312512063_張祐維 robotics_project1

How to run

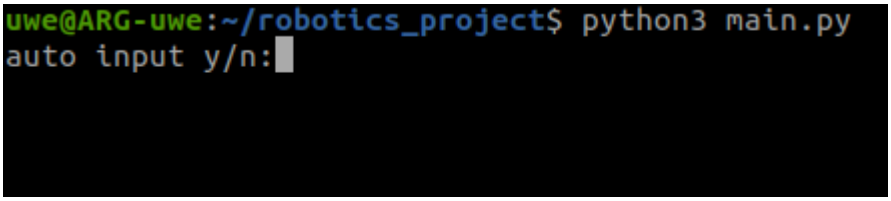
Use ubuntu open terminal and enter the following commands...
Make sure your ubuntu has python3.8 above.

```
source environment.sh
python3 main.py
```

source environment.sh to install numpy.

Running the code

First you'll see...

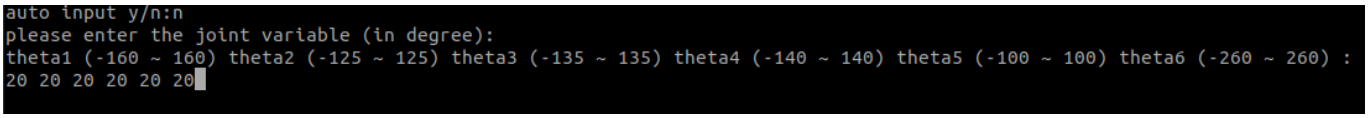


Enter "y" or "n" to chose auto input or not.

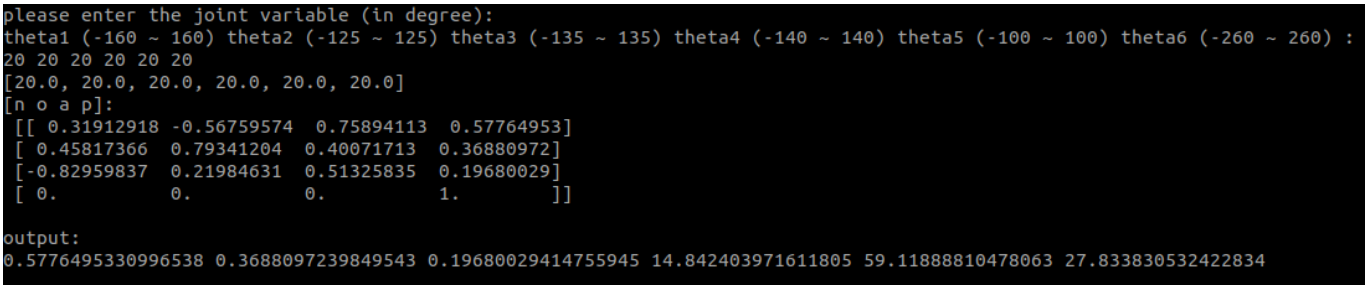
Question1

Copy the following string for forward kinematric input(if your not auto input):

```
20 20 20 20 20 20
20 -53.687 165.1699 171.566 52.9026 -136.0080
```



Then the output should be...



or

```
please enter the joint variable (in degree):
theta1 (-160 ~ 160) theta2 (-125 ~ 125) theta3 (-135 ~ 135) theta4 (-140 ~ 140) theta5 (-100 ~ 100) theta6 (-260 ~ 260) :
20 -53.687 165.1699 171.566 52.9026 -136.0080
[20.0, -53.687, 165.1699, 171.566, 52.9026, -136.008]
theta3 is out of range! range: -135~135, value: 165.1699
theta4 is out of range! range: -140~140, value: 171.56599999999997
[n o a p]:
 [[-0.52236705  0.38879603  0.75892708  0.57494129]
 [-0.09598119 -0.91115901  0.40072044  0.367824  ]
 [ 0.84730176  0.13648043  0.51327655  0.20813873]
 [ 0.          0.          0.          1.        ]]

output:
0.5749412880619346 0.36782400340412535 0.20813873260288454 170.8495922338977 59.1176735613057 27.834463522165116
```

Question2

Copy the following string for inverse kinematric input(Cartesian matrix) (if your not auto input)

```
0.10575416,-0.64251414,0.75894113,0.57764953/0.70190531,0.58885882,0.400717
13,0.36880972/-0.7043756,0.49032731,0.51325835,0.19680029/0.,0.,0.,1.
```

```
please enter Cartesian point:
col separate by ,
row separate by /

0.10575416,-0.64251414,0.75894113,0.57764953/0.70190531,0.58885882,0.40071713,0.36880972/-0.7043756,0.49032731,0.51325835,0.19680029/0.,0.,0.,1.
```

then the output should be...

```
[[ 0.10575416 -0.64251414  0.75894113  0.57764953]
 [ 0.70190531  0.58885882  0.40071713  0.36880972]
 [-0.7043756   0.49032731  0.51325835  0.19680029]
 [ 0.          0.          0.          1.        ]]
======times:1========
Corresponding Variable (theta1, theta2, theta3, theta4, theta5, theta6)
19.999999770073902 20.0000008690169 19.99999869320877 19.999999681169054 20.000000591109426 20.000000241400972
======times:2========
theta2 is out of range! range: -125~125, value: -127.2131453354694
Corresponding Variable (theta1, theta2, theta3, theta4, theta5, theta6)
-134.88628593447376 -127.2131453354694 19.99999869320877 19.182428858472317 50.88301405871323 -166.60854991899086
======times:3========
theta3 is out of range! range: -135~135, value: 165.2891547050245
theta4 is out of range! range: -140~140, value: 171.67672416269184
Corresponding Variable (theta1, theta2, theta3, theta4, theta5, theta6)
19.999999770073902 -52.786854664530594 165.2891547050245 171.67672416269184 53.90983566939056 -136.1927769224759
======times:4========
theta3 is out of range! range: -135~135, value: 165.2891547050245
theta2 is out of range! range: -125~125, value: 159.99999913098313
theta4 is out of range! range: -140~140, value: 146.2403252271089
Corresponding Variable (theta1, theta2, theta3, theta4, theta5, theta6)
-134.88628593447376 159.99999913098313 165.2891547050245 146.2403252271089 27.306165446071077 56.47860370818547
uwe@ARG-uwe:~/robotics_project$
```

Program Architecture Explanation

Program Flow

The program flow revolves around the classes defined in robot_model.py.
Specifically, the Joint class encapsulates the parameters of a robotic joint, such as the Denavit-Hartenberg (DH) parameters and the joint angle range.
The puma560 class, representing the PUMA 560 robot, initializes six instances of the Joint class to model its joints.
The crucial aspect of the program's flow is the calculation of forward and inverse kinematics.

Core Code Explanation

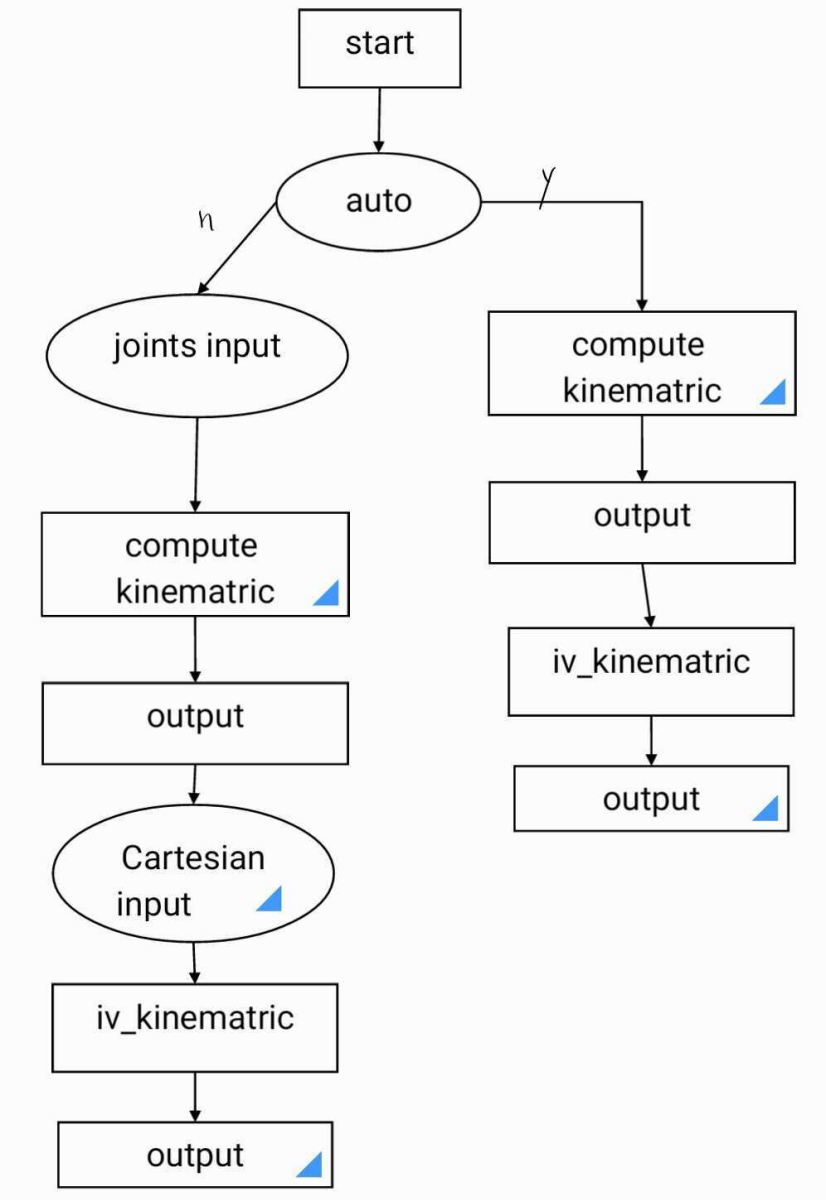
Forward Kinematics

The kinematics method in the Joint class computes the Denavit-Hartenberg (DH) transformation matrix for the corresponding joint.
It uses trigonometric functions (cosine and sine) to calculate the elements of the transformation matrix based on the joint parameters (theta, alpha, a, and d).
The resulting transformation matrix represents the position and orientation of the joint in the robot's coordinate system.

Inverse Kinematics

The dh_inverse_kinematics method in the puma560 class implements an inverse kinematics solution.
It takes a desired end-effector pose in the form of a 4x4 homogeneous transformation matrix (matrix).
The method iteratively explores possible joint angles for the robot to reach the specified pose.
This process involves solving trigonometric equations and considering multiple solutions due to the nature of trigonometric functions.
The calculated joint angles are then printed as a result.

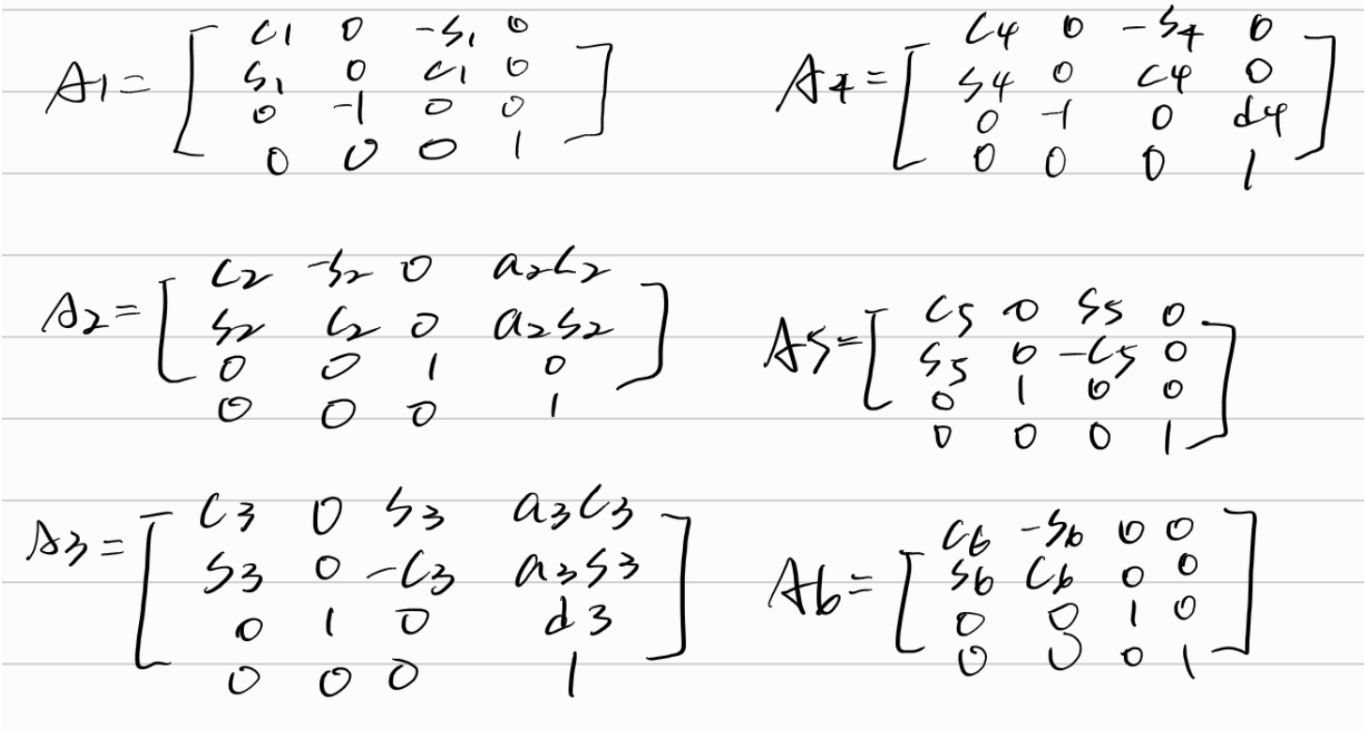
Program running process



Mathematical operations instructions

Forward kinematric

Every times joint's theta got change it'll run Joint::kinamatics()
Then the joint will update his transform matrix with the fololwing image.



Inverse kinematric

When input by Cartesian matrix we need to get each joint's theta.
It'll run puma560::dh_inverse_kinematics()
Then the thetas' equation need to be pre-program with the following equations

θ_1's

$$A_1^{-1}T_6 = {}^1T_6 = \prod_{i=2}^{6} A_i$$

$$\Rightarrow \begin{bmatrix} c_1 & s_1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ -s_1 & c_1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n & o & a & p \\ 0 & 0 & 0 & 1 \end{bmatrix} = {}^1T_6 = \begin{bmatrix} - & - & - & s_{23}d_4 + c_{23}a_3 + a_2c_2 \\ - & - & - & -c_{23}d_4 + s_{23}a_3 + a_2s_2 \\ - & - & - & d_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Rightarrow -s_1P_x + c_1P_y = d_3 \quad , P = \sqrt{P_x^2 + P_y^2}$$

$$\Rightarrow \frac{P_y}{P}c_1 - \frac{P_x}{P}s_1 = \frac{d_3}{P}$$

$$\Rightarrow \cos\left(\theta_1 + \tan^{-1}\left(\frac{P_x}{P_y}\right)\right) = \frac{d_3}{P}$$

$$\therefore \sin\left(\theta_1 + \tan^{-1}\left(\frac{P_x}{P_y}\right)\right) = \pm\sqrt{1 - \frac{d_3^2}{P^2}}$$

$$\Rightarrow \theta_1 + \tan^{-1}\left(\frac{P_x}{P_y}\right) = \tan^{-1}\left(\pm\sqrt{1 - \frac{d_3^2}{P^2}} \Big/ \frac{d_3}{P}\right) \Rightarrow \theta_1 = -\tan^{-1}\left(\frac{P_x}{P_y}\right) + \tan^{-1}\left(\frac{\sqrt{P_x^2 + P_y^2 - d_3^2}}{d_3}\right) \ast$$

θ_3's

$$c_1P_x + s_1P_y = s_{23}d_4 + c_{23}a_3 + a_2c_2 \quad \text{①}$$

$$-P_z = -c_{23}d_4 + s_{23}a_3 + a_2s_2 \quad \text{②}$$

$$-s_1P_x + c_1P_y = d_3 \quad \text{③}$$

$$a_3c_3 + d_4s_3 = \frac{P_x^2 + P_y^2 + P_z^2 - a_2^2 - a_3^2 - d_3^2 - d_4^2}{2a_2} = M$$

$$\therefore \theta_3 = \tan^{-1}\left(M \Big/ \left(\pm\sqrt{a_3^2 + d_4^2 - M^2}\right)\right) - \tan^{-1}\left(\frac{a_3}{d_4}\right)$$

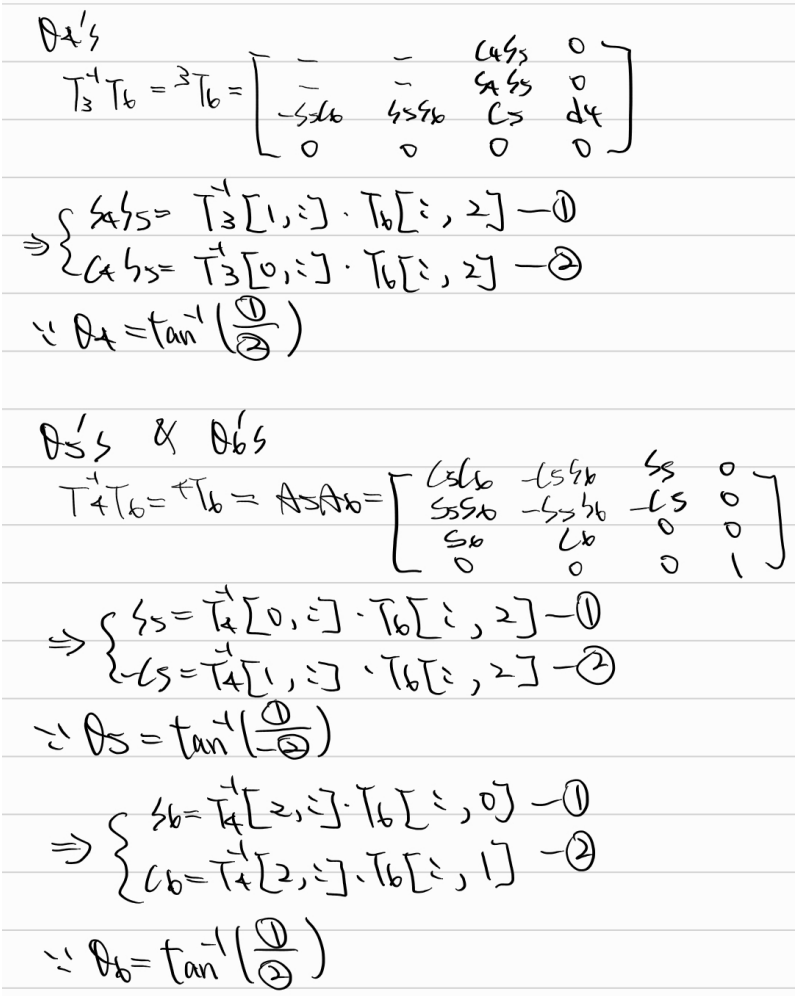
θ_2's

$$T_3^{-1}T_6 = {}^3T_6 = A_4A_5A_6$$

$$\Rightarrow \begin{bmatrix} c_1c_{23} & s_1c_{23} & -s_{23} & -a_3 - a_2c_3 \\ -s_1 & c_1 & 0 & -d_3 \\ c_1s_{23} & s_1s_{23} & c_{23} & -a_2s_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} n & o & a & p \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} - & - & - & 0 \\ - & - & - & 0 \\ - & - & - & d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\Rightarrow \begin{cases} \underbrace{(c_1P_x + s_1P_y)}_{a}c_{23} - P_zs_{23} = a_3 + a_2c_3 & P = \sqrt{a^2 + b^2} \\ \underbrace{(c_1P_x + s_1P_y)}_{b}s_{23} + P_zc_{23} = d_4 + a_2s_3 \,, & \cos\phi = \frac{a}{P} \end{cases} \Rightarrow \begin{cases} \cos(\phi + \theta_2 + \theta_3) = a_3 + a_2c_3 \\ \sin(\phi + \theta_2 + \theta_3) = d_4 + a_2s_3 \end{cases}$$

$$\therefore \theta_2 = \tan^{-1}\left(\frac{d_4 + a_2s_3}{a_3 + a_2c_3}\right) - \theta_3 - \tan^{-1}\left(\frac{P_z}{c_1P_x + s_1P_y}\right) \ast$$



Bonus

Algebraic & Geometric method

The algebraic method utilizes techniques such as matrix inversion and equation solving to determine θ and d. It requires a substantial amount of computation and can handle problems involving more than three axes. The calculations are more complex compared to the geometric method; however, this intricate computational approach is well-suited for rapid processing in computers.

On the other hand, the geometric method relies on spatial concepts and projection techniques. It calculates θ1, θ2, and θ3 by projecting the final position in space. The computational process is comparatively straightforward. However, a drawback is its inability to handle the directional aspects of the endpoint, making it unsuitable for solving six-axis problems.