Perception and Decision Making in Intelligent Systems
Homework 4: A Robot Manipulation Framework
Report

1. About task 1 (15%)

1.1 Briefly explain how you implement **your_fk()** function **(3%)**

First, we get the transformation from joint 0 to end effector. For each joint, we can get the transformation matrix to the next one by using DH params, and propagate the transformation until we meet the end effector. Note that I store each cumulative transformation matrix for calculating the Jacobian matrix.

```python
#### your code ####
# get transformation matrix
cumulative_T_list = []
cumulative_T = np.eye(4)
for i in range(len(DH_params)):
    theta = q[i]
    a = DH_params[i]['a']
    d = DH_params[i]['d']
    alpha = DH_params[i]['alpha']
    # current transformation matrix
    T = np.array([
        [np.cos(theta), -np.sin(theta)*np.cos(alpha), np.sin(theta)*np.sin(alpha), a*np.cos(theta)],
        [np.sin(theta), np.cos(theta)*np.cos(alpha), -np.cos(theta)*np.sin(alpha), a*np.sin(theta)],
        [0, np.sin(alpha), np.cos(alpha), d],
        [0, 0, 0, 1]
    ])

    cumulative_T_list.append(cumulative_T)
    cumulative_T = cumulative_T @ T
A = A @ cumulative_T
```

Then, I can calculate the jacobian matrix of which ith column means the contribution of the change in ith joint to the end effector.

```python
# Get Jacobian
# ith column of the Jacobian matrix means the contribution of the ith joint to the end effector
for i in range(len(DH_params)):
    # update the Jacobain matrix
    joint_z = cumulative_T_list[i][:3, 2]
    joint_pose = cumulative_T[:3, 3] - cumulative_T_list[i][:3, 3]
    jacobian[:, i] = np.concatenate((cross(joint_z, joint_pose), joint_z), axis=0)
```

1.2 What is the difference between D-H convention and Craig's convention? **(2%)**

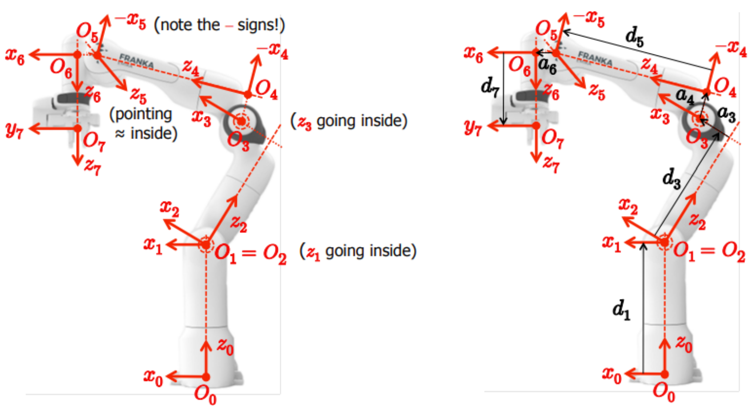
Both DH convention and Craig's provide 4 numbers that define the orientation of the *ith* link with respect to the *(i - 1)th* link. Both conventions define the z axis in the direction of the joint axis, the x-axis is parallel to the common normal, and the y-axis follows from x, z by right-handed coordinate system.

- Standard convention assumes ***ith coordinate frame*** is at the ***(i + 1)th joint***, while Craig's convention assumes that the ***ith coordinate frame*** is at the ***ith joint***. While
- *d*: In standard convention, *d* represents the distance along ***(i-1)th z-axis*** between the ***(i-1)th origin*** and the point where common normal intersects ***ith z***

axis. However, in Craig's convention, *d* represents the distance between ***(i-1)th x-axis*** and ***ith x-axis*** about ***ith z-axis***.

- θ: In standard convention, θ is the angle about the ***(i-1)th z-axis*** to align ***(i-1)th x-axis*** with the ***ith x-axis***. However, in Craig's convention, θ represents the angle between ***(i-1)th x-axis*** and ***ith x-axis*** about ***ith z-axis***.
- *a*: In standard convention, *a* represents the length along ***ith x-axis*** of the common normal between ***(i-1)th z*** and ***ith z-axis***. However, in Craig's convention, *a* represents the length along ***(i-1)th x-axis*** of the common normal between ***(i-1)th z*** and ***ith z-axis***.
- α: In standard convention, α represents the angle between ***(i-1)th z-axis*** and ***ith z-axis*** about ***ith x-axis***. However, in Craig's convention, α represents the angle between ***(i-1)th z-axis*** and ***ith z-axis*** about ***(i-1)th x-axis***.

1.3 Complete the D-H table in your report following **D-H convention (10%)**



The coordinate frames of the robot arm in this homework following D-H convention

A D-H table example format (please fill in it in your report)

i	d	alpha	a
1	d1	pi / 2	0
2	0	pi / 2	0
3	d3	pi / 2	a3
4	0	-pi / 2	-a4
5	d5	pi / 2	0
6	0	pi / 2	a6
7	d7	0	0

1. **About task 2 (10% + 5% bonus)**

2.1 Briefly explain how you implement **your_ik()** function **(5%)**

In each iteration, I estimate the current end effector pose with DH params and current joint.Then, I calculate the error of the current end effector pose and target end effector pose. And I multiply the pseudo inverse of the jacobian matrix with the error to update the joint angle (tmp_q). If the error is smaller than the threshold, I get the final joint angle.

```
#### your code ####
target_pose = np.array(new_pose) # 7D end effector target pose (position(x,y,z) quaternion(x,y,z,w))
# Pseudo Inverse Mthod
step_rate = 0.01
for iter in range(max_iters):
    current_pose, jacobian = your_fk(get_ur5_DH_params(), tmp_q, base_pos)
    d_error = target_pose - current_pose
    if(np.linalg.norm(d_error) < stop_thresh):
        break
    # delta_joint = pseudo inverse of jacobian * delta end effector error
    d_q = pinv(jacobian) @ d_error[:6] # When the degree is small, can omit the w in quaternion
    tmp_q += d_q * step_rate
```

2.2 What problems do you encounter and how do you deal with them? **(5%)**

The target pose we use is 7D (3 for position 4 for quanterion), but the Jacobian matrix is 6*6. So at first the matrix multiplication will fail. After I researched this problem, I realized that if we take a little step in joint angle, we can omit the last value of the error, which is *w* in quanterion.

2.3 Bonus! Do you also implement other IK methods instead of pseudo-inverse method? How about the results? **(5% bonus)**

Besides pseudo jacobian matrix method. I also use the Damped Least Square Method, which is a way of removing and reducing near singularity in the jacobian matrix and stabilizing error of joint. It can solve the least square solution more stable than the pseudo inverse method. I also found that it's more efficient than the pseudo inverse method.

```
# Bonus: Damped Least Square Method
for iter in range(max_iters):
    current_pose, jacobian = your_fk(get_ur5_DH_params(), tmp_q, base_pos)
    d_error = target_pose - current_pose
    if(np.linalg.norm(d_error) < stop_thresh):
        break
    # delta_joint = pseudo inverse of jacobian * delta end effector error
    tmp_q, success = leastsq(residual_func, tmp_q, args=(target_pose, base_pos))
```

Elapsed time comparison:

```
========================== Task 2 : Inverse Kinematic ==================
==========

- Testcase file : ik_test_case_easy.json
- Mean Error : 0.001744
- Error Count :   0 / 300
- Your Score Of Inverse Kinematic : 13.333 / 13.333

- Testcase file : ik_test_case_medium.json
- Mean Error : 0.001163
- Error Count :   0 / 100
- Your Score Of Inverse Kinematic : 13.333 / 13.333

- Testcase file : ik_test_case_hard.json
- Mean Error : 0.001175
- Error Count :   0 / 100
- Your Score Of Inverse Kinematic : 13.333 / 13.333

==========================================================================
==========
- Your Total Score : 40.000 / 40.000
==========================================================================
==========
Elapsed time for Pseudo-Inverse Method : 164.14642238616943 secs
```

```
========================== Task 2 : Inverse Kinematic ==================
==========

- Testcase file : ik_test_case_easy.json
- Mean Error : 0.001372
- Error Count :   0 / 300
- Your Score Of Inverse Kinematic : 13.333 / 13.333

- Testcase file : ik_test_case_medium.json
- Mean Error : 0.000467
- Error Count :   0 / 100
- Your Score Of Inverse Kinematic : 13.333 / 13.333

- Testcase file : ik_test_case_hard.json
- Mean Error : 0.000340
- Error Count :   0 / 100
- Your Score Of Inverse Kinematic : 13.333 / 13.333

==========================================================================
==========
- Your Total Score : 40.000 / 40.000
==========================================================================
==========
Elapsed time for Damped Least Square Method : 71.6645438671112 secs
```

pseudo-inverse method | damped least square method

2. About task 3 (5%)

This part uses the **your_ik()** function to control the robot and complete the block insertion task.

3.1 Compare your results between your_ik function and pybullet_ik

Speed: pybullet_ik > Damped Least-Square >> Pseudo-inverse

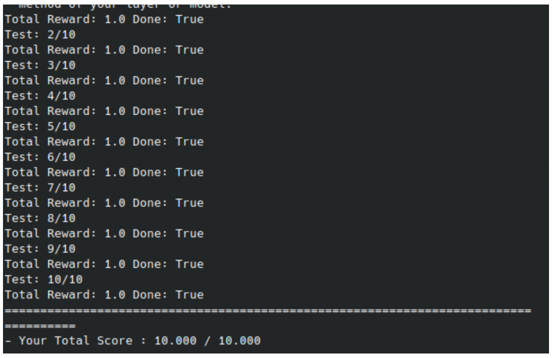
- *pybullet_ik*

```
Total Reward: 1.0 Done: True
Test: 2/10
Total Reward: 1.0 Done: True
Test: 3/10
Total Reward: 1.0 Done: True
Test: 4/10
Total Reward: 1.0 Done: True
Test: 5/10
Total Reward: 1.0 Done: True
Test: 6/10
Total Reward: 1.0 Done: True
Test: 7/10
Total Reward: 1.0 Done: True
Test: 8/10
Total Reward: 1.0 Done: True
Test: 9/10
Total Reward: 1.0 Done: True
Test: 10/10
Total Reward: 1.0 Done: True
==========================================================================
==========
- Your Total Score : 10.000 / 10.000
==========================================================================
```

- *my ik function (Pseudo-inverse)*

```
Total Reward: 1.0 Done: True
Test: 2/10
Total Reward: 1.0 Done: True
Test: 3/10
Total Reward: 1.0 Done: True
Test: 4/10
Total Reward: 1.0 Done: True
Test: 5/10
Total Reward: 1.0 Done: True
Test: 6/10
Total Reward: 1.0 Done: True
Test: 7/10
Total Reward: 1.0 Done: True
Test: 8/10
Total Reward: 1.0 Done: True
Test: 9/10
Total Reward: 1.0 Done: True
Test: 10/10
Total Reward: 1.0 Done: True
==========================================================================
==========
- Your Total Score : 10.000 / 10.000
==========================================================================
```

- *my ik function (DLS):*



```
Total Reward: 1.0 Done: True
Test: 2/10
Total Reward: 1.0 Done: True
Test: 3/10
Total Reward: 1.0 Done: True
Test: 4/10
Total Reward: 1.0 Done: True
Test: 5/10
Total Reward: 1.0 Done: True
Test: 6/10
Total Reward: 1.0 Done: True
Test: 7/10
Total Reward: 1.0 Done: True
Test: 8/10
Total Reward: 1.0 Done: True
Test: 9/10
Total Reward: 1.0 Done: True
Test: 10/10
Total Reward: 1.0 Done: True
==========================================================================
==========
- Your Total Score : 10.000 / 10.000
```

Reference:

- https://automaticaddison.com/the-ultimate-guide-to-jacobian-matrices-for-robotics/
- https://xiaobaidiy.github.io/2020/05/24/robot-arm-jacobian-matrix/
- https://www.diva-portal.org/smash/get/diva2:1018821/FULLTEXT01.pdf
- https://www.etedal.net/2014/03/dh-parameters.html