory-controller
sudo apt-get install ros-noetic-moveit
sudo apt-get install ros-noetic-moveit-visual-tools
sudo apt-get install ros-noetic-dynamixel-workbench-toolbox
```

- Lastly, with all dependencies ready, we can build the ROS package by the following commands:

```
cd ~/rbe500_ros
catkin_make
```

- Get familiar with the robot model and identify the location and direction of axes of rotation by launching it in RVIZ and playing with the joint state publisher.
  When assigning the coordinate frames for the DH method in Section 2.6, the frames must be assigned based on the axis and direction of revolution of each joint.

```
roslaunch interbotix_descriptions description.launch robot_name:=rx150
    jnt_pub_gui:=true
```

2.5 Programming Tips

i) We provide two scripts `forward_kinematics.py` and `test_forward_kinematics.py` for you, but only the first one is what you need to complete and submit. The second one is for your testing.

ii) Please pay attention to the data type used in your computation.

- If two matrices A and B are of the type `np.matrix`, then `np.matmul(A, B)` will perform matrix multiplication.

- If two matrices A and B are of the type `np.array`, then `A * B` will not perform matrix multiplication, but element-wise multiplication.
- If two matrices A and B are of the type `np.array`, then `np.dot(A, B)` will perform matrix multiplication.

iii) The roll, pitch, and yaw angles associated to an orientation can be determined directly from a rotation matrix, as discussed in the lectures and shown below:

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}$$

$$yaw = \tan^{-1}(r_{21}, r_{11}), \qquad pitch = \tan^{-1}(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}), \qquad roll = \tan^{-1}(r_{32}, r_{33})$$

2.6 Forward Kinematics of the Robot Manipulator

Considering the robot schematic and dimensions provided in Figure 1, derive the forward kinematics of the robot using the Denavit-Hartenberg (D-H) method.

To simplify computation, we will regard `joint4` as the end effector. In other words, you need to return the position and orientation of `joint4` instead of the actual end-effector.

Note that Frame 0 and Frame 3 are already assigned to the robot, as shown in Figure 1. To assign other frames, make sure to investigate the direction and axis of rotation of the joints by using the joint state publisher provided by RVIZ.

Requirement for Forward Kinematics:

- Formulate Denavit-Hartenberg parameters between each frame and multiply all transformation matrices.
- To find the direction and axis of rotation for each joint, make sure to use the joint state publisher provided by RVIZ.
- You can either compute everything in your program, or pre-compute some matrices by hand and hard-code them.

2.7 Sample Code

Open a new terminal and go to the `rbe500_ros` directory. We will start from a new ROS package.

```
cd ~/rbe500_ros/src
catkin_create_pkg forward_kinematics rospy
```

Create a new directory under the ROS package for the Python scripts. We will place all the python scripts (i.e. `.py` files) inside this directory.

```
roscd forward_kinematics
mkdir scripts
cd scripts
```

Go to the `scripts` folder and create a new python script.

```
roscd forward_kinematics/scripts
touch forward_kinematics.py
chmod +x forward_kinematics.py
subl forward_kinematics.py
```

Copy and paste the following code into `forward_kinematics.py`, and complete the `forward_kinematics` function in this file. Feel free to define new functions and variables in your program if needed.

```
import numpy as np
from math import pi, cos, sin, atan2, sqrt

def forward_kinematics(joints):
    # input: joint angles [joint1, joint2, joint3]
    # output: the position and orientation of joint 4 (as the end-effector):
    # [x, y, z, roll, pitch, yaw]

    joint1 = joints[0]
    joint2 = joints[1]
    joint3 = joints[2]

    # add your code here to complete the computation

    # convert the final orientation of the end-effector to roll-pitch-yaw

    return [x, y, z, roll, pitch, yaw]
```

We provide another script for testing.

```
roscd forward_kinematics/scripts
touch test_forward_kinematics.py
chmod +x test_forward_kinematics.py
subl test_forward_kinematics.py
```

Please copy and paste the following code into `test_forward_kinematics.py`. You can change the `test_case` variable to other values for testing.

```
#!/usr/bin/env python3
import rospy
import numpy as np
from math import pi
from std_msgs.msg import Header
from sensor_msgs.msg import JointState
from forward_kinematics import forward_kinematics

class Manipulator():
    def __init__(self):
        rospy.init_node('manipulator')
        rospy.loginfo("Press Ctrl + C to terminate")
        self.rate = rospy.Rate(1000)
        self.joint_pub = rospy.Publisher('/rx150/joint_states', JointState,
            queue_size=10)
```

```
        # prepare joint message to publish
        joint_msg = JointState()
        joint_msg.header = Header()
        joint_msg.name = ['waist', 'shoulder', 'elbow', 'wrist_angle',
                    'wrist_rotate', 'gripper', 'left_finger', 'right_finger']
        joint_msg.position = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.026, -0.026]

        # test case for forward kinematics
        case1 = [pi/6, -pi/3, -pi/6] # joint angle in radian
        case2 = [0, 0, 0]
        case3 = [-pi/3, pi/4, pi/2]
        case4 = [3*pi/5, -pi/3, -pi/4]

        # adjust the test case here
        test_case = case1

        result = forward_kinematics(test_case)

        print("joint angles = ", np.around(test_case, 3))
        print("x y z = ", np.around(result[0:3], 3))
        print("r p y (deg) = ", np.around(np.rad2deg(result[3:6]), 3))

        while not rospy.is_shutdown():
            joint_msg.header.stamp = rospy.Time.now()
            joint_msg.position[0:3] = test_case
            self.joint_pub.publish(joint_msg)
            self.rate.sleep()

if __name__ == '__main__':
    whatever = Manipulator()
```

2.8 Performance Testing

i) Open a terminal, launch RVIZ simulator and spawn a new robot:

```
roslaunch interbotix_descriptions description.launch robot_name:=rx150
   jnt_pub_gui:=true
```

ii) Once the robot is successfully spawned in RVIZ, we can test the scripts by running the `test_forward_kinematics` script in another terminal.

```
roscd forward_kinematics/scripts
python3 test_forward_kinematics.py
```

iii) To check the joint 4 position and orientation, run the following command in another terminal.

```
rosrun tf tf_echo /rx150/base_link /rx150/wrist_link
```

iv) Make sure to check the performance of your script for all the provided test cases in the `test_forward_kinematics.py` code above. Compare your solution with the position and orientation provided by the `tf_echo` command, and discuss them in your report.

2.9 Submission

- Submission: individual submission via Gradescope.
- Due time: as specified on Canvas
- Files to submit: (please use exactly the same filename)
  - `forward_kinematics.py`
  - `ros2_report.pdf`
- Grading rubric:
  - 60%: implement forward kinematics and pass test cases.
  - 40%: clearly describe your approach and your code, and discuss the test cases in the report.

2.10 Autograder

All code submissions will be graded automatically by an autograder uploaded to Gradescope. Your scripts will be tested on a Ubuntu cloud server using a similar ROS environment. The grading results will be available in a couple of minutes after submission.

Testing parameters are as follows.

- The tolerance for distance error is set to 0.002m (Manhattan distance on x, y, z axes).
  - The autograder will take the maximum of the error in x, y, z axes respectively, and check if the maximum error is less than 0.002m.
  - For example, if the computed position is [0.020, 0.013, 0.298], and the ground truth is [0.021, 0.012, 0.297], it should pass the test.
- For the forward kinematics test, three test cases are visible to you; two test cases are hidden. The hidden ones are designed to test some "corner cases" and to make sure there is no hard-coded computation in the submitted script. The test cases used by the autograder are different from the test cases provided in the `test_forward_kinematics.py` script.
- The autograder works in the way that it sends joint angles to the manipulator and reads the results after the manipulator actually moves according to these angles. It will compare the ground truth position with the position after moving according to the computed angles.

2.11 Manipulator Specification

The dimension of the ReactorX 150 manipulator is the following. We will take `joint4` as the end-effector point (instead of the actual gripper).

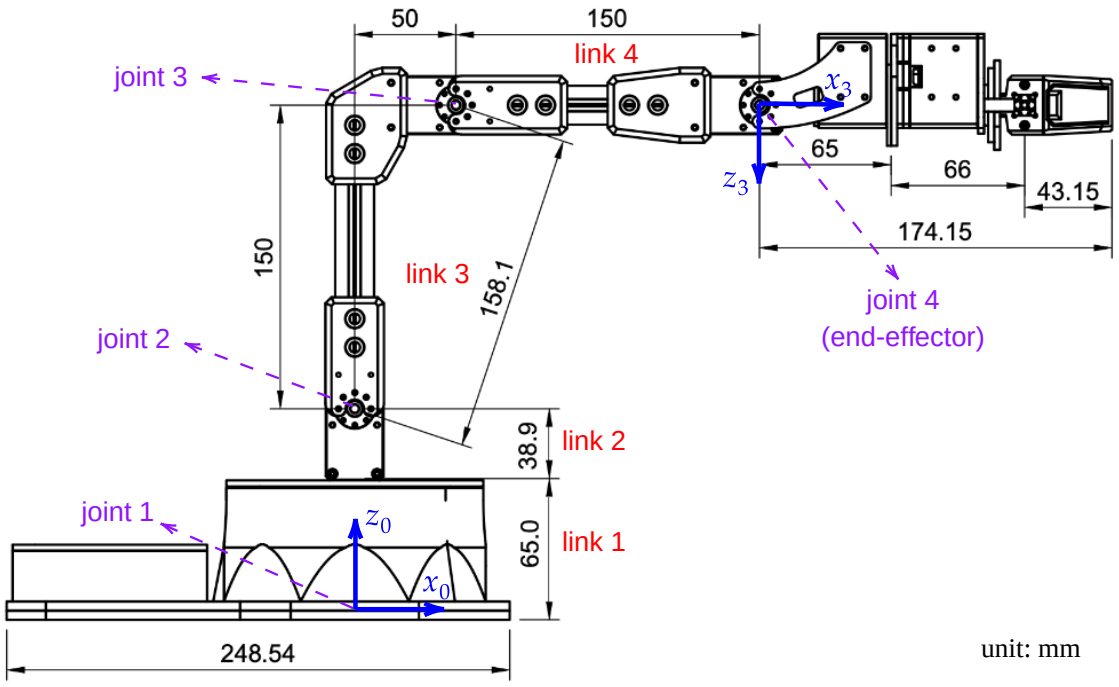

Figure 1: Schematic and dimensions of the ReactorX 150 robot manipulator.