```python
import numpy as np
import sympy as sym
import math

##### A METTRE ########
from sympy import lambdify
##############################

from scipy.optimize import fsolve


def menu():
    choix = int(input("1: Cinématique directe\n2: Cinématique inverse\nVotre choix: "))
    return choix


def transfo_homogene(theta, d, a, alpha):
    C_A=np.round(np.cos(alpha),4)
    C_T=np.round(np.cos(theta),4)
    S_T=np.round(np.sin(theta),4)
    S_A=np.round(np.sin(alpha),4)

    T = np.array([[C_T, -S_T*C_A, S_T*S_A, a*C_T],
        [S_T, C_T*C_A, -C_T*S_A, a*S_T],
        [0, S_A, C_A, d],
        [0,0,0,1]])

    return T

def donnees() :
    theta1 = np.deg2rad(float (input ("Donnez la valeur de theta 1 en degrés :")))
    theta2 = np.deg2rad(float (input ("Donnez la valeur de theta 2 en degrés :")))
    theta3 = np.deg2rad(float (input ("Donnez la valeur de theta 3 en degrés :")))
    theta4 = np.deg2rad(float (input ("Donnez la valeur de theta 4 en degrés :")))
    theta5 = np.deg2rad(float (input ("Donnez la valeur de theta 5 en degrés :")))
    theta6 = np.deg2rad(float (input ("Donnez la valeur de theta 6 en degrés :")))
    return theta1, theta2, theta3, theta4, theta5, theta6

def matrice_final(matrices):
    A01 = matrices[0]
    A12 = matrices[1]
    A23 = matrices[2]
    A34 = matrices[3]
    A45 = matrices[4]
    A56 = matrices[5]

    A06 = A01@A12@A23@A34@A45@A56

    return A06

def pos_effecteur(matrice_final):
    position_final= matrice_final[:3,3]
    print("le vecteur de position final est:", position_final)

def Euler_angl(matrice_finale):
    Angle_y = np.rad2deg(np.arcsin(-matrice_finale[2,0]))
    Angle_x = np.rad2deg(np.arcsin(matrice_finale[2,1]/np.cos(np.deg2rad(Angle_y))))
    Angle_z = np.rad2deg(np.arcsin(matrice_finale[1,0]/np.cos(np.deg2rad(Angle_y))))
    vecteur=[Angle_x,Angle_y,Angle_z]
    return vecteur


def rotm2eul(rotm):
    eul = [0,0,0]
    if (rotm[2,0] < 1):
        if (rotm[2,0] > -1): # case 1: if r31 ~= ±1
            # Solution with positive sign. It limits the range of the values
            # of theta_y to (-pi/2, pi/2):
            eul[2] = np.atan2(rotm[1,0], rotm[0,0]) # theta_z
            eul[1] = np.asin(-rotm[2,0])            # theta_y
            eul[0] = np.atan2(rotm[2,1], rotm[2,2]) # theta_x
        else: #  case 2: if r31 = -1
            # theta_x and theta_z are linked --> Gimbal lock:
            # There are infinity number of solutions for theta_x - theta_z = atan2(-r23, r22).
            # To find a solution, set theta_x = 0 by convention.
            eul[2] = -np.atan2(-rotm[1,2], rotm[1,1])
            eul[1] = math.pi/2
            eul[0] = 0
    else: # case 3: if r31 = 1
        # Gimbal lock: There is not a unique solution for
        # theta_x + theta_z = atan2(-r23, r22), by convention, set theta_x = 0.
        eul[2] = np.arctan2(-rotm[1,2], rotm[1,1])
        eul[1] = -math.pi/2
        eul[0] = 0
    return eul

def demande_pos():
    x=float(input("Donne moi la coordonée en x (mm) : "))
    y=float(input("Donne moi la coordonée en y (mm) : "))
    z=float(input("Donne moi la coordonée en z (mm) : "))
    position=[x,y,z]
    return position

def demande_ori():
    phi=float(input("Donne moi l'angle phi (autour de x en deg) : "))
    theta=float(input("Donne moi l'angle theta (autour de y en deg) : "))
    psi=float(input("Donne moi l'angle psi (autour de z en deg) : "))
    orientation=[phi,theta,psi]
    return orientation

def ComputeT06(pos,ori):
    Cph=np.cos(np.deg2rad(ori[0]))
    Sph=np.sin(np.deg2rad(ori[0]))
    Cth=np.cos(np.deg2rad(ori[1]))
    Sth=np.sin(np.deg2rad(ori[1]))
    Cps=np.cos(np.deg2rad(ori[2]))
```

```
    Sps=np.sin(np.deg2rad(ori[2]))
    px=pos[0]
    py=pos[1]
    pz=pos[2]
    nx= Cth*Cps
    ox= Sph*Sth*Cps-Cph*Sps
    ax= Cph*Sth*Cps+Sph*Sps
    ny= Cth*Sps
    oy= Sph*Sth*Sps+Cph*Cps
    ay= Cph*Sth*Sps-Sph*Cps
    nz= -Sth
    oz= Cth*Sph
    az= Cth*Cph
    T=np.array([[nx,ox,ax,px],[ny,oy,ay,py],[nz,oz,az,pz],[0,0,0,1]])
    return T

def func(X,AUX1,AUX2,AUX3,Eqx,Eqy,Eqz):
    q1=X[0]
    q2=X[1]
    q3=X[2]
    eq1=float(Eqx)-AUX1
    eq2=float(Eqy)-AUX2
    eq3=float(Eqz)-AUX3
    return [eq1,eq2,eq3]


def fonc(X,AUX1,AUX2,AUX3,Q1,Q2,Q3):
    Q4=X[0]
    Q5=X[1]
    Q6=X[2]
    eq1 = np.cos(Q6)*(np.cos(Q4)*np.sin(Q1) - np.sin(Q4)*np.cos(Q1)*(np.cos(Q2)*np.sin(Q3) + np.cos(Q3)*np.sin(Q2))) - np.sin(Q6)*(np.cos(Q5)*(np.sin(Q1)*np.si
    eq2 = - np.sin(Q5)*(np.sin(Q1)*np.sin(Q4) + np.cos(Q4)*np.cos(Q1)*(np.cos(Q2)*np.sin(Q3) + np.cos(Q3)*np.sin(Q2))) - np.cos(Q5)*np.cos(Q1)*(np.sin(Q2)*np.s
    eq3 = np.sin(Q5)*(np.cos(Q1)*np.sin(Q4) - np.cos(Q4)*(np.cos(Q2)*np.sin(Q1)*np.sin(Q3) + np.cos(Q3)*np.sin(Q1)*np.sin(Q2))) - np.cos(Q5)*(np.sin(Q1)*np.sin
    return [eq1,eq2,eq3]



def Symbolic_TH(q,theta, d, a, alpha):
    C_A=np.round(np.cos(alpha),4)

    C_T=sym.cos(q)*np.round(np.cos(theta),1)-sym.sin(q)*np.round(np.sin(theta),1)
    S_T=sym.sin(q)*np.round(np.cos(theta),1)+sym.cos(q)*np.round(np.sin(theta),1)

    S_A=np.round(np.sin(alpha),1)

    T = np.array([[C_T, -S_T*C_A, S_T*S_A, a*C_T],
        [S_T, C_T*C_A, -C_T*S_A, a*S_T],
        [0, S_A, C_A, d],
        [0,0,0,1]])

    return T


def InverseSymbol(A):
    nx=A[0,0]
    ny=A[1,0]
    nz=A[2,0]

    ox=A[0,1]
    oy=A[1,1]
    oz=A[2,1]

    ax=A[0,2]
    ay=A[1,2]
    az=A[2,2]

    px=A[0,3]
    py=A[1,3]
    pz=A[2,3]
    Tinv=[[nx,ny,nz,-(nx*px+ny*py+nz*pz)],
          [ox,oy,oz,-(ox*px+oy*py+oz*pz)],
          [ax,ay,az,-(ax*px+ay*py+az*pz)],
          [0,0,0,1]]

    return Tinv


##### FONCTION POUR PASSER LES EQS AUX VARIABLES NUMERIQUES
def convert_2_numeric(sym_eqx,sym_eqy,sym_eqz,q1,q2,q3,px,py,pz):
    funx=lambdify((q1,q2,q3),sym_eqx,"numpy")
    funy=lambdify((q1,q2,q3),sym_eqy,"numpy")
    funz=lambdify((q1,q2,q3),sym_eqz,"numpy")
    def numeric_func(X):
        Q1,Q2,Q3=X
        return [funx(Q1,Q2,Q3)-px,funy(Q1,Q2,Q3)-py,funz(Q1,Q2,Q3)-pz]
    return numeric_func

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

def main():
    m=0

    q1=sym.Symbol('q1')
    q2=sym.Symbol('q2')
    q3=sym.Symbol('q3')
    q4=sym.Symbol('q4')
    q5=sym.Symbol('q5')
    q6=sym.Symbol('q6')

    matrice = []
    """d=[0,445,0,0,880,0]
    a=[0,150,700,115,0,0]
    alpha=np.deg2rad([0,-90,0,-90,90,-90])
    theta = np.deg2rad([90,0,-90,0,0,0])
"""
    d=[445,0,0,795,0,85]
```

```
    a=[150,700,115,0,0,0]
    alpha=np.deg2rad([-90,0,-90,90,-90,0])
    theta = np.deg2rad([0,-90,0,0,0,0])


    A01_S=Symbolic_TH(q1,theta[0],d[0],a[0],alpha[0])
    A12_S=Symbolic_TH(q2,theta[1],d[1],a[1],alpha[1])
    A23_S=Symbolic_TH(q3,theta[2],d[2],a[2],alpha[2])
    A34_S=Symbolic_TH(q4,theta[3],d[3],a[3],alpha[3])
    A45_S=Symbolic_TH(q5,theta[4],d[4],a[4],alpha[4])
    A56_S=Symbolic_TH(q6,theta[5],d[5],a[5],alpha[5])

    """print("\nA01:\n", A01_S)
    print("\nA12:\n", A12_S)
    print("\nA23:\n", A23_S)
    print("\nA34:\n", A34_S)
    print("\nA45:\n", A45_S)
    print("\nA56:\n", A56_S)"""

    A04_S=A01_S@A12_S@A23_S@A34_S

    A46_S=A45_S@A56_S
    invA46_S=InverseSymbol(A46_S)
    print("\n\n\n\n\n\n",invA46_S,"\n\n\n\n\n\n")

    m=menu()
    if m == 1:
        THETA = donnees()
        for i in range(6):
            #print(transfo_homogene(THETA[i]+theta[i],d[i],a[i],alpha[i]))
            matrice.append(transfo_homogene(THETA[i]+theta[i],d[i],a[i],alpha[i]))

        print("\nA01:\n", matrice[0])
        print("\nA12:\n", matrice[1])
        print("\nA23:\n", matrice[2])
        print("\nA34:\n", matrice[3])
        print("\nA45:\n", matrice[4])
        print("\nA56:\n", matrice[5])

        print("La matrice final est :", matrice_final(matrice))
        pos_effecteur(matrice_final(matrice))
        print(rotm2eul(matrice_final(matrice)))
    else:
        pos_connue=demande_pos()
        ori_connue=demande_ori()


        T06=ComputeT06(pos_connue,ori_connue)

        POS=T06@invA46_S
        Pos_x=POS[0,3]
        Pos_y=POS[1,3]
        Pos_z=POS[2,3]

        Aux1=sym.simplify(Pos_x)
        Aux2=sym.simplify(Pos_y)
        Aux3=sym.simplify(Pos_z)

        """print(Aux1)
        print(Aux2)
        print(Aux3)"""

        Comparaison_x=sym.trigsimp(A04_S[0,3])
        Comparaison_y=sym.trigsimp(A04_S[1,3])
        Comparaison_z=sym.trigsimp(A04_S[2,3])

        print("\n\n\neq1: ",Comparaison_x, "-",Aux1)
        print("\neq2: ",Comparaison_y, "-",Aux2)
        print("\neq3: ",Comparaison_z, "-",Aux3)

        ########### CINEMATIQUE INVERSE RESOLUE ##############

        NFunc=convert_2_numeric(Comparaison_x,Comparaison_y,Comparaison_z,q1,q2,q3,Aux1,Aux2,Aux3)
        #print(NFunc)
        sol=fsolve(NFunc,[0,0,0],maxfev=10000)
        q1_n,q2_n,q3_n=sol

        """print(np.rad2deg(q1_n))
        print(np.rad2deg(q2_n))
        print(np.rad2deg(q3_n))"""

        OX=T06[0,1]
        AX=T06[0,2]
        AY=T06[1,1]

        T06_S=transfo_homogene(q1_n+theta[0],d[0],a[0],alpha[0])@transfo_homogene(q2_n+theta[1],d[1],a[1],alpha[1])@transfo_homogene(q3_n+theta[2],d[2],a[2],al

        OX_s=T06_S[0,1]
        AX_s=T06_S[0,2]
        AY_s=T06_S[1,1]

        print("\n\n\neq4: ",OX_s, "-",OX)
        print("\neq5: ",AX_s, "-",AX)
        print("\neq6: ",AY_s, "-",AY)

        NFunc2=convert_2_numeric(OX_s,AX_s,AY_s,q4,q5,q6,OX,AX,AY)
        sol2=fsolve(NFunc2,[0,0,0],maxfev=10000)

        q4_n,q5_n,q6_n=sol2
        print(np.rad2deg(q4_n))
        print(np.rad2deg(q5_n))
        print(np.rad2deg(q6_n))

        ########################################################
main()
```