france lawrence adalin <francelawrence.adalin@g.batstate-u.edu.ph>

CALCULATOR

1 message

raphael angelo valladolid <raphaelangelo.valladolid@g.batstate-u.edu.ph> Mon, Jun 6, 2022 at 7:34 AM
To: france lawrence adalin <francelawrence.adalin@g.batstate-u.edu.ph>

```
import numpy as np
import math
import PySimpleGUI as sg
import pandas as pd

# GUI code
sg. theme('DarkBlue')

# Excel read code

EXCEL_FILE = 'Cartesian_Design_Data_FK.xlsx'
df = pd.read_excel(EXCEL_FILE)

# Lay-out code

Main_layout = [
  [sg.Push(), sg.Text('Cartesian MEXE Calculator', font = ("Consolas", 25)), sg.Push()],

  [sg.Text('Fill out the following fields:', font = ("Consolas",15)),
    sg.Push(), sg.Button('Solve Forward Kinematics',
    font = ("Consolas",12), size=(35,0), button_color=('black','yellow')),sg.Push(),sg.Text('OR',font =
("Consolas",12)),sg.Push(),
    sg.Push(),sg.Button('Inverse Kinematics',font = ("Consolas",12),
    size=(35,0), button_color=('white','green'))],

  [sg.Text('a1 = ', font = ("Consolas", 10)),sg.InputText('', key = 'a1', size =(20,10)),
    sg.Text('d1 = ',font = ("Consolas", 10)),sg.InputText('',key='d1', size=(20,10)),
    sg.Push(), sg.Button('Jacobian Matrix (J)', font = ("Consolas", 12),size=(20,0),button_color=('white','green')),
    sg.Button('Det(J)',font = ("Consolas", 12), size=(15,0), button_color=('white','orange')),
    sg.Button('Inverse of J',font = ("Consolas", 12), size=(15,0), button_color=('white','gray')),
    sg.Button('Transpose of J', font = ("Consolas", 12), size=(15,0), button_color=('white','blue')), sg.Push()],
  [sg.Text('a2 = ', font = ("Consolas", 10)),sg.InputText('',key='a2', size=(20,10)),
    sg.Text('d2 = ',font = ("Consolas", 10)),
    sg.InputText('',key='d2',size=(20,10)),
    sg.Push(),sg.Button('Path and Trajectory Planning', font = ("Consolas",12), size=(40,0), button_color=
('white','black')), sg.Push()],

  [sg.Text('a3 = ', font = ("Consolas", 10)),sg.InputText('',key='a3', size=(20,10)),
    sg.Text('d3 = ',font = ("Consolas", 10)),
    sg.InputText('',key='d3',size=(20,10))],
```

```
    [sg.Text('a4 = ', font = ("Consolas",10)),sg.InputText('',key='a4', size=(20,10))],


    [sg.Button('Click this before Solving Foward Kinematics',tooltip = 'Solve Forward Kinematics !!!', font =
("Consolas",12), button_color=('white','purple')), sg.Push(),
    sg.Push(),sg.Frame('Position Vector: ',[[
      sg.Text('X= ', font = ("Consolas",12)),sg.InputText(key ='X', size=(20,0)),
      sg.Text('Y= ', font = ("Consolas",12)),sg.InputText(key ='Y', size=(20,0)),
      sg.Text('Z= ', font = ("Consolas",12)),sg.InputText(key ='Z', size=(20,0))]]),sg.Push()],

    [sg.Push(), sg.Frame('H0_3 Transformation Matrix = ',[[sg.Output(size=(80,15))]]),
      sg.Push(),sg.Image('Cartesian Manipulator.gif'),sg.Push()],
    [sg.Submit(font = ("Consolas",10)),sg.Exit(font = ("Consolas",10))]

   ]

# Windows Code
window = sg.Window('Cartesian MEXE Calculator', Main_layout, resizable = True)

# Inverse Kinematics Window Function
def Inverse_Kinematics_window():
    sg.theme('DarkBlue')
    EXCEL_FILE = 'Cartesian_Design_Data_Inverse_Kinematics.xlsx'
    IK_df = pd.read_excel(EXCEL_FILE)

    IK_Layout = [
        [sg.Push(),sg.Text('Inverse Kinematics', font = ("Consolas",20)),sg.Push()],
        [sg.Text('Fill out the followingg fields:', font =("Consolas",10))],
        [sg.Text('a1= ',font = ("Consolas",10)), sg.InputText('',key= 'a1',size=(8,10)),
          sg.Text('mm',font =("Consolas",10)),
          sg.Text('X = ',font =("Consolas",10)),sg.InputText('',key='X',size=(8,10)),
          sg.Text('mm',font =("Consolas",10))],

        [sg.Text('a2= ',font = ("Consolas",10)), sg.InputText('',key= 'a2',size=(8,10)),
         sg.Text('mm',font =("Consolas",10)),
           sg.Text('Y = ',font =("Consolas",10)),sg.InputText('',key='Y',size=(8,10)),
           sg.Text('mm',font =("Consolas",10))],
        [sg.Text('a3= ',font = ("Consolas",10)), sg.InputText('',key= 'a3',size=(8,10)),
         sg.Text('mm',font =("Consolas",10)),
           sg.Text('Z = ',font =("Consolas",10)),sg.InputText('',key='Z',size=(8,10)),
           sg.Text('mm',font =("Consolas",10))],
        [sg.Text('a4= ',font = ("Consolas",10)), sg.InputText('',key= 'a4',size=(8,10)),
         sg.Text('mm',font =("Consolas",10))],
        [sg.Button('Inverse Kinematics',font =("Consolas",12), button_color = ('yellow','black')),sg.Push()],

        [sg.Frame ('Position Vector: ',[[
           sg.Text('d1 =',font = ("Consolas",10)),sg.InputText( key='IK_d1',size =(10,1)),
           sg.Text('mm',font = ("Consolas",10)),
           sg.Text('d2 =',font = ("Consolas",10)),sg.InputText( key='IK_d2',size =(10,1)),
           sg.Text('mm',font = ("Consolas",10)),
```

```
            sg.Text('d3 =',font = ("Consolas",10)),sg.InputText( key='IK_d3',size =(10,1)),
            sg.Text('mm',font = ("Consolas",10)),]])],
        [sg.Submit(font = ("Consolas",10)),sg.Exit(font = ("Consolas",10))]

    ]

# Windows Code for Inverse Kinematics
    Inverse_Kinematics_window = sg.Window('Inverse Kinematics', IK_Layout)

    while True:
        event,values =  Inverse_Kinematics_window.read()
        if event == sg.WIN_CLOSED or event == 'Exit':
            break

        elif event == 'Inverse Kinematics':

            #linklengths
            a1 = float(values['a1'])
            a2 = float(values['a2'])
            a3 = float(values['a3'])
            a4 = float(values['a4'])


            #position vecotors
            X = float(values['X'])
            Y = float(values['Y'])
            Z = float(values['Z'])

            # X = 110.00000000000004
            # Y = 119.99999999999993s
            # Z = 0.0
            #d1
            d1 = Y - a2

            #d2
            d2 = X -a3
            #d3
            d3 = a1 - a4 - Z

            #print("d1= ",np.around(d1,3))
            #print("d2= ",np.around(d2,3))
            #print("d3= ",np.around(d3,3))
            d1 = Inverse_Kinematics_window['IK_d1'].Update(np.around(d1,3))
            d2 = Inverse_Kinematics_window['IK_d2'].Update(np.around(d2,3))
            d3 = Inverse_Kinematics_window['IK_d3'].Update(np.around(d3,3))
        elif event =='Submit':
            IK_df = IK_df.append(values, ignore_index=True)
            IK_df.to_excel(EXCEL_FILE, index =False)
            sg.popup('Data Saved!')

    Inverse_Kinematics_window.close()
```

```
def clear_input():
    for key in values:
        window[key]('')
    return None

#Variable Codes for disabling buttons
disable_FK = window['Solve Forward Kinematics']
disable_J = window['Jacobian Matrix (J)']
disable_DetJ = window['Det(J)']
disable_IV = window ['Inverse of J']
disable_TJ = window ['Transpose of J']
disable_PT = window ['Path and Trajectory Planning']

while True:
    event,values = window.read()
    if event == sg.WIN_CLOSED or event == 'Exit':
        break

    if event ==('Click this before Solving Forward Kinematics'):
        disable_J.update(disabled=True)
        disable_DetJ.update(disabled=True)
        disable_IV.update(disabled=True)
        disable_TJ.update(disabled=True)
        disable_PT.update(disabled=True)


    if event == 'Solve Forward Kinematics':

        # Forward Kinematic Codes
        # link lengths in cm
        a1 = values['a1'] # For Testing, 150cm
        a2 = values['a2'] # For Testing, 80cm
        a3 = values['a3'] # For Testing, 80cm
        a4 = values['a4'] # For Testing, 80cm

        # Joint Variable Thetas in degrees
        d1 = values['d1'] # For Testing, 40cm
        d2 = values['d2'] # For Testing, 30cm
        d3 = values['d3'] # For Testing, 70cm

        # If Joint Variable are ds don't need to convert

        ## D-H Parameter Table (This is the only part you only edit for every new mechanical manipulator.)
        # Rows = no. of HTM, Colums = no. of Parameters
        # Theta, alpha, r, d

        DHPT = [[0,(270.0/180.0)*np.pi,0,float(a1)],
                [(270.0/180.0)*np.pi,(270.0/180.0)*np.pi,0,float(a2)+float(d1)],
                [(270.0/180.0)*np.pi,(90.0/180.0)*np.pi,0,float(a3)+float(d2)],
```

```
          [0,0,0,float(a4)+float(d3)]
          ]

    # np.trigo function (DHPT[row][column])

    i = 0
    H0_1 = [[np.cos(DHPT[i][0]),-np.sin(DHPT[i][0])*np.cos(DHPT[i][1]),np.sin(DHPT[i][0])*np.sin(DHPT[i]
[1]),DHPT[i][2]*np.cos(DHPT[i][0])],
           [np.sin(DHPT[i][0]),np.cos(DHPT[i][0])*np.cos(DHPT[i][1]),-np.cos(DHPT[i][0])*np.sin(DHPT[i]
[1]),DHPT[i][2]*np.sin(DHPT[i][0])],
           [0,np.sin(DHPT[i][1]),np.cos(DHPT[i][1]),DHPT[i][3]],
           [0,0,0,1]]

    i = 1
    H1_2 = [[np.cos(DHPT[i][0]),-np.sin(DHPT[i][0])*np.cos(DHPT[i][1]),np.sin(DHPT[i][0])*np.sin(DHPT[i]
[1]),DHPT[i][2]*np.cos(DHPT[i][0])],
           [np.sin(DHPT[i][0]),np.cos(DHPT[i][0])*np.cos(DHPT[i][1]),-np.cos(DHPT[i][0])*np.sin(DHPT[i]
[1]),DHPT[i][2]*np.sin(DHPT[i][0])],
           [0,np.sin(DHPT[i][1]),np.cos(DHPT[i][1]),DHPT[i][3]],
           [0,0,0,1]]

    i = 2
    H2_3 = [[np.cos(DHPT[i][0]),-np.sin(DHPT[i][0])*np.cos(DHPT[i][1]),np.sin(DHPT[i][0])*np.sin(DHPT[i]
[1]),DHPT[i][2]*np.cos(DHPT[i][0])],
           [np.sin(DHPT[i][0]),np.cos(DHPT[i][0])*np.cos(DHPT[i][1]),-np.cos(DHPT[i][0])*np.sin(DHPT[i]
[1]),DHPT[i][2]*np.sin(DHPT[i][0])],
           [0,np.sin(DHPT[i][1]),np.cos(DHPT[i][1]),DHPT[i][3]],
           [0,0,0,1]]

    i = 3
    H3_4 = [[np.cos(DHPT[i][0]),-np.sin(DHPT[i][0])*np.cos(DHPT[i][1]),np.sin(DHPT[i][0])*np.sin(DHPT[i]
[1]),DHPT[i][2]*np.cos(DHPT[i][0])],
           [np.sin(DHPT[i][0]),np.cos(DHPT[i][0])*np.cos(DHPT[i][1]),-np.cos(DHPT[i][0])*np.sin(DHPT[i]
[1]),DHPT[i][2]*np.sin(DHPT[i][0])],
           [0,np.sin(DHPT[i][1]),np.cos(DHPT[i][1]),DHPT[i][3]],
           [0,0,0,1]]

    # Transportation Matrices from base to end-effector
    #print("H0_1 = ")
    #print(np.matrix(H0_1))
    #print("H1_2 = ")
    #print(np.matrix(H1_2))
    #print("H2_3 = ")
    #print(np.matrix(H2_3))

    # Dot Product of H0_3 = H0_1*H1_2*H2_3
    H0_2 = np.dot(H0_1,H1_2)
    H0_3 = np.dot(H0_2,H2_3)
    H0_4 = np.dot(H0_3,H3_4)

    # Transportation Matrix of the Manipulator
```

```
        print("H0_3 = ")
        print(np.matrix(H0_3))

        # Position Vector X Y Z
        X0_4 = H0_4[0,3]
        print("X = ", X0_4)
        Y0_4 = H0_4[1,3]
        print("Y = ", Y0_4)
        Z0_4 = H0_4[2,3]
        print("Z = ", Z0_4)

        disable_J.update(disabled=False)
        disable_PT.update(disabled=False)


    if event == 'Submit' :
        df = df.append(values, ignore_index=True)
        df.to_excel(EXCEL_FILE, index=False)
        sg.popup('Data Saved!')

    if event == 'Jacobian Matrix (J)' :

        # Defining the equations

        i = [[0],[0],[1]]
        A = [[0],[0],[0]]
        IM = [[1,0,0],[0,1,0],[0,0,1]]

     #  try:
     #  H0_1 = np.matrix(H0_1)
     #  except:
     #  H0_1 = -1
     #  sg.popup('WARNING')
     #  sg.popup('Restart the GUI, then click first the "Click Here to Start Calculation" button!')
     #  break


        # Row 1 - 3, column 1
        #H0_0 = np.dot(H0_1,H0_1)
        #R0_0 = H0_0[0:3, 0:3]
        #R0_0 = i
        J0 = np.dot(IM,i)

        # Row 1-3, column 2
        H0_1a = np.dot(H0_1,1)
        R0_1 = H0_1a[0:3,0:3]
        J1 = np.dot(R0_1,i)

        # Row 1-3, column 3
        R0_2 = H0_2[0:3, 0:3]
        J2 = np.dot(R0_2,i)
```

```
        # Row 1-3, column 4
        R0_3 = H0_3[0:3, 0:3]
        J3 = np.dot(R0_3,i)



        # Jacobian Matrix
        JM1 = np.concatenate((J0, J1, J2, J3), 1)
        JM2 = np.concatenate((A, A, A, A), 1)
        Jacobian = np.concatenate((JM1, JM2), 0)
        sg.popup('J =', Jacobian)
        JM1a = np.concatenate((J0, J1, J2), 1)
        #Jacobi_a = Jacobian[4:0, 4:]
        #Jacobi_b = np.dot(Jacobi_a,1)
        # Disabler program
        disable_J.update(disabled=True)
        disable_DetJ.update(disabled=False)
        disable_TJ.update(disabled=False)

    if event == 'Det(J)':
        # Singularity = Det(J)
        # np.linalg.    det(M)
        # Let JM1 become the 3x3 position matrix for obtaining  the Determinant
        try:
            JM1 = np.concatenate((J1,J2,J3),1)
        except:
            JM1 = -1 #NAN
            sg.popup('Warning!')
            sg.popup('Restart the GUI then, go first "Click before Solving Forward Kinematics!!!"')
            break

        DJ = np.linalg.det(JM1)
        #print("DJ = ",DJ)
        sg.popup('DJ = ', DJ)

        if DJ == 0.0 or DJ == -0:
            disable_IV.update(disabled=True)
            sg.popup('Warning:Jacobian Matrix is Non-Invertible!')

    if event == 'Inverse of J':
        # Inv(J)
        try:
            JM1 = np.concatenate((J1,J2,J3),1)
        except:
            JM1 = -1 #NAN
            sg.popup('Warning!')
            sg.popup('Restart the GUI then, go first "Click before Solving Forward Kinematics!!!"')
            break
```

```
        IJ = np.linalg.inv (JM1)
        #print("IV =")
        #print(IV)
        sg.popup('IJ = ',IJ)

    if event == 'Transpose of J':
        #Transpose of Jacobian Matrix
        try:
            JM1 = np.concatenate((J1,J2,J3),1)
        except:
            JM1 = -1 #NAN
            sg.popup('Warning!')
            sg.popup('Restart the GUI then, go first "Click before Solving Forward Kinematics!!!"')
            break


        TJ = np.transpose(JM1)
        #print("TJ= ",TJ)
        sg.popup('TJ = ',TJ)

    elif event == 'Inverse Kinematics':
        Inverse_Kinematics_window()


window.close()
```