HW4

311512064 鄧書桓

Task1:

```
(pybullet_env) widden@widden-desktop:~/NCTU/Perception_and_Decision_Making/hw4/P
DMS-HW4$ python fk.py
pybullet build time: May 20 2022 19:43:01
============================ Task 1 : Forward Kinematic ======================
====

- Testcase file : fk_testcase.json
- Your Score Of Forward Kinematic : 10.000 / 10.000, Error Count :    0 /  900
- Your Score Of Jacobian Matrix   : 10.000 / 10.000, Error Count :    0 /  900

================================================================================
====
- Your Total Score : 20.000 / 20.000
================================================================================
====
```

這邊僅附上結果得分圖，想法與如何實作將會在 task 做說明。

Task2:

```
- Your Score Of Inverse Kinematic : 4.333 / 10.000

================================================================================
====
- Your Total Score : 34.333 / 40.000
================================================================================
====
```

這邊僅附上結果得分圖，想法與如何實作將會在 task 做說明。

Task3:

1. code about fk.py

```
#### your code ####
T = np.empty([7, 4, 4])
for i in range(7):
    theta = q[i]
    a = DH_params[i]['a']
    d = DH_params[i]['d']
    alpha = DH_params[i]['alpha']
    T[i] = np.array([[math.cos(theta), -math.sin(theta), 0, a],
                     [math.sin(theta)*math.cos(alpha), math.cos(theta) *
                         math.cos(alpha), -math.sin(alpha), -d*math.sin(alpha)],
                     [math.sin(theta)*math.sin(alpha), math.cos(theta) *
                         math.sin(alpha), math.cos(alpha), d*math.cos(alpha)],
                     [0, 0, 0, 1]
                     ])
T_total = T[0]@T[1]@T[2]@T[3]@T[4]@T[5]@T[6]
A = A@T_total
# jacobian = ? # may be more than one line
for i in range(7):
    if i == 0:
        t_i = T[0]
    else:
        t_i = t_i @ T[i]
    z_i = t_i[0:3, 2]
    p_i = T_total[0:3, 3] - t_i[0:3, 3]
    J_Li = cross(z_i, p_i)
    J_Ai = z_i
    jacobian[:, i] = np.append(J_Li, J_Ai)
    # -45 degree adjustment along z axis
    # details : see "pybullet_robot_envs/panda_envs/robot_data/franka_panda/panda_model.urdf"
adjustment = R.from_rotvec(
    [0, 0, -0.785398163397]).as_matrix()  # Don't touch
A[:3, :3] = A[:3, :3] @ adjustment  # Don't touch
###################
```

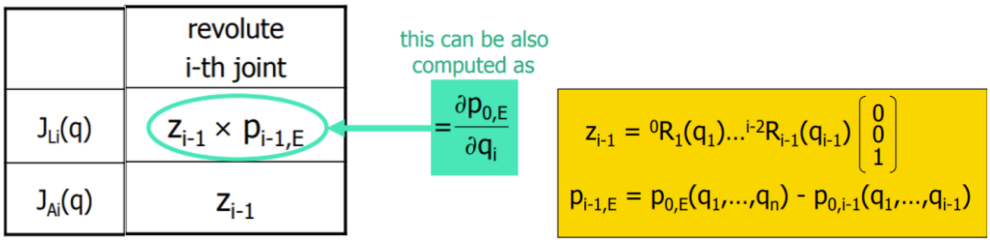
1.1

A:

$$\begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & a_i \\ sin(\theta_i)cos(\alpha_i) & cos(\theta_i)cos(\alpha_i) & -sin(\alpha_i) & -d_i sin(\alpha_i) \\ sin(\theta_i)sin(\alpha_i) & cos(\theta_i)sin(\alpha_i) & cos(\alpha_i) & d_i cos(\alpha_i) \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

這邊根據已知的 DH model、已知每個時間點個軸的轉速與投影片上推導出的轉移函數式子(上面的式子)可以得到各個轉軸間的關係(這邊利用 for 迴圈來實現個軸的關係 T)，並以此來獲得題目所要求的 A，也就是末端點的 pose。

Jacobin:



這邊的 Jacobian 是利用幾何解的形式來求得的，根據上課推導出(化簡)的式子能夠得出上面的關係式，接著利用 for 迴圈與上方的式子能夠得到每個軸各自的線速度與角速度。

1.2

D-H convention 也稱為 classic D-H convention，其將 Oi 定義在第 i+1 軸、將 a 與 alpha 定義為第 i 至 i+1 軸之間的變化；Craig's convention 則是將 Oi 定義在第 i 軸、將 a 與 alpha 定義為第 i-1 至 i 軸之間的變化。這樣的差異會導致同個 robot 的 D-H model 不同，進而影響轉移矩陣的內容與其前乘後乘的順序。

1.3

i	d	α(rad)	a	θ_i(rad)
1	d_1	0	0	θ_1
2	0	$-\pi/2$	0	θ_2
3	d_3	$\pi/2$	0	θ_3
4	0	$\pi/2$	a_3	θ_4
5	d_5	$-\pi/2$	a_4	θ_5
6	0	$\pi/2$	0	θ_6
7	d_7	$\pi/2$	a_6	θ_7

2. code about ik.py

```
dh_params = get_panda_DH_params()
your_pose, your_jacobian = your_fk(
            robot, dh_params, tmp_q)
iters = 0
stepsize = 0.005
while(iters < max_iters):
    # err = np.asarray(pose_7d_to_6d(new_pose)) - np.asarray(pose_7d_to_6d(your_pose))
    # delta_x = np.asarray(pose_7d_to_6d(new_pose-your_pose))
    delta_matrix = get_matrix_from_pose(new_pose) @ np.linalg.inv(get_matrix_from_pose(your_pose))
    delta_x = get_pose_from_matrix(delta_matrix, 6)

    jaconian_T = np.transpose(your_jacobian)
    delta_theta = stepsize * jaconian_T @ np.linalg.inv(your_jacobian @ jaconian_T) @ delta_x
    tmp_q = tmp_q + delta_theta
    for i in range(7):
        if (tmp_q[i] < joint_limits[i][0]):
            tmp_q[i]  = joint_limits[i][0]
        elif(tmp_q[i] > joint_limits[i][1]):
            tmp_q[i]  = joint_limits[i][1]
    iter_state = False
    # for i in range(6):
    #     if(delta_x[i] > stop_thresh):
    #         iter_state = False
    if (np.linalg.norm(delta_x) < stop_thresh):
        iter_state = not iter_state
    if (iter_state):
        print("converge")
        break
```

```
    else:
        your_pose, your_jacobian = your_fk(
            robot, dh_params, tmp_q)
    iters = iters + 1
    if (iters == max_iters):
        print("failed to converge")
return list(tmp_q) + list(gripper_pos) # 9 DoF
```

2.1

$$\Delta\boldsymbol{\theta} = \alpha J^T(\boldsymbol{\theta})\left(J(\boldsymbol{\theta})J^T(\boldsymbol{\theta})\right)^{-1}\Delta\mathbf{x} = J^{\#}\Delta\mathbf{x}$$

這邊是使用疊代法來找 inverse Jacobian，會不斷逼近想要到的 pose 並直到誤差小於一個固定值才會停下來。在每次疊代中，都會使用到上面這個公式，其中的 delta(x)求得的方式較不直觀，是使用向量(matrix)的形式來實現的，並利用下面這個概念: matrix 型態的 delta(x) @ matrix 型態的 your_pose - matrix 型態的 new_pose，得到 delta(x)之後只要根據上方的算式即可得到 delta(theta)。接著要考慮變更後的角速度是否會超過 joint limit 與其誤差是否小於所設的疊代誤差。最終，當疊代至誤差小於固定值後，其所生成的 inverse Jacobin 與各軸角速度即是我們所想要的。

另外，我還有設計 stepsize，用來將每次疊代過後的修正變得更小，以免出現震盪導致到不了目標點的現象。

2.2

我一開始的 delta(x)是直接利用當下位置的 6D pose 與目標位置的 6D pose 相減而來，但是這樣得到的結果會超級差，在我將 delta(x)印出來後發現在疊代的途中，僅有 x,y,z 會變小，角度都不太會收斂，不太確定是什麼原因，但我認為是因為角度無法直接相減所求得，需要利用上述類似向量的方式才能將其考慮進去。

另外，調整 max iteration 與 stop threshold 的效果很好，若僅使用預設值的話，最終的成果都不太理想，無法得到較高的分數，但直到我將值設置為 0.01 時總分能接近 40 分。

3. manipulation.py

3.1

i. get_src2dst_transform_from_kpts:

首先先將各自不同 frame 的 dis 與 src keypoints 利用 extrinsic 轉到一樣的座標(world frame)，接著利用 SVD 分解的特性找出轉移與平移矩陣，以此將 src 的關鍵點轉移到 dis 上。值得注意的是，本來利用 SVD 尋找相同 key point 的方法必須在同個 frame 下，因此，這編譯開始才需要使用外參將基於各自相機 frame 的關鍵點轉置世界座標。

ii. Template_gripper_transform:

這邊的 transform 為從 object frame 到 gripper frame ，而這個是從助教給的.jason 而來的，並不用自己求得。另外，此用途為之後做夾取時(robot_dense_action)會用到，到時會需要用到它的反矩陣，也就是 gripper frame to object frame，以此來獲得該物件要被夾的地方位於 gripper pose 中為何點。

3.2

Minimum number of keypoint:

至少需要 3 個點，由於此 robot 的操作是位於三維空間中，因此至少需要三個點才能成功計算出轉移矩陣。

3.3

我認為造成誤差的最大原因為 3 個圖的 keypoint 圖點的不夠準確造成的，儘管這三張圖已經算是滿大的圖了，但是是人的話難免會有誤差，只要稍微有偏差就會導致 pose matching 不準確。改進的方式的話為由 ML 的方式來尋找物品的 keypoint，以此來取代人為手動給予 keypoint，這樣能確保同個物品給的 keypoint 相同率較高，進而提高 matching accuracy。