Perception and Decision Making in Intelligent Systems Homework 3

313554024 林慧旻

1. About task 1 (15%)

1.1 Briefly explain how you implement your_fk() function (3%)(You can paste the screenshot of your code and explain it)

```
def your_fk(DH_params : dict, q : list or tuple or np.ndarray, base_pos) -> np.ndarray:

    # robot initial position
    base_pose = list(base_pos) + [0, 0, 0]

    assert len(DH_params) == 6 and len(q) == 6, f'Both DH_params and q should contain 6 values,\n' \
                                                f'but get len(DH_params) = {DH_params}, len(q) = {len(q)}'

    A = get_matrix_from_pose(base_pose) # a 4x4 matrix, type should be np.ndarray
    jacobian = np.zeros((6, 6)) # a 6x6 matrix, type should be np.ndarray

    # ------------------------------------------------------------------------- #
    # --- TODO: Read the task description                                   --- #
    # --- Task 1 : Compute Forward-Kinematic and Jacobain of the robot by yourself --- #
    # ---          Try to implement `your_fk` function without using any pybullet  --- #
    # ---          API. (20% for accuracy)                                  --- #
    # ------------------------------------------------------------------------- #

    #### your code ####

    t_list = []
    cumulate_t = np.eye(4)
    for i in range(len(DH_params)):
        t_list.append(cumulate_t)
        theta = q[i]
        a = DH_params[i]['a']
        d = DH_params[i]['d']
        alpha = DH_params[i]['alpha']
        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(the
                       [0, np.sin(alpha), np.cos(alpha), d],
                       [0, 0, 0, 1]])

        cumulate_t = cumulate_t @ T
    A = A @ cumulate_t
    for i in range (len(DH_params)):
        p = cumulate_t[:3, 3] - t_list[i][:3, 3]
        z = t_list[i][:3, 2]
        v = np.cross(z, p)
        jacobian[:, i] = np.concatenate((v, z), axis=0)

    ###################################################

    # adjustment don't touch
    adjustment = np.asarray([[ 0, -1,  0],
                             [ 0,  0,  0],
                             [ 0,  0, -1]])
    A[:3, :3] = A[:3,:3]@adjustment
    pose_7d = np.asarray(get_pose_from_matrix(A,7))

    return pose_7d, jacobian
```

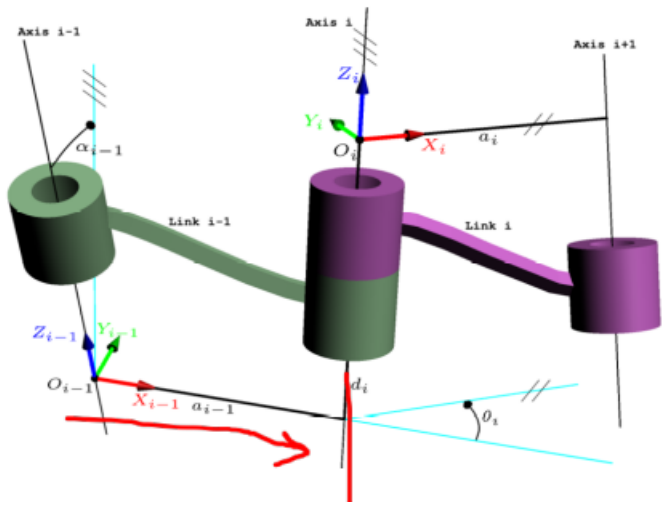
1. A list `t_list` is created to store intermediate transformation matrices for each joint's position, and `cumulate_t` is initialized to represent the commulated transformed matrix.
2. For each joint, using Denavit-Hartenberg (DH) parameters:
    - The joint angle `theta`, link offset `d`, link length `a`, and twist angle `alpha` are extracted from `DH_params`.
    - The individual transformation matrix `T` is computed for each joint based on DH parameters. This matrix represents the transformation from one joint to the next.
    - `cumulate_t` is updated by multiplying it with `T`, accumulating transformations from the base to the end-effector.
    - Each intermediate transformation `cumulate_t` is stored in `t_list` to calculate Jacobian later.

- End the loop, A is updated by multiplying with `cumulate_t`.

3. Jacobian Calculation:
   - `p` is the position of the end-effector minus the position of joint i, and `z` is the rotational axis derived from `t_list` for the Jacobian.
   - The linear velocity contribution of each joint is computed by taking the cross-product of `z` and `p`.
   - The Jacobian columns are populated by concatenating the linear and angular velocity vectors for each joint.

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

- **The four transformation steps of Modified D-H convention**

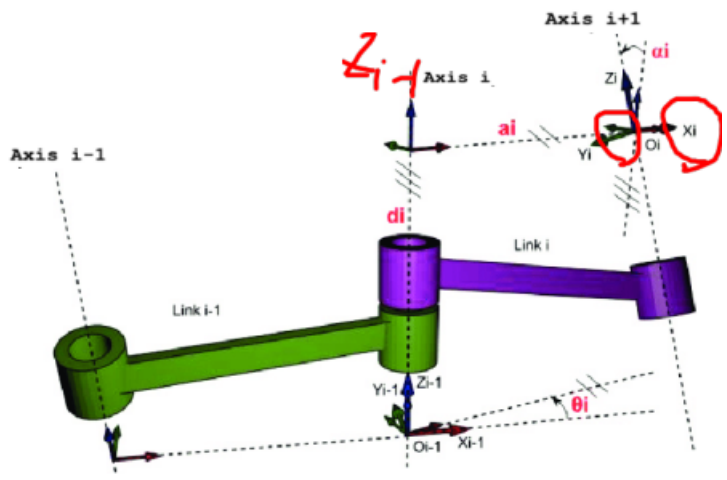


1. Set the initial coordinate frame Ai-1 at link i−1 so that the Zi−1 axis is parallel to Zi.
2. Rotate Xi−1 by angle αi−1 to align Zi−1 with Zi.
3. Translate along Zi (or align Zi with Zi+1), allowing a parallel transformation along Z.
4. Translate along Xi, transforming it to align with Ai.

This gives the transformation matrix:

$$ {}_{i}^{i-1}T = Rot(X, \alpha_{i-1})Trans(a_{i-1}, 0, 0)Rot(Z, \theta_i)Trans(0, 0, d_i) $$

- **The four transformation steps of D-H convention**



1. Set the initial coordinate frame Ai−1 at link i−1 so that the Zi−1 axis is parallel to Zi.
2. Rotate Xi−1 by angle αi−1 to align Zi−1 with Zi.
3. Translate along Zi (or align Zi with Zi+1), allowing a parallel transformation along Z.
4. Translate along Xi, transforming it to align with Ai.

This gives the transformation matrix:

$$ {}_{n+1}^{n}T = Rot(X, \theta_{n+1})Trans(0, 0, d_{n+1})Rot(X, \alpha_{n+1})Trans(a_{n+1}, 0, 0) $$

In conclustion, the main difference between them is that the D-H Convention aligns Zi, whereas the Modified D-H Convention aligns Zi with Zi−1. Both conventions have their constraints: in the D-H, the Xi-axis must be perpendicular to and intersect Zi−1, while in the Modified D-H, the Zi-axis is perpendicular to and intersect Xi−1.

reference: https://blog.csdn.net/hangl_ciom/article/details/102752733

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

i	d	α	a	θ (rad)

1	d1	π/2	0	θ_1
2	0	-π/2	0	θ_2
3	d3	π/2	a3	θ_3
4	0	-π/2	a4	θ_4
5	d5	π/2	0	θ_5
6	0	π/2	a6	θ_6
7	d7	0	0	θ_7

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

2.1 Briefly explain how you implement your_ik() function (5%)(You can paste the screenshot of your code and explain it)

```
def your_ik(robot_id, new_pose : list or tuple or np.ndarray,
                base_pos, max_iters : int=1000, stop_thresh : float=.001):



    joint_limits = np.asarray([
            [-3*np.pi/2, -np.pi/2], # joint1
            [-2.3562, -1],           # joint2
            [-17, 17],              # joint3
            [-17, 17],              # joint4
            [-17, 17],              # joint5
            [-17, 17],              # joint6
        ])

    # get current joint angles and gripper pos, (gripper pos is fixed)
    num_q = p.getNumJoints(robot_id)
    q_states = p.getJointStates(robot_id, range(0, num_q))

    tmp_q = np.asarray([x[0] for x in q_states][2:8]) # current joint angles 6d (You only need to modify this)

    # pseudo-inverse method
    step_rate = 0.01
    for _ in range(max_iters):

        cur_pose, jacobian = your_fk(get_ur5_DH_params(), tmp_q, base_pos)
        error = np.asarray(new_pose) - cur_pose
        if np.linalg.norm(error) < stop_thresh:
            break
        jacobian_pseudo_inverse = np.linalg.pinv(jacobian)
        delta_q = jacobian_pseudo_inverse @ error[:6]
        tmp_q += step_rate * delta_q
        tmp_q = np.clip(tmp_q, joint_limits[:, 0], joint_limits[:, 1])

    return list(tmp_q) # 6 DoF
```

Pseudo-Inverse Method:

- The function enters a loop to iteratively update joint angles until the target pose is reached or the maximum number of iterations (`max_iters`) is hit.
- **Forward Kinematics Calculation**: In each iteration, the current end-effector pose (`cur_pose`) and the Jacobian matrix (`jacobian`) are computed by calling `your_fk()` with the current joint angles (`tmp_q`).
- **Error Calculation**: The difference (`error`) between the target pose (`new_pose`) and the current pose (`cur_pose`) is calculated. If this error falls below a small threshold (`stop_thresh`), the loop breaks, indicating that the target pose has been achieved.
- **Jacobian Pseudo-Inverse**: The pseudo-inverse of the Jacobian (`jacobian_pseudo_inverse`) is calculated to allow for solving under-constrained systems.
- **Joint Angle Update**: The change in joint angles (`delta_q`) is calculated as `jacobian_pseudo_inverse @ error[:6]`. This update is scaled by a `step_rate` factor and added to `tmp_q` to iteratively adjust the joint angles toward the target.

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

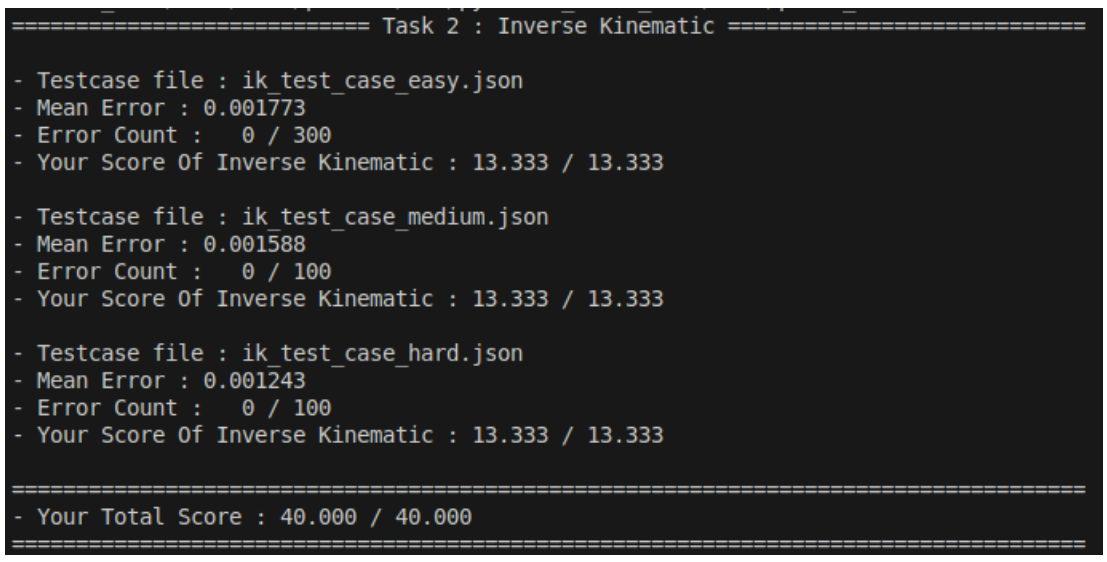
- **Problem**: During testing, it was observed that the function could sometimes return joint angles that exceeded the physical limits of the robot's joints, especially when trying to reach extreme poses.
- **Solution**: To fix this, a `joint_limits` check was added within the iteration loop. After updating `tmp_q`, the constraint was applied to clamp each joint angle within its respective limits, ensuring that the joint angles remained within the allowed range.

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

Yes, I also implemented jacobian_transpose method as follows:

```
def your_ik_bonus(robot_id, new_pose : list or tuple or np.ndarray,
                base_pos, max_iters : int=1000, stop_thresh : float=.001):



    joint_limits = np.asarray([
            [-3*np.pi/2, -np.pi/2], # joint1
            [-2.3562, -1],           # joint2
            [-17, 17],              # joint3
            [-17, 17],              # joint4
            [-17, 17],              # joint5
            [-17, 17],              # joint6
        ])

    # get current joint angles and gripper pos, (gripper pos is fixed)
    num_q = p.getNumJoints(robot_id)
    q_states = p.getJointStates(robot_id, range(0, num_q))

    tmp_q = np.asarray([x[0] for x in q_states][2:8]) # current joint angles 6d (You only need to modify this)

    step_rate = 0.1
    for _ in range(max_iters):

        cur_pose, jacobian = your_fk(get_ur5_DH_params(), tmp_q, base_pos)
        error = np.asarray(new_pose) - cur_pose
        if np.linalg.norm(error) < stop_thresh:
            break
        jacobian_transpose = jacobian.T
        delta_q = jacobian_transpose @ error[:6]
        tmp_q += step_rate * delta_q
                tmp_q = np.clip(tmp_q, joint_limits[:, 0], joint_limits[:, 1])


    return list(tmp_q) # 6 DoF
```

Result:



3. 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

`your_ik`

```
========================== Task 3 : Transporter Network ==========================

Test: 1/10
WARNING:tensorflow:From /home/user/pdm-f24/hw3/ravens/ravens/agents/transporter.py:167
Instructions for updating:
Simply pass a True/False value to the `training` argument of the `__call__` method of
W1026 19:33:22.928828 138168745157760 deprecation.py:323] From /home/user/pdm-f24/hw3/
ter 2020-10-11.
Instructions for updating:
Simply pass a True/False value to the `training` argument of the `__call__` method of
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
- Time: 441.21252393722534
============================================================================================
```

`pybullet_ik`

```
========================== Task 3 : Transporter Network ==========================

Test: 1/10
WARNING:tensorflow:From /home/user/pdm-f24/hw3/ravens/ravens/agents/transporter.py:167
Instructions for updating:
Simply pass a True/False value to the `training` argument of the `__call__` method of
W1026 19:31:05.813097 133519207253120 deprecation.py:323] From /home/user/pdm-f24/hw3/
ter 2020-10-11.
Instructions for updating:
Simply pass a True/False value to the `training` argument of the `__call__` method of
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
- Time: 28.118496417999268
============================================================================================
```

Both the results can finish all 10 tests as shown above. However, the time `your_ik` took is much longer than that of `pybullet_ik`. The visualization results of `your_ik` also performs much slower than the other.