Week 8 - UR3e Inverse Kinematics on Gazebo

Code Writeup

Ari Ferneau

Adam Malyshev

Shaunak Roy


```python
import rclpy
from rclpy.node import Node

from ur3e_mrc.msg import CommandUR3e

import sys
import numpy as np
class InverseKinematicsUR3e(Node):

    def __init__(self, args):
        super().__init__("fk_ur3e_pub")

        # create a publisher to publish to /ur3/comamnd
        self.publisher_ = self.create_publisher(CommandUR3e,
        ↪  '/ur3/command', 10)
        # remove any ros args from system arguments
        args = rclpy.utilities.remove_ros_args(args)

        # get all command line arguments and convert from strings
        ↪  to floats
        xWgrip, yWgrip, zWgrip, yawWgrip = tuple([float(x) for x
↪   in args[1:5]])
        #timer_period = 0.5  # seconds
        ↪
        #self.timer = self.create_timer(timer_period,
        ↪  self.move_robot)

        # move the robot given command line args
        self.move_robot(xWgrip, yWgrip, zWgrip, yawWgrip)

    def to_rad(self, value):
        return value * np.pi/180

    def move_robot(self, xWgrip, yWgrip, zWgrip, yawWgrip):
        # find ik solution
        q = self.inverse_kinematics(xWgrip, yWgrip, zWgrip,
↪   yawWgrip)
```

```
        # publish joint angles to command topic
        msg = CommandUR3e(destination = list(q), v= 1.0,a =
↪   1.0,io_0 = False)
        self.publisher_.publish(msg)

        # print joint angles in degrees
        
        ↪   self.get_logger().info(f'q:{np.rad2deg(q).astype(int).tolist()}')
        # find fk solution
        T = self.calculate_fk_from_dh(q)
        self.get_logger().info(f'\n{np.array_str(T, precision=3,
        ↪   suppress_small=True)}')


    def calculate_fk_from_dh(self,q):
        # find the complete transformation matrix from frame 0 to
        ↪   end
        # effector

        L1 = 0.152
        L2 = 0.120
        L3 = 0.244
        L4 = 0.093
        L5 = 0.213
        L6 = 0.104
        L7 = 0.083
        L8 = 0.092
        L9 = 0.0535
        L10 = 0.059

        A1 = self.get_a_matrix(0,    L1,     q[0],
↪   -np.pi/2)
        A2 = self.get_a_matrix(0,    L2,     q[1]-np.pi/2,
↪   -np.pi/2)
        A3 = self.get_a_matrix(0,    L3,     0,
↪   -np.pi/2)
        A4 = self.get_a_matrix(0,    L4,     q[2],
↪   np.pi/2)
        A5 = self.get_a_matrix(0,    L5,     0,
↪   np.pi/2)
        A6 = self.get_a_matrix(0,    L6,     q[3],
↪   -np.pi/2)
        A7 = self.get_a_matrix(0,    L7,     q[4],
↪   np.pi/2)
```

```
        A8 = self.get_a_matrix(0,    L8,     q[5]+np.pi/2,
↪   np.pi/2)
        A9 = self.get_a_matrix(0,    L9,     0,
↪   np.pi/2)
        A10 = self.get_a_matrix(0,   -L10,    np.pi/2,       0)

        #compute T by multiplying all A matrices
        T = A1@A2@A3@A4@A5@A6@A7@A8@A9@A10

        return T
    def get_a_matrix(self, r, d, theta, alpha):
        # given dh params find the transformation matrix btwn 2
        ↪  links
        return  np.array([
            [np.cos(theta), -np.sin(theta)*np.cos(alpha),
↪   np.sin(theta)*np.sin(alpha),    r*np.cos(theta)],
            [np.sin(theta), np.cos(theta)*np.cos(alpha),
↪   -np.cos(theta)*np.sin(alpha),   r*np.sin(theta)],
            [0          ,   np.sin(alpha)                ,
↪   np.cos(alpha),                  d],
            [0          ,   0                            ,   0
↪   ,   1]
        ])

    def rot_z(self, theta):
        # given an angle in radians find z rotation matrix
        return np.array([
            [np.cos(theta), -np.sin(theta), 0],
            [np.sin(theta),  np.cos(theta), 0],
            [0,              0,             1]
        ])

    def inverse_kinematics(self, xWgrip, yWgrip, zWgrip,
    ↪  yawWgrip):

        # TODO: Function that calculates an elbow up
        # inverse kinematics solution for the UR3

        #all lengths in meters
        #all angles in radians

        L1 = 0.152
        L2 = 0.120
        L3 = 0.244
        L4 = 0.093
        L5 = 0.213
```

```
L6 = 0.104
L7 = 0.083
L8 = 0.092
L9 = 0.0535
L10 = 0.059

# Step 1: find gripper position relative to the base of
↪  UR3,
# and set theta_5 equal to -pi/2

# translate gripper coords from world frame to base frame
# they are the reverse of the position of the base frame
# to the world frame
xgrip = xWgrip + 0.15
ygrip = yWgrip - 0.15
zgrip = zWgrip - 0.01

# convert given yaw into radians
yawgrip = yawWgrip*np.pi/180

# set theta_5 to -pi/2
theta_5 = -np.pi/2


# Step 2: find x_cen, y_cen, z_cen


# z_cen remains the same
z_cen = zgrip

# x_cen is the gripper x minus the length of gripper
↪  plate
# along the x-axis
x_cen = xgrip - L9*np.cos(yawgrip)

# y_cen is the gripper y minus the len of gripper plate
# along the y-axis
y_cen = ygrip - L9*np.sin(yawgrip)


# Step 3: find theta_1

# beta is the angle from the x axis to the vector (x_cen,
↪  y_cen)
beta = np.arctan2(y_cen, x_cen)
```

```
        # dy represents y_cen when theta_1 = 0
        dy = L2 - L4 + L6

        # distance to the cen point from base frame in x-y plane
        r = np.sqrt(x_cen**2 + y_cen**2)

        #alpha is the angle from the x axis to r vector when
        ↪  theta_1 is 0
        alpha = np.arcsin(dy/r)

        # theta_1 is the difference of these angles
        theta_1 = beta - alpha


        # Step 4: find theta_6

        # yaw + theta_6 = theta_1 + pi/2

        theta_6 = theta_1 + np.pi/2 - yawgrip


        # Step 5: find x3_end, y3_end, z3_end

        # rotation matrix from base frame to frame 1
        R01 = self.rot_z(theta_1)

        # find the cen coords in frame 1
        cen1 = R01.T @ np.array([x_cen, y_cen, z_cen]).T

        # 3end coords easier to calculate in frame 1 relative to
        ↪  robot
        _3_end1 = np.array([cen1[0] - L7, cen1[1] - L6 - 0.027,
↪   cen1[2] + L8 + 0.052])

        # translate 3_end into frame 0
        _3_end0 = R01 @ _3_end1.T

        # set values to that of 3_end frame 0 values
        x3_end = _3_end0[0]
        y3_end = _3_end0[1]
        z3_end = _3_end0[2]


        # Step 6: find theta_2, theta_3, theta_4

        # with absolute values geometrically:
```

```
        # theta_3 - theta_2 = theta_4

        # theta_2, theta_3 can be found same way as two link robot in text
        # where "s" is z3_end - L1 and "r" is sqrt(x3_end**2 + y3_end**2)


        r = np.sqrt(x3_end**2+y3_end**2)

        s = z3_end - L1

        D = (r**2 + s**2 - L3**2 - L5**2)/(2*L3*L5)
        theta_3 = np.arctan2(-np.sqrt(1-D**2), D)
        theta_2 = np.arctan2(s, r) - np.arctan2(L5*np.sin(theta_3), L3+L5*np.cos(theta_3))

        # reverse signs to match what the robot uses
        theta_3 = -theta_3
        theta_2 = -theta_2

        # implement the formula from before, then match sign
        theta_4 = -(np.abs(theta_3) - np.abs(theta_2))

        # Return the set of joint angles to move the robot
        return theta_1, theta_2, theta_3, theta_4, theta_5, theta_6

def main(args=None):
    # intialize node
    rclpy.init(args=args)
    ik_ur3e_pub = InverseKinematicsUR3e(args)

    # start node
    rclpy.spin_once(ik_ur3e_pub)

    # destroy node
    ik_ur3e_pub.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    # ensure 4 arguments were given
    if len(sys.argv) < 5:
        print("usage: ur3e_fk xWgrip yWgrip zWgrip yawWgrip")
    else:
        main(sys.argv)
```

Test Cases

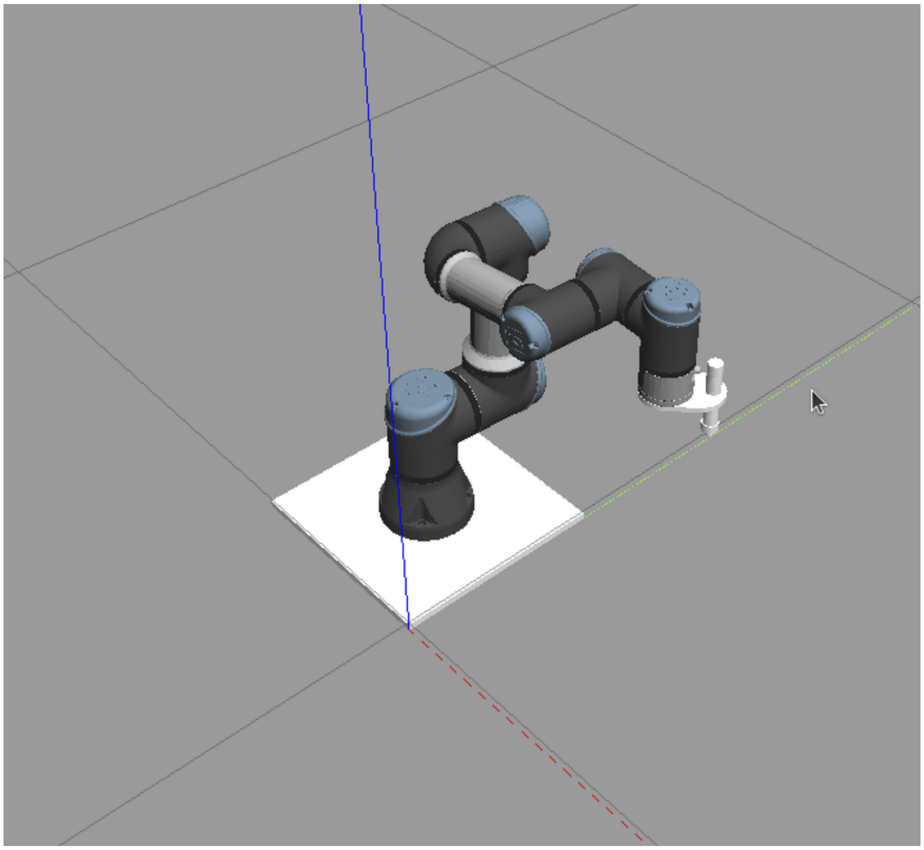

Figure 1: Test 1 image

Test Point Inputs (x, y, z, yaw)	IK solution ($\theta_1, ..., \theta_2$)	Output from `/ur3/position`
(0.2, 0.3, 0.3, 45)	-3, -87, 76, 10, -90, 41	0.2024327952511368, 0.2998730416938792, 0.2922309541454413, -3.1410653726283084, -0.0005972651532348697, 2.355397861113095
(0.1, 0.4, 0.1, 90)	13, -75, 122, -46, -90, 13	0.10221953963658278, 0.40058781416084865, 0.09208110189432546, -3.1413923344001526, -0.000778260193922649, 3.1407961970043345

Test Point Inputs (x, y, z, yaw)	IK solution ($\theta_1, ..., \theta_2$)	Output from `/ur3/position`
(0.2, 0.2, 0.2, 0)	-16, -93, 110, -16, -90, 73	0.20236654463515366, 0.19931977423280323, 0.1921250278612357, -3.1408315233541204, -0.00022523289422661107, 1.5699999427884836
(0.2, -0.2, 0.1, 0)	-66, -46, 73, -26, -90, 23	0.20088998162902214, -0.20194812505570606, 0.09223543084367461, -3.1412674345271707, -0.0007235896475745881, 1.5699998332989389
(0.2, 0.3, 0.4, 30)	-1, -75, 31, 43, -90, 58	0.2022798976349426, 0.29995554081154185, 0.39233735635051703, -3.14087941453308, -0.0004332842927020843, 2.0935987773817746

Some error sources might include the position topic is off by a little because it is not publishing the position of the end effector we are using. Another possible source of error is rounding errors and floating point conversions.

Singularities

A singularity occur when the robot would become aligned or desired (x = L3 + L5 + L7 - 0.15, y = 0, z = L1 + 0.01). It can be fixed by randomly kicking one of the angles.

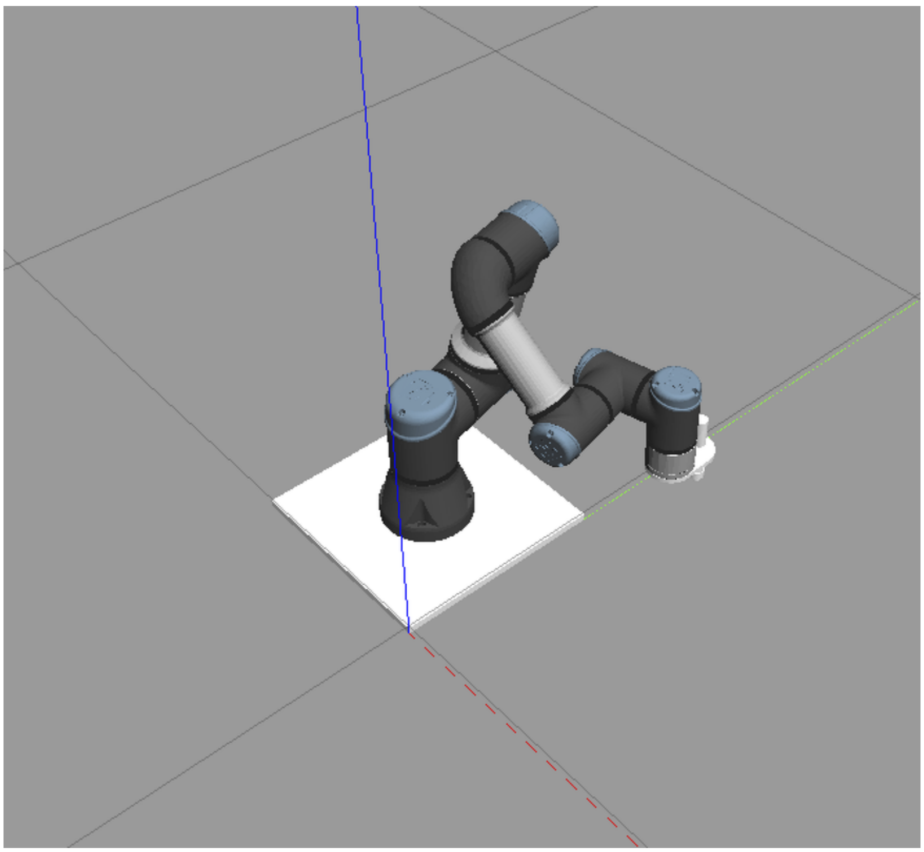

Figure 2: Test 2 image

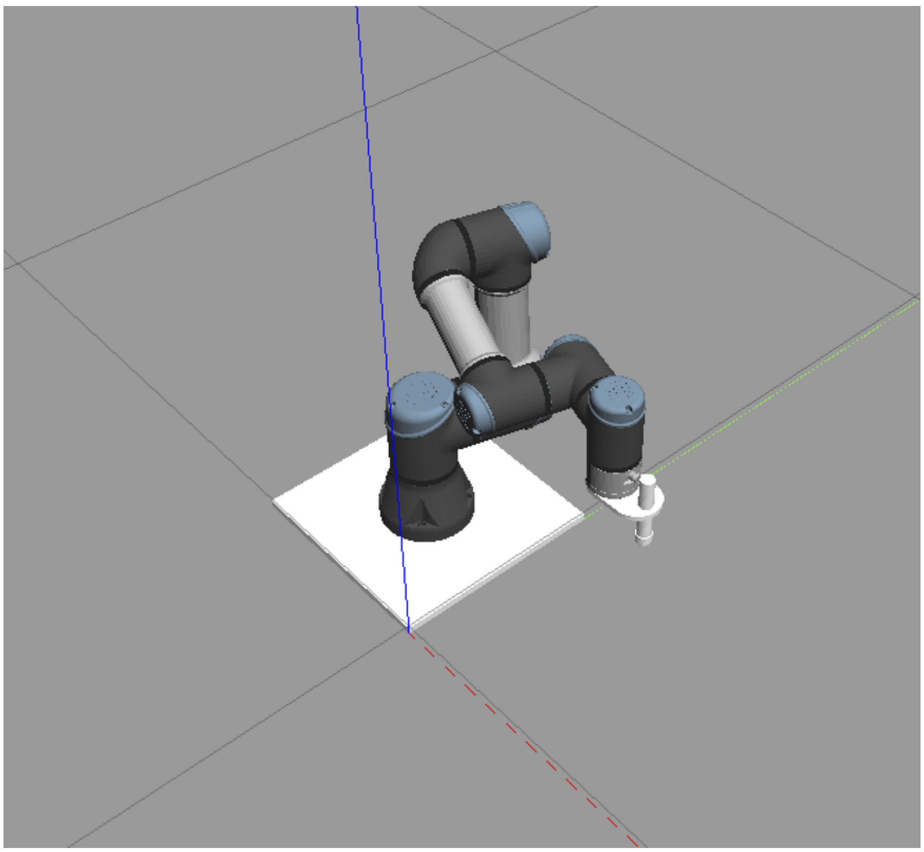

Figure 3: Test 3 image

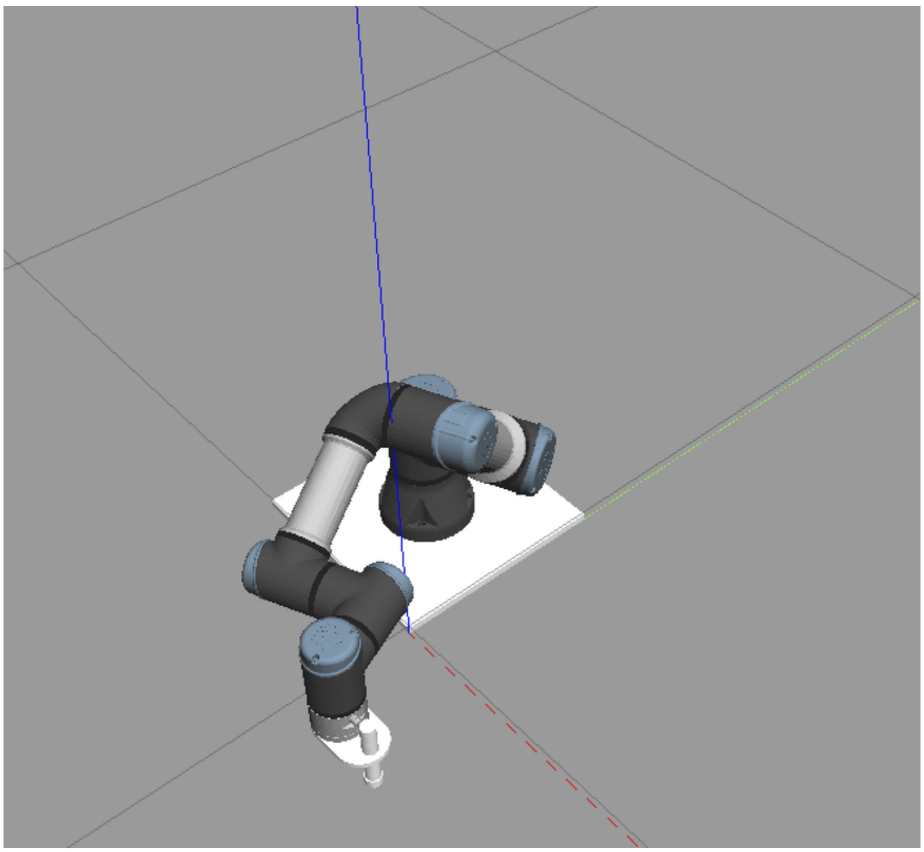

Figure 4: Test 4 image

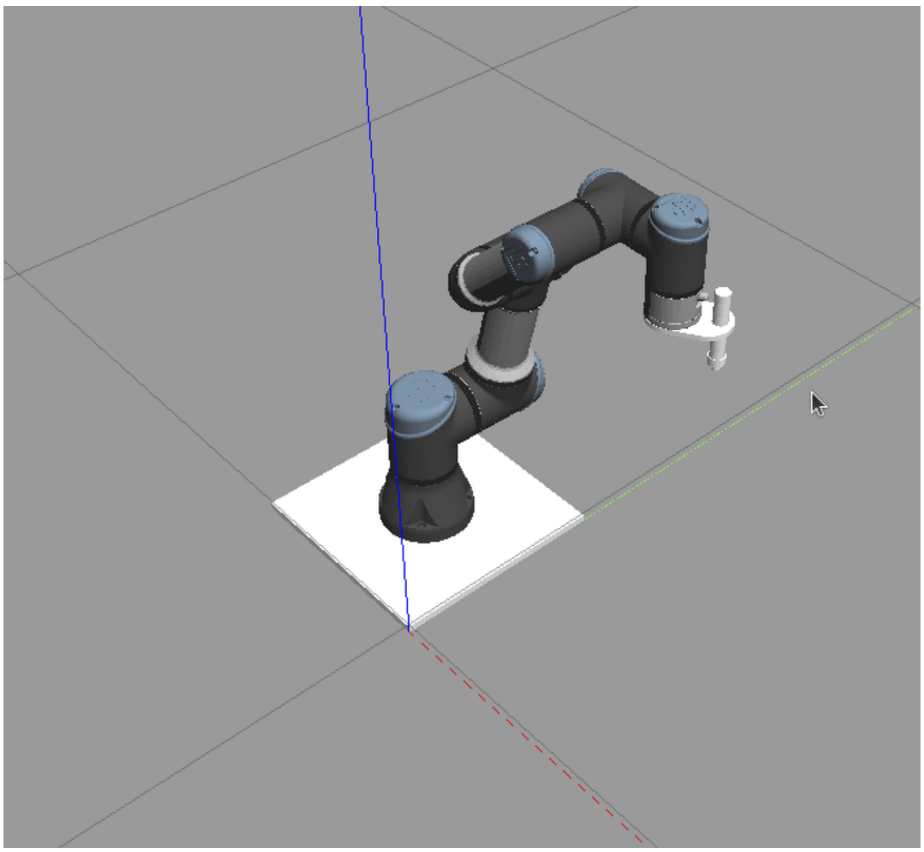

Figure 5: Test 5 image