RBE 500 — Foundations of Robotics

Instructor: Siavash Farzan

Spring 2022

ROS Assignment 3: Inverse Kinematics of Robot Manipulators

3.1 Objectives

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

- Solve the inverse kinematics for a robotic manipulator
- Compare your own inverse 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

3.2 Problem Statement

In this assignment, we will continue working on the manipulator and solve the Inverse Kinematics problem*. Specifically, we will write a script to solve the Inverse Kinematics problem for a robot manipulator. Given the position of the end-effector, the task is to solve for a set of feasible joint angles. It is recommended to use a geometric approach. 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.

3.3 Introduction to Interbotix Robot Arm

For this assignment, we will use the same ReactorX 150 Robot Arm that we used for ROS Assignment 2. The schematic and dimensions of the manipulator are shown in Figure 1.

3.4 Set Up the Robot Manipulator

To setup the ROS packages for the robot manipulator, please follow the steps described in Section 2.4 of ROS Assignment 2 (if you haven't done so already).

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

3.5 Programming Tips

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

ii) Two math tools that might be helpful for the geometric approach:

- Two-argument arctangent $\theta = arctan(y, x) \in (-\pi, \pi]$ (`atan2` can be used in python)
- Law of cosines $\gamma = arccos\big(\frac{a^2 + b^2 - c^2}{2ab}\big) \in (0, \pi)$ (`acos` can be used in python)

3.6 Inverse Kinematics of the Robot Manipulator

Considering the robot schematic and dimensions provided in Figure 1, derive the inverse kinematics of the robot using a geometric or analytical method.

To simplify computation, we will regard `joint4` as the end-effector. In other words, you are given the position of `joint4` instead of the actual end-effector.

Two more annotated figures are provided in Figure 2 and Figure 3 to help you understand the trigonometry. The θ_1, θ_2 and θ_3 marked in the figures are the joint angles you need to compute.

3.7 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 inverse_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 inverse_kinematics
mkdir scripts
cd scripts
```

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

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

Copy and paste the following code into `inverse_kinematics.py`, and complete the `inverse_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, acos, sqrt

def inverse_kinematics(position):
    # input: the position of end effector [x, y, z]
    # output: joint angles [joint1, joint2, joint3]

    x = position[0]
    y = position[1]
    z = position[2]

    # add your code here to complete the computation and calculate joint 1,
       joint 2 and joint 3 values

    return [joint1, joint2, joint3]
```

We provide another script for testing.

```
roscd inverse_kinematics/scripts
touch test_inverse_kinematics.py
chmod +x test_inverse_kinematics.py
subl test_inverse_kinematics.py
```

Please copy and paste the following code into `test_inverse_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 inverse_kinematics import inverse_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 inverse kinematics (position [x, y, z] in meter)
        case1 = [0.200, 0.000, 0.254]
        case2 = [0.17320, 0.10000, 0.25391]
        case3 = [0.02165, 0.01250, 0.29721]
        case4 = [-0.012, 0.038, 0.261]

        # adjust the test case here
        test_case = case1

        joint_angle = inverse_kinematics(test_case)

        np.set_printoptions(suppress=True)
        print("joint angles (deg) = ", np.around(np.rad2deg(joint_angle), 3))
        print("target position (m) = ", np.around(test_case, 3))

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

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

3.8 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_inverse_kinematics` script in another terminal.

```
roscd inverse_kinematics/scripts
python3 test_inverse_kinematics.py
```

iii) You need to check if the joint angles computed by your script can lead the manipulator move to the required position. To check the end-effector position, 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_inverse_kinematics.py` code above. Compare your solution with the position and orientation provided by the `tf_echo` command, and discuss them in your report.

3.9 Submission

- Submission: individual submission via Gradescope.
- Due time: as specified on Canvas
- Files to submit: (please use exactly the same filename)
  - `inverse_kinematics.py`
  - `ros3_report.pdf`
- Grading rubric:
  - 60%: implement inverse kinematics and pass test cases.
  - 40%: write a brief report and clearly describe your geometric approach formulation, explain how your program works, and discuss the performance of your code and the results for the four test cases provided in the `test_inverse_kinematics.py` script.

3.10 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.
- 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.

3.11 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).

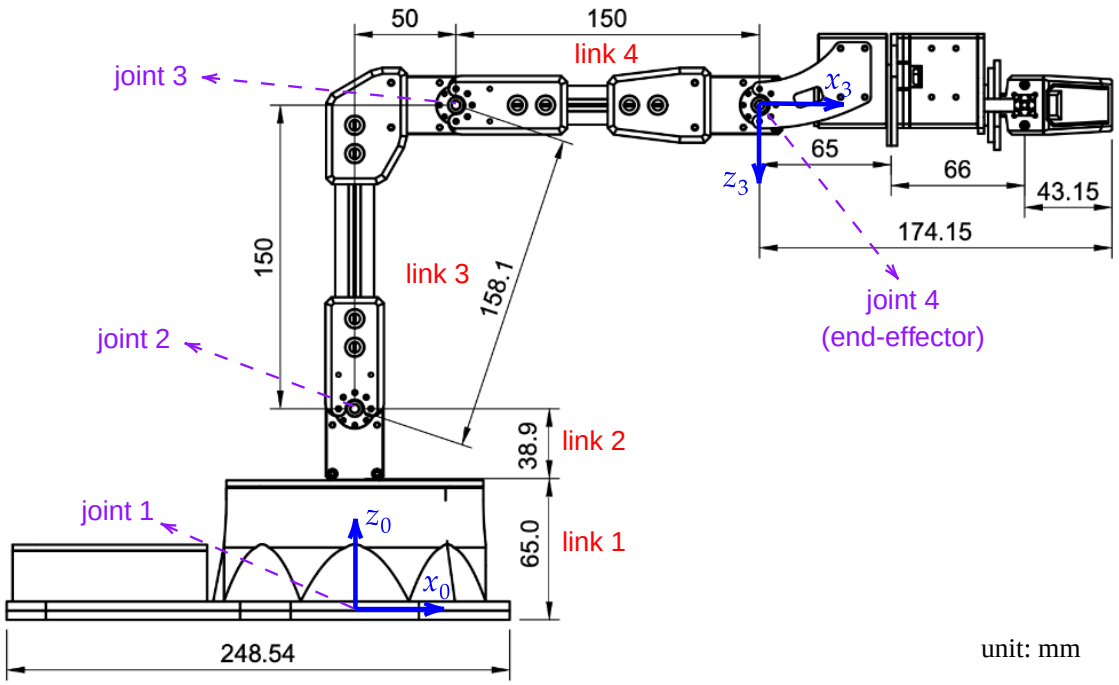

Figure 1: Schematic and dimensions of the ReactorX 150 robot manipulator.

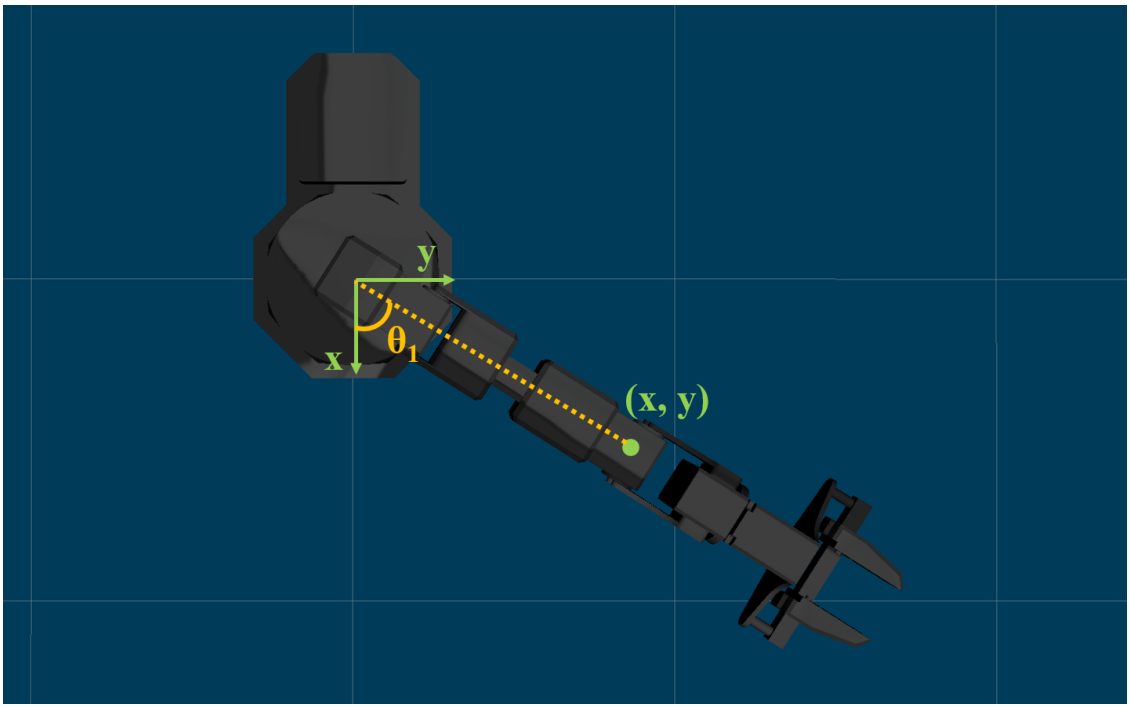

Figure 2: The joint angle θ_1, as seen from the top view.

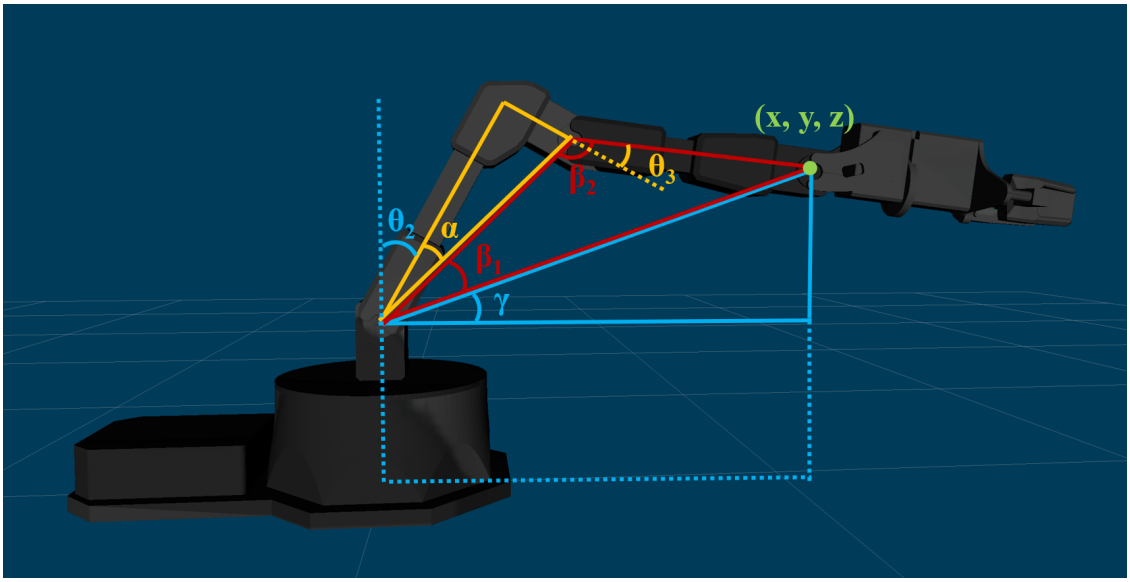

Figure 3: The joint angles θ_2 and θ_3, as seen from the side view.