UNIVERSIDADE FEDERAL DE ALAGOAS

Robótica – AB1 – Parte 2 - Respostas

Aluno: José Renilson Almeida da Silva

Código disponível em: https://github.com/RenilsonA/Robotica/blob/main/ab1-parte2.py

1º)

Para esta questão, deve-se levar em consideração o seguinte algoritmo:

```
print("############Questão 1############")

L1 = L2 = 1
p = [0.5, 0.5]
bp = ET2.R() * ET2.tx(L1) * ET2.R() * ET2.tx(L2)
aux = p[0]**2 + p[1]**2 - L1**2 - L2**2
div = 2 * L1 * L2
B = math.acos(aux / div)
aux1 = math.atan2(p[1], p[0])
aux2 = math.atan2(L2 * math.sin(B), L1 + L2 * math.cos(B))
A = aux1 - aux2

fk = bp.fkine([A, B])
print("fkine =")
print(fk)
bp.teach([A, B])
```

Que gera a seguinte saída:

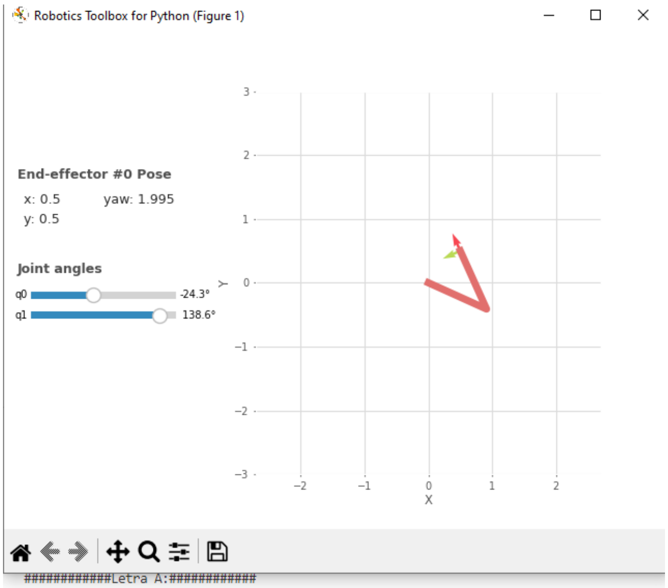
```
############Questão 1############
fkine =
  -0.4114   -0.9114    0.5
   0.9114   -0.4114    0.5
   0         0          1
```

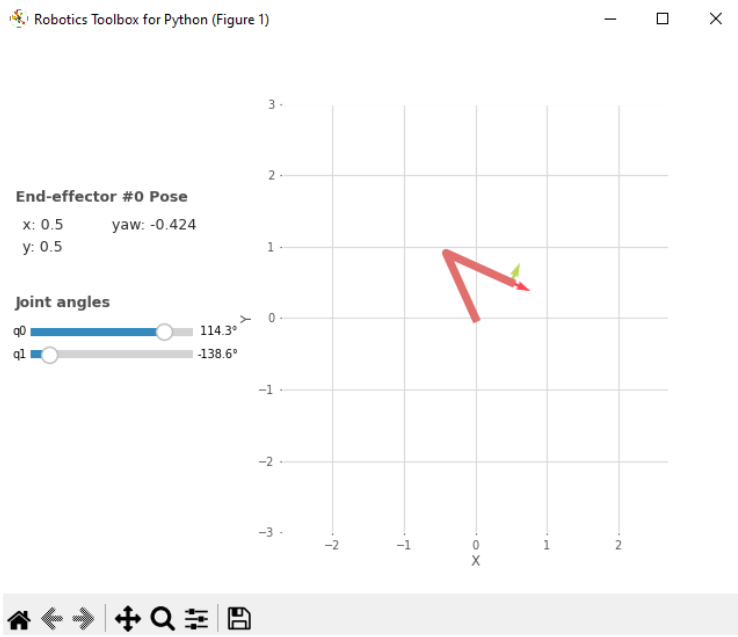
a) Considerando os conjuntos de ângulos fornecidos na equação, e o algoritmo acima, onde temos o posicionamento do efetuador e a matriz de transformação, podemos fazer:

```
############Letra A:############

print("############Letra A:############")
qp = [-0.424, 2.4188]
qn = [1.9948, -2.4188]
fkp = bp.fkine(qp)
fkn = bp.fkine(qn)
print("A:")
print("fkine para o conjunto q+:")
print(fkp)
bp.teach(qp)
```

```
print("Pose q+:")
print(fkp.printline())
print("fkine para o conjunto q-:")
print(fkn)
bp.teach(qn)
print("Pose q-:")
print(fkn.printline())
```

Que gera a seguinte saída:






```
############Letra A:############
A:
fkine para o conjunto q+:
  -0.4114   -0.9114    0.5
   0.9114   -0.4114    0.5
   0         0         1
```

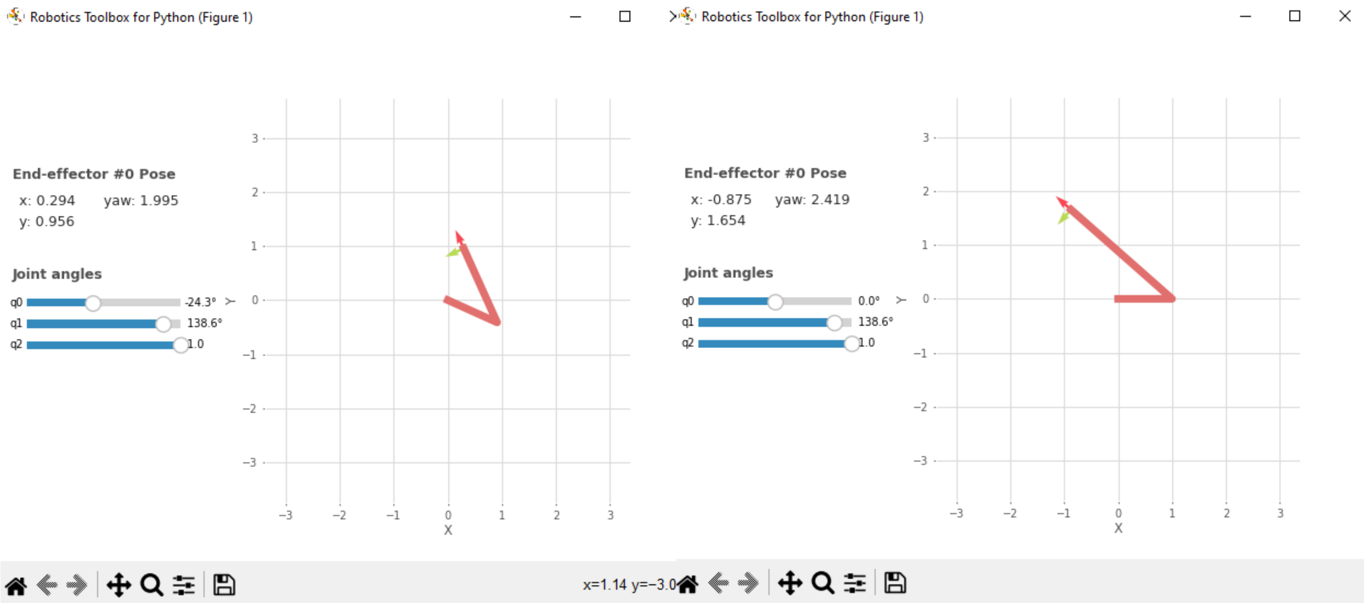

```
############Letra A:############
A:
fkine para o conjunto q+:
  -0.4114   -0.9114    0.5
   0.9114   -0.4114    0.5
   0         0         1

Pose q+:
t = 0.5, 0.5; 114°
None
fkine para o conjunto q-:
   0.9115    0.4114    0.5
  -0.4114    0.9115    0.5
   0         0         1

Pose q-:
t = 0.5, 0.5; -24.3°
None
```

Temos que ambas configurações nos leva para um ponto específico, devido que a função fkine permite que dois ou mais conjuntos nos leve a um ponto específico alcançável pela peça robótica.

b) Considerando o código abaixo, e colocando a junta prismática:

```
print("############Letra B:############")

a = 1
b = 0.5
bpb = ET2.R() * ET2.tx(a) * ET2.R() * ET2.tx(b) * ET2.tx(qlim=[0, 1])
bpb.teach([A, B, 1])
bpb.teach([0, B, 2])
```

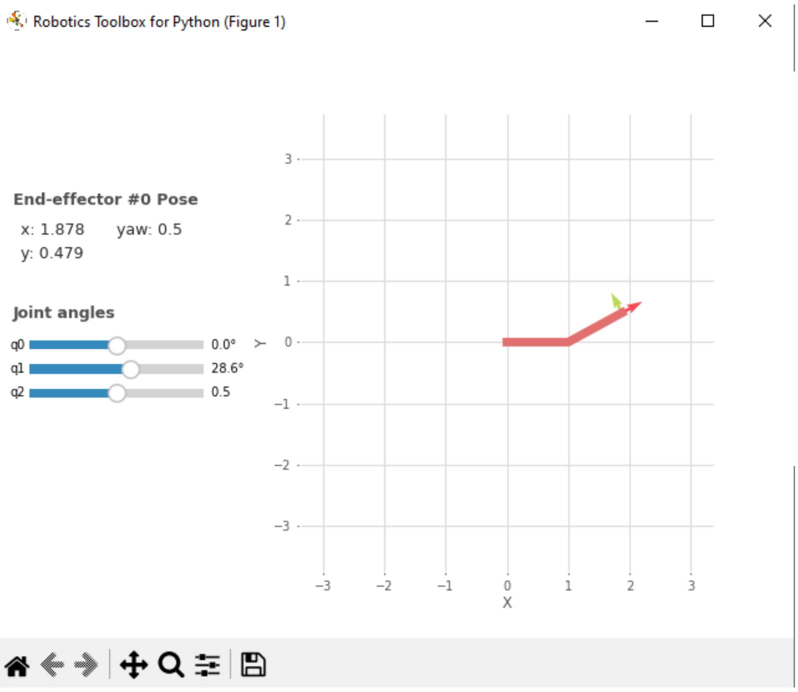
O que nos gera a seguinte saída:



c) Considerando o código abaixo:

```
print("############Letra C:############")

q = [0, 0.5, 0.5]
fkc = bpb.fkine(q)
print("fkine =")
print(fkc)
print("Pose q:")
print(fkc.printline())
bpb.teach(q)
```

O que nos gera a seguinte saída, com a matriz de transformação e a pose e também a simulação 2D do braço:




```
#############Letra C:#############
fkine =
   0.8776   -0.4794    1.878
   0.4794    0.8776    0.4794
   0         0         1

Pose q:
t = 1.88, 0.479; 28.6°
None
```

2º) Na segunda questão, temos que considerar as variáveis abaixo:

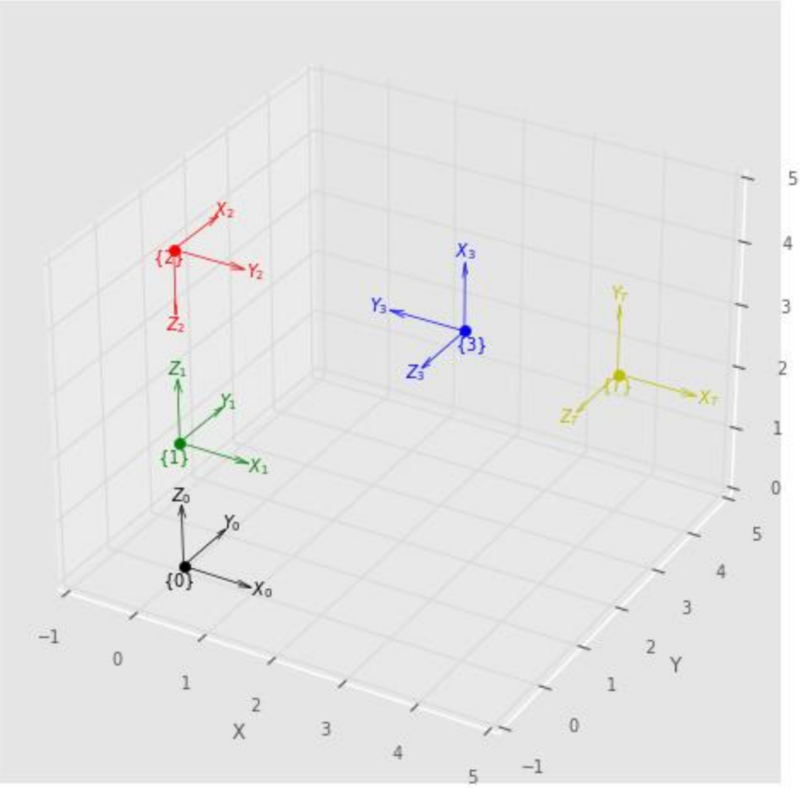
```
print("############Questão 2############")

L1 = 2
L2 = 3
L3 = 4
L4 = 2
```

a) Atribuindo as referências das posições das juntas, temos então o trecho de código abaixo:

```
print("############Letra A:############")

fig = plt.figure()
ax = fig.add_subplot(111,projection='3d')
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

ax.set_xlim([-1, 5])
ax.set_ylim([-1, 5])
ax.set_zlim([0, 5])

E0 = transl(0, 0, 0)
E1 = transl(0, 0, L1)
E2 = transl(0, 0, L1 + L2) @ trotx(np.pi / 2) @ troty(np.pi / 2) @
trotx(np.pi / 2)
E3 = transl(L3, 0, L1 + L2) @ trotx(np.pi / 2) @ trotz(np.pi / 2)
T0 = transl(L3 + L4, 0, L1 + L2) @ trotx(np.pi / 2)
```

```
trplot(E0, frame="0", color="k")
trplot(E1, frame="1", color="g")
trplot(E2, frame="2", color="r")
trplot(E3, frame="3", color="b")
trplot(T0, frame="T", color="y")
plt.show()
```

Que gera o seguinte gráfico:



b) Considerando o trecho do algoritmo abaixo, onde implementamos as revoluções de Denavit-Hartenberg e desenvolvemos então a tabela do robô:

Temos então a seguinte saída:

```
print("############Letra B:############")


d1 = RevoluteDH(d = L1 + L2, alpha = np.pi / 2, name = '1')
d2 = RevoluteDH(a = L3)
d3 = RevoluteDH(a = L4)

DHR = DHRobot([d1, d2, d3], name = 'RRR')
print(DHR)
```

```
DHRobot: RRR, 3 joints (RRR), dynamics, standard DH parameters
```

θ_j	d_j	a_j	α_j
q1	5	0	90.0°
q2	0	4	0.0°
q3	0	2	0.0°

c) Considerando os cálculos:

$$T_3 = {}^0_1T * {}^1_2T * {}^2_3T =$$

$$\begin{bmatrix}1&0&0&0\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} * \begin{bmatrix}1&0&0&0\\0&0&-1&0\\0&1&0&5\\0&0&0&1\end{bmatrix} * \begin{bmatrix}1&0&0&2\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} * \begin{bmatrix}1&0&0&4\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} =$$

$$\begin{bmatrix}1&0&0&0\\0&0&-1&0\\0&1&0&5\\0&0&0&1\end{bmatrix} * \begin{bmatrix}1&0&0&6\\0&1&0&0\\0&0&1&0\\0&0&0&1\end{bmatrix} = \begin{bmatrix}1&0&0&6\\0&0&-1&0\\0&1&0&5\\0&0&0&1\end{bmatrix}$$

E rodando o algoritmo abaixo:

```
print("############Letra C:############")

dhfk = DHR.fkine_all(q=[0, 0, 0])
print(dhfk)
DHR.teach(q = [0, 0, 0])
DHR.teach(q = [0, -np.pi/4, np.pi/2])
```

```
############Letra C:############
[0] =
   1         0         0         0
   0         1         0         0
   0         0         1         0
   0         0         0         1
[1] =
   1         0         0         0
   0         0        -1         0
   0         1         0         5
   0         0         0         1
[2] =
   1         0         0         4
   0         0        -1         0
   0         1         0         5
   0         0         0         1
[3] =
   1         0         0         6
   0         0        -1         0
   0         1         0         5
   0         0         0         1
```

Com base no algoritmo, e com os cálculos realizados, temos que os dados batem perfeitamente com o da biblioteca.

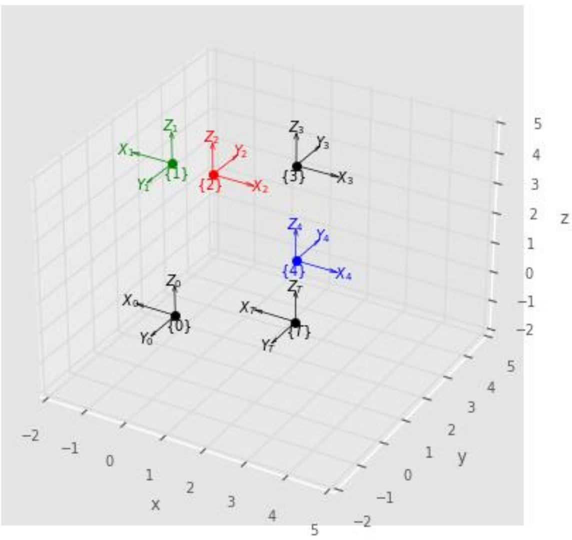
3º) Considerando o algoritmo abaixo, temos então a simulação do robô SCARA, desde seu modelo, manipulador, e a comparação com um exemplo:

```
def funcao(vector, L1, L2, D1, D4):
        P1 = np.matrix([[math.cos(vector[0]), -math.sin(vector[0]),  0, L1 * math.cos(vector[0])],
                    [math.sin(vector[0]),  math.cos(vector[0]),  0, L1 * math.sin(vector[0])],
                    [0,                    0,                    1, D1],
                    [0,                    0,                    0, 1]])

        P2 = np.matrix([[math.cos(vector[1]),  math.sin(vector[1]),  0, L2 * math.cos(vector[1])],
                        [math.sin(vector[1]), -math.cos(vector[1]),  0, L2 * math.sin(vector[1])],
                        [0,                    0,                   -1.0, 0],
                        [0,                    0,                    0, 1]])

        P3 = np.matrix([[1,  0,  0, 0],
                        [0,  1,  0, 0],
                        [0,  0,  1, vector[2]],
                        [0,  0,  0, 1]])

        P4 = np.matrix([[math.cos(vector[3]), -math.sin(vector[3]),  0, 0],
                        [math.sin(vector[3]),  math.cos(vector[3]),  0, 0],
                        [0,                    0,                    1, D4],
                        [0,                    0,                    0, 1]])

        return np.around(np.dot(np.dot(np.dot(P1, P2), P3) ,P4),2)

    print("############Questão 3############")

    vector = [1, 2, 1, 0.3]
    L0 = 5
    L1 = 1
    L2 = 2
    D1 = 1
    D3 = 3
    D4 = 2

    fig = plt.figure()
    ax = fig.add_subplot(111,projection='3d')

    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')

    ax.set_xlim([-2, 5])
    ax.set_ylim([-2, 5])
    ax.set_zlim([-2, 5])
```

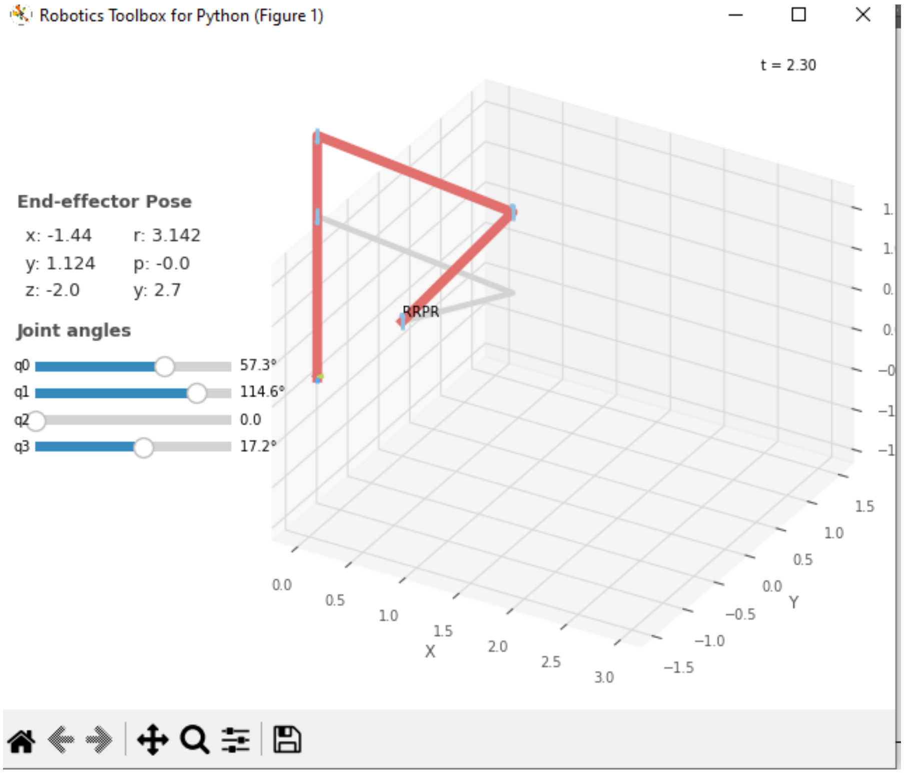
```
E0 = transl(0, 0, 0) @ trotz(np.pi)
E1 = transl(0, 0, L0) @ trotz(np.pi)
E2 = transl(L1, 0, L0)
E3 = transl(L1 + L2, 0, L0 + D1)
E4 = transl(L1 + L2, 0, L0 + D1 - D3)
T0 = transl(L1 + L2, 0, L0 + D1 - D3 - D4) @ trotz(np.pi)

trplot(E0, frame="0", color="k")
trplot(E1, frame="1", color="g")
trplot(E2, frame="2", color="r")
trplot(E3, frame="3", color="k")
trplot(E4, frame="4", color="b")
trplot(T0, frame="T", color="k")

plt.show()

RDH1 = RevoluteDH(a = L1, d = D1)
RDH2 = RevoluteDH(a = L2, alpha = np.pi)
RDH3 = PrismaticDH(qlim = [0, D3])
RDH4 =  RevoluteDH(d = D4)
DHR = DHRobot([RDH1, RDH2, RDH3, RDH4], name = 'RRPR')
print(DHR)

T = funcao(vector, L1, L2, D1, D4)
print("T:")
print(T)

print("DHR Fkine")
print(DHR.fkine(vector))

DHR.teach(vector)
```

Rodando o exemplo, temos o seguinte:

Modelo:



Considerando um exemplo com os valores $L_1 = 1, L_2 = 2, D_1 = 1, D_4 = 2$ e $vetor = [1, 2{,}1, 0.3]$.




```
############Questão 3############
DHRobot: RRPR, 4 joints (RRPR), dynamics, standard
```

θ_j	d_j	a_j	α_j	q^-	q^+
q1	1	1	0.0°	-180.0°	180.0°
q2	0	2	180.0°	-180.0°	180.0°
0.0°	q3	0	0.0°	0.0	3.0
q4	2	0	0.0°	-180.0°	180.0°

Considerando a função implementada, temos que o valor de T é bem próximo de DHR Fkine.

```
T:
[[-0.9   0.43  0.   -1.44]
 [ 0.43  0.9   0.    1.12]
 [ 0.    0.   -1.   -2.  ]
 [ 0.    0.    0.    1.  ]]
DHR Fkine
  -0.9041    0.4274     0         -1.44
   0.4274    0.9041     0          1.124
   0          0        -1         -2
   0          0         0          1
```