Homework 4: A Robot Manipulation Framework

task 1

1.1 Briefly explain how you implement your_fk() function

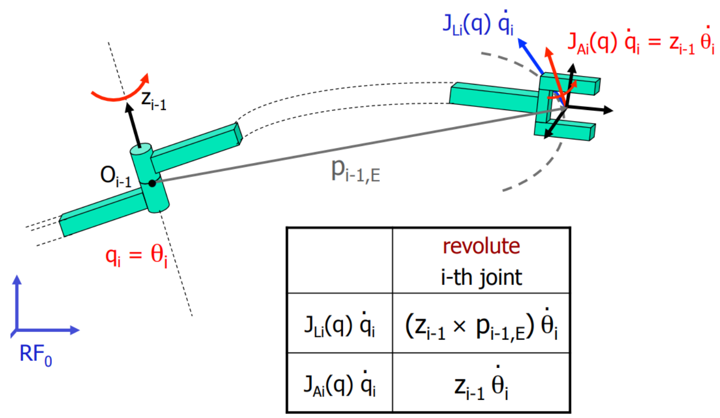
A

```
# A = ? # may be more than one line
trans_ = []
T6 = np.eye(4)
for i, dh in enumerate(DH_params):
    Ti = np.array([[np.cos(q[i]), -np.sin(q[i]), 0, dh['a']],
    [np.sin(q[i])*np.cos(dh['alpha']), np.cos(q[i])*np.cos(dh['alpha']), -np.sin(dh['alpha']), -dh['d']*np.sin(dh['alpha'])],
    [np.sin(q[i])*np.sin(dh['alpha']), np.cos(q[i])*np.sin(dh['alpha']), np.cos(dh['alpha']), dh['d']*np.cos(dh['alpha'])],
    [0, 0, 0, 1]])
    A = A.dot(Ti)
    T6 = T6.dot(Ti)
    trans_.append(Ti)
```

因為順向運動學主要是用到modify的方式，因此我將DH參數代如其矩陣公式，並且每一次都用後乘的方式，找到不同joint座標的轉換，最終可以得到手臂最終點與base的相對關係

jacobian

在這次作業中所提供的資料是使用Geometric Jacobian透過參考資料得知如何去分別處理各軸對卡式座標的影響



	revolute i-th joint
$J_{Li}(q)\,\dot{q}_i$	$(z_{i-1} \times p_{i-1,E})\,\dot{\theta}_i$
$J_{Ai}(q)\,\dot{q}_i$	$z_{i-1}\,\dot{\theta}_i$

且這次實作的手臂皆為選轉軸透過以上公式有以下程式碼

```
# jacobian = ? # may be more than one line

trans_temp = np.eye(4)
p_ee = T6[0:3,3]
for i in range(7):
    trans_temp = trans_temp.dot(trans_[i])
    R_ = trans_temp[0:3,0:3]
    P = trans_temp[0:3,3]
    zi_temp = R_.dot([0,0,1])
    pi_temp = cross(zi_temp, (p_ee - P))
    jacobian[0:3,i] = pi_temp
    jacobian[3:6,i] = zi_temp
```

trans_是一個list我存下每一軸的的轉移矩陣，而剛好與公式相同，每推進一個軸，期必須找到這個軸相對於基座標的座標關係，並且找到其旋轉的部分，再透過這個旋轉軸的方向z與末端點做cross可以找到這個軸對末端點所造成的xyz的變化。

1.2 What is the difference between D-H convention and Craig’s convention?

Classic D-H Conventions: coordinates of Oi is put on axis i+1

Modified D-H Conventions: coordinates of Oi is put on the axis i, not the axis i+1

因此最大的差異是在其座標系所設定的位置不同

1.3 Complete the D-H table in your report following D-H convention

i	d	α (rad)	a	θ_i (rad)
1	d_1	0	0	θ_1
2	0	$-\frac{\pi}{2}$	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

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

task 2

2.1 Briefly explain how you implement your_ik() function

Jacobian是將joint space的微分傳換成Cartesian，且此關係是非線性的，那我們因此可以用invers-Jacobian找到Cartesian轉換到joint space，然後用不斷迭代的方式去逼近特定Cartesian相對應的joint，需要迭代的原因是，我們不能一次就得到答案，Cartesian與joint space為非線性，只有在很小的變化下可趨近於線性

$$\left\|F\left(q^N\right) - x^d\right\| < \epsilon$$

每次用The Pseudo Inverse Method找出一組微小的dq去更新當前的q並且放到順向運動學來去跟我們要找得位置x去做比較，當其norm小於伐值時我們就停止跌代

```python
def base_line(delta_x, stop_thresh):

    Norm = la.norm(delta_x)
    if(Norm >= stop_thresh):
        return True, Norm
    else:
        return False, Norm
```

因為要以迭代得方法去做逆向運動學，我判斷得條件獨立寫成一個函式，當小於伐值時會跳出迭代的迴圈，也就是norm小於伐值時

```python
dh_params = get_panda_DH_params()
iters = 0
step_size = 0.05
flag = True
while(flag and (iters<=max_iters)):
    pose, jacobian = your_fk(robot, dh_params, tmp_q)
    #delta_x using matrix
    delta_matrix =get_matrix_from_pose(new_pose)@la.inv(get_matrix_from_pose(pose))
    delta_x = get_pose_from_matrix(delta_matrix, 6)
    #Pseudo Inverse
    j = jacobian
    jt = np.transpose(jacobian)
    delta_q = step_size*jt @ la.inv(j@jt)@delta_x
    tmp_q = tmp_q+ delta_q
    #Joint limitation
    for i , limits in enumerate(joint_limits):
        if(tmp_q[i]<limits[0]):
            tmp_q[i]=limits[0]
        elif(tmp_q[i]>limits[1]):
            tmp_q[i]=limits[1]
    flag, Norm = base_line(delta_x, stop_thresh)
    iters+=1
```

上圖是ik主要的實做程式碼，主要是利用Pseudo Inverse jacobian來求得dx到dq的關係，那每一次要讓這個非線性的關係變成線性，程式碼中的step_size就是alpha

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

那這邊的重點，就是delta_x的算法，我們是4*4矩陣來表示座標，並將兩個座標做inverse得到當前座標到目的座標的轉移矩陣，將其轉回原本的6D pose

其中一個重點就是每次都要去檢查是否有超過邊界角度，若超過就要用邊界角度代替，因此會用一個迴圈去做檢查。

2.2 What problems do you encounter and how do you deal with them?

在一開始的時候會發現一直沒有辦法收斂，我把值給print出來發現，主要都是roll,pitch,yaw的問題，後來才發現，當我沒有把座標以矩陣形式去找delta_x會出現邊界問題，例如178度和-180度其實只差2度但會算成358度的差，用矩陣去處理delta_x後問題就會被改善了

manipulation.py

3.1 Briefly explain how get_src2dst_transform_from_kpts() function works for pose matching and how template_gripper_transform work for gripper pose estimation in manipulation.py

get_src2dst_transform_from_kpts()

這個函式最主要的目的就是找到兩個視角下的轉換關系，跟第一次作業的BEV projection的概念一樣，因為我們會在不同視角下的圖片裡，手動點下一樣的點，可以透過這些點的座標去找到這些不同照片不同視角下的轉換矩陣，其中使用了SVD的方法去找到兩個座標的轉移矩陣

template_gripper_transform

因為可以從np.asarray(template_grasping['obj2gripper'])找到obj到girpper座標的轉移矩陣，那將其inverse便可以得到gripper到object的轉移矩陣，那將這個轉移矩陣將會在robot_dense_action()裡面被用到，拿來給手臂移動做使用

3.2 What is the minimum number of keypoints we need to ensure the program runs properly? Why?

從實驗裡面來看，至少要3個點才不會導致失敗，原因我認為手臂是在一個**3維空間**工作，則至少需要3個點才有足夠的資訊量。

3.3 Briefly explain how to improve the matching accuracy

可以更好配對的前提就是至少點要點的夠好，兩張圖中被認定為一樣點的點必須對應到相同位置，而且我覺的可以點多於3個點，讓資訊多一點，可是也不能太多，因為會造成誤差過大，5個點是比較合適的數量，其中因為座標與座標之間得相對關係是利用SVD來做轉換，可以針對轉換關係計算去優化配對精確度。