Robótica e Automação

Projeto 1

Erica da Cunha Ferreira

Fevereiro 2022

UFRJ

Sumário

1 Introdução

Neste relatório será descrita as etapas para a modelagem cinemática do manipulador *Kinova Gen3 Ultra lightweight* de 7 juntas, mostrado na Figura 1. Para isso, foi utilizado como referência para a posição zero a configuração presente na Figura 2. Como é possível ver na representação em 2D do modelo, há desvios horizontais, estes, entretanto, serão desconsiderados, a fim de facilitar os cálculos.

Figura 1: Manipulador Kinova Gen3 Ultra lightweight

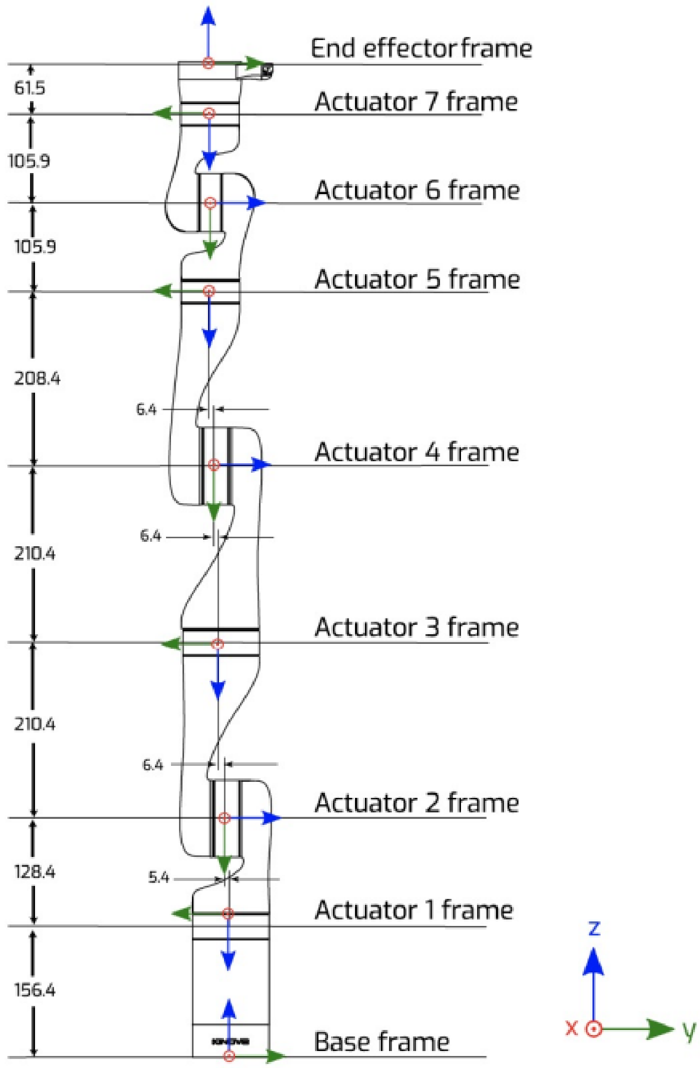

Figura 2: Representação em 2D do modelo Kinova Gen3

2 Metodologia

Para que seja feita a modelagem do manipulador, será calculada a cinemática direta através dos parâmetros de *Denavit-Hartenberg*. Também será feita a cinemática inversa a partir da matriz de transformação homogênea. Além disso, será calculada velocidades, forças e posição do manipulador.

2.1 Denavit-Hartenberg

Denavit-Hartenberg é uma convenção normalmente usada representar as equações cinemáticas de um manipulador. Esta convenção está presente em duas formas **Standard** e **Modificado**, ambos utilizam 4 parâmetros principais para a descrição da cinemática. Neste relatório será utilizado o *Denavit-Hartenberg* ***Standard***.

Antes de aplicar o método, se faz necessário estabelecer algumas regras, como:

- Cada junta conecta 2 elos*
- Cada elo conecta 2 juntas*
- * Exceto o primeiro e último elo

Então temos:

- N juntas
- N+1 elos

A primeira etapa deste método consta em definir eixos de origem para cada junta, a definição segue as regras abaixo: Para isso, existe uma metodologia para a representação de coordenadas de cada eixo.

2.1.1 Eixo de Coordenadas

Os parâmetros de *Denavit-Hartenberg* fornecem a posição relativa de uma junta a outra junta. Para isso, é colocado um eixo de coordenadas no final de cada elo.

Para a localização da origem do eixo de coordenadas, temos:

- Definir $\vec{z_i}$ no eixo de rotação/translação na junta i + 1

  Se $\vec{z_i}$ e $\vec{z_{i-1}}$ se interceptam, O_i fica na interseção.

  Se $\vec{z_i}$ e $\vec{z_{i-1}}$ são paralelas, O_i fica na junta O_{i+1}

- Definir $\vec{x_i}$ ao longo da normal comum a $\vec{z_i}$ e $\vec{z_{i-1}}$.

  ]Se $\vec{z_i}$ e $\vec{z_{i-1}}$ se interceptam, escolher direção* normal ao plano definido por $\vec{z_i}$ e $\vec{z_{i-1}}$. O sentido é arbitrário.

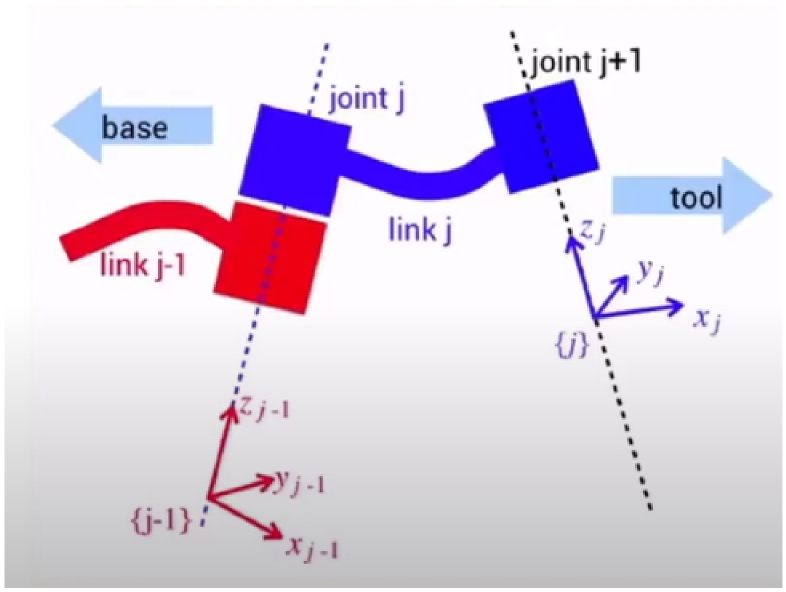

Figura 3: Representação dos eixos de coordenadas utilizando *Denavit-Hartenberg*

Para melhor visualização, podemos ver na Figura 3 os eixos, o Eixo$_j$ é o referencial e o Eixo$_{j-1}$ é o eixo anterior ao referencial.

Considerando as regras definidas anteriormente e definindo como posição zero, isto é, onde todos os ângulos de rotação são iguais a zero, como a posição mostrada na Figura 2, foram definidos os sistemas de coordenadas a seguir.

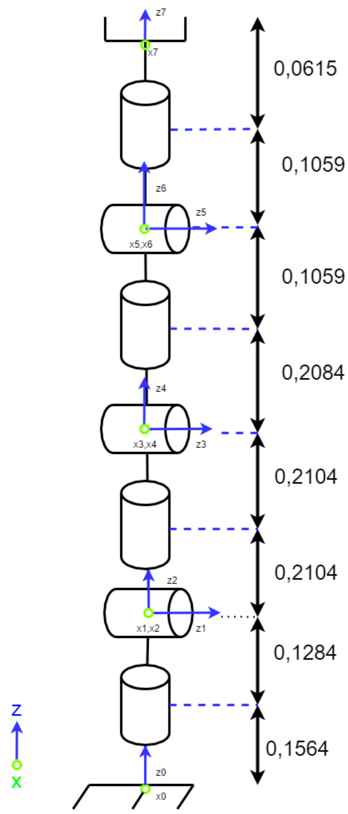

Figura 4: Definição dos sistemas de coordenadas para *Denavit-Hartenberg Standard*

2.1.2 Parâmetros

Após a definição das coordenadas de cada eixo, são obtidos os parâmetros de *Denavit-Hartenberg Standard*, estes são θ_i, d_i, a_i e α_i, são definidos por:

- θ_i - ângulo entre x_{i-1} e $\vec{x_i}$ ao redor de $\vec{z_{i-1}}$
- d_i - distância entre $\vec{z_{i-1}}$ e $\vec{z_i}$ ao longo de $\vec{z_{i-1}}$
- a_i - distância entre $\vec{z_{i-1}}$ e $\vec{z_i}$ ao longo de $\vec{x_i}$
- α_i - ângulo entre z_{i-1} e $\vec{z_i}$ ao redor de $\vec{x_{i-1}}$

Agora que foi definido o que cada parâmetro representa, podemos os definir seguindo as seguintes regras:

- θ_i* - Rotacionar o eixo $x\vec{x_i}$ em torno de $\vec{z_i}$ até ficar paralelo ao $\vec{x_{i+1}}$

- d_i* - Transladar o eixo de coordenadas j_i ao longo de $\vec{z_i}$ até a intersecção entre $\vec{z_i}$ e $\vec{x_{i+1}}$
- a_i - Transladar o eixo de coordenadas j_{i+1} ao longo de $\vec{x_{i+1}}$
- α_i - Rotacionar o eixo de $\vec{z_i}$ em torno de $\vec{x_{i+1}}$ até ficar paralelo ao eixo $\vec{z_{i+1}}$
- * Para juntas de rotação, o θ é variável e representado por θ_i
- * Para juntas prismáticas, o d é variável e representado por d_i

i	θ_i	d_i (m)	a_i (m)	α_i
1	θ_1	0,2488	0	$-\frac{\pi}{2}$
2	θ_2	0	0	$\frac{\pi}{2}$
3	θ_3	0,4208	0	$-\frac{\pi}{2}$
4	θ_4	0	0	$\frac{\pi}{2}$
5	θ_5	0,3143	0	$-\frac{\pi}{2}$
6	θ_6	0	0	$\frac{\pi}{2}$
7	θ_7	0,1674	0	0

2.2 Cinemática Inversa

A cinemática inversa é um método utilizado para determinar os parâmetros de uma junta de um manipulador a partir da configuração do efetuador (p_{be}, R_{be}) e tem como características:

1. Pode não existir solução
2. Podem existir múltiplas soluções
3. Podem existir infinitas soluções (manipuladores redundantes)

A cinemática inversa pode ser resolvida por vários métodos. como:

1. Desacoplamento Cinemático
2. Decomposição em sub-problemas
3. Algoritmo Interativo

Usaremos o método do Algoritmo Interativo através da função *kinova.ikine()* do pacote Robotic Toolbox, considerando a seguinte configuração:

$$R_{be} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$

$$p_{be} = [0.58, -0.2, 0.4]^T (m)$$

E sabendo que a Matriz de Transformação Homogênea é formada por:

$$T_{be}(\theta) = \begin{bmatrix} R_{be}(\theta) & p_{be}(\theta) \\ 0 & 1 \end{bmatrix}$$

Utilizando os valores de R_{be} e p_{be}, temos a seguinte Matriz de Transformação Homogênea.

$$T_{be}(\theta) = \begin{bmatrix} 0 & 0 & 1 & 0.58 \\ 1 & 0 & 0 & -0.2 \\ 0 & 1 & 0 & 0.4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.3 Cinemática Diferencial

O Jacobiano é uma parte integral da *Cinemática Diferencial*, ele estabelece a relação entre as velocidades das juntas $\dot{\theta}$ e a velocidade linear e angular do efetuador.

Temos que o Jacobiano Geométrico $J(\theta)$ é definido pela junção do Jacobiano de posição $J_p(\theta)$ e o Jacobiano de orientação $J_o(\theta)$, como podemos ver na equação abaixo.

$$J(\theta) = \begin{bmatrix} J_p(\theta) \\ J_o(\theta) \end{bmatrix}$$

Neste relatório será calculado o Jacobiano de posição do manipulador, considerando o punho com relação à base, sendo o punho o ponto de interseção das 3 últimas juntas, então ao calcularmos o Jacobiano de posição teremos a relação da velocidade linear do punho em função das velocidades das junta $\dot{\theta}_i$ com i = 1,...,4.

Como no Kinova Gen3 Ultra lightweight, nosso manipulador de estudo, só contém juntas de rotação, o Jacobiano de posição $J_p(\theta)$ é definido como:

$$J_{pn}(\theta) = \begin{bmatrix} \vec{h_1} \times \vec{p_{1n}} & \vec{h_2} \times \vec{p_{2n}} & ... & \vec{h_n} \times \vec{p_{nn}} \end{bmatrix} \quad (1)$$

O Jacobiano de posição também pode ser exprimido em um sistema de coordenadas, usaremos na base E_0.

$$(J_{p5})_0 = \begin{bmatrix} \widehat{(\vec{h_1})}_0(\vec{p_{15}})_0 & \widehat{(\vec{h_2})}_0(\vec{p_{25}})_0 & \widehat{(\vec{h_3})}_0(\vec{p_{35}})_0 & \widehat{(\vec{h_4})}_0(\vec{p_{45}})_0 \end{bmatrix}$$

Como estamos fazendo com o enfoque por produtos de exponenciais, o h_i pode ser determinado no sistema de coordenadas da base, então temos:

$$(J_{p5})_0 = \begin{bmatrix} \widehat{R_{01}(\vec{h_1})}_1(\vec{p_{15}})_0 & \widehat{R_{02}(\vec{h_2})}_2(\vec{p_{25}})_0 & \widehat{R_{03}(\vec{h_3})}_3(\vec{p_{35}})_0 & \widehat{R_{04}(\vec{h_4})}_4(\vec{p_{45}})_0 \end{bmatrix}$$

Temos que para os h_i com i ímpar, apontam para $\vec{z}$ e que os pares apontam para $\vec{y}$, como podemos ver na Figura abaixo:

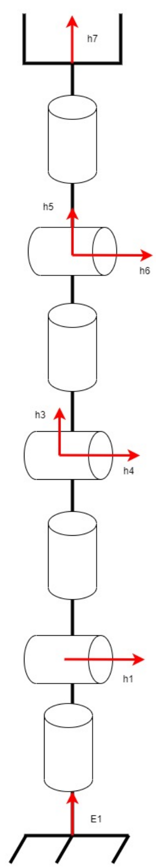

Figura 5: h_i por enfoque por produto exponenciais

Então temos:

$$h_1 = h_3 = h_5 = h_7 = \vec{z} = [0, 0, 1]$$

$$h_2 = h_4 = h_6 = \vec{y} = [0, 1, 0]$$

Podemos definir o p_i em relação ao p_{i-1} comparando as distâncias entre os h_i. Como a nossa posição zero é a do manipulador empinado, todas as distâncias serão no eixo $\vec{z}$, como podemos ver abaixo:

$$\begin{gathered}
\vec{p_{01}} = 0 \\
\vec{p_{12}} = (0,1564 + 0,2104)\vec{z_1} \\
\vec{p_{23}} = (0,2104 + 0,2104)\vec{z_2} \\
\vec{p_{34}} = 0 \\
\vec{p_{45}} = (0,2084 + 0,1059)\vec{z_4} \\
\vec{p_{56}} = 0 \\
\vec{p_{67}} = (0,1059 + 0,0615)\vec{z_6}
\end{gathered}$$

O p_i pode ser determinado pela seguinte expressão:

$$(\vec{p_i})_0 = (p_{i+1})_0 + R_{0,i-1}(p_{i-1,i})_{i-1}$$

Então temos:

$$(\vec{p_{15}})_0 = (\vec{p_{25}})_0 + R_{01}(p_{12})_1$$

$$(\vec{p_{25}})_0 = (\vec{p_{35}})_0 + R_{02}(p_{23})_2$$

$$(\vec{p_{35}})_0 = (\vec{p_{45}})_0 + R_{03}(p_{34})_3$$

$$(\vec{p_{45}})_0 = R_{04}(p_{45})_4$$

Sabendo a matriz de rotação é definida pela equação a seguir, com c_i e s_i, representando o *coseno* e o *seno* da junta i, respectivamente, logo temos:

$$R_{01} = e^{\hat{z}\theta_1} = \begin{bmatrix} c_1 & -s_1 & 0 \\ s_1 & c_1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

...

Agora que temos as matrizes de rotação em relação à junta anterior, é necessário definir em relação à base, isso é feito utilizando a máxima a seguir:

$$R_{02} = R_{01}R_{12}$$
$$R_{03} = R_{01}R_{12}R_{23}$$
...

2.4 Torque

O torque aplicado nas juntas é definido como:

$$\tau = J_0^T(\theta)(\vec{F_n})_o$$

Vamos determinar o torque, então usando o Jacobiano na posição home transposto usando a função $kinova.jacob0()$ e multiplicando pela força aplicada (-25N) no End effector Frame.

3 Resultados

3.1 Cinemática Direta

A partir da definição dos parâmetros, estes são inseridos em um código (Figura 6) para a simulação do modelo utilizando o Robot ToolBox do MATLAB.

```
clear L
%               theta     d            a        alpha   type mdh   offset
L(1) = Link([0          0.2848      0 -pi/2 0 0], 'standard');
L(2) = Link([0          0.0         0  pi/2 0 0], 'standard');
L(3) = Link([0          0.4208      0 -pi/2 0 0], 'standard');
L(4) = Link([0          0.0         0  pi/2 0 0], 'standard');
L(5) = Link([0          0.3143      0 -pi/2 0 0], 'standard');
L(6) = Link([0          0           0  pi/2 0 0], 'standard');
L(7) = Link([0          0.1674      0  0    0 0], 'standard' );

kinova=SerialLink(L, 'name', 'Projeto 1 - Erica Ferreira');

qz=[0 0 0 0 0 0 0];

kinova.teach(qz)
```

Figura 6: Código utilizado para a simulação no MATLAB

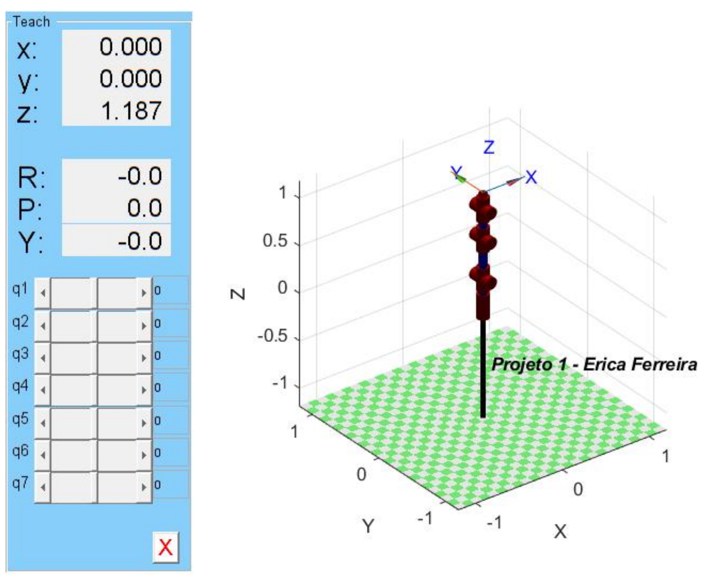

Figura 7: Simulação do Kinova Gen3 no MATLAB

3.1.1 Simulação com os parâmetros de Denavit-Hartenberg Standard

Para validar a simulação, 4 posições diferentes de fácil visualização foram testadas no modelo. A Matriz de Transformação Homogênea foi gerada usando a função *kinova.fkine(qz)*, sendo esta uma função do pacote Robotic Toolbox do *MATLAB*.

Para a primeira validação, temos uma rotação de $\pi/2$ na segunda junta (qz = [0 pi/2 0 0 0 0 0]).

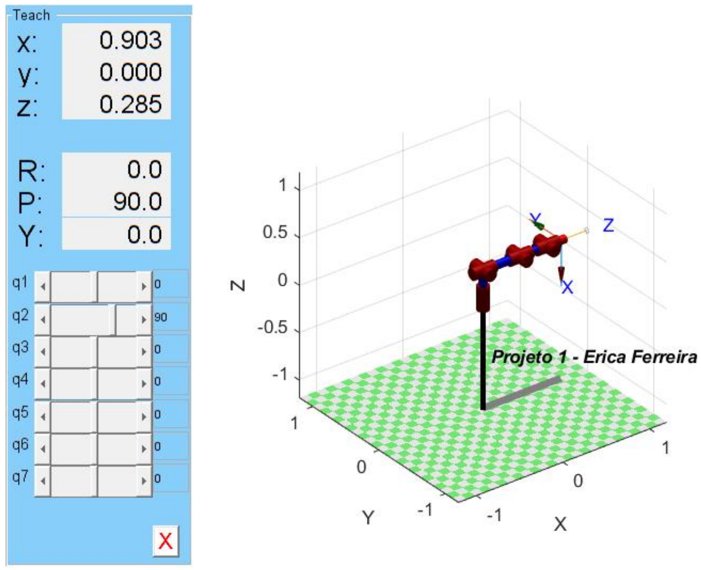

Figura 8: Plot da Configuração 1

ans =

$$\begin{matrix} 0 & 0 & 1 & 0.9025 \\ 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0.2848 \\ 0 & 0 & 0 & 1 \end{matrix}$$

Figura 9: Matriz de Transformação Homogênea da Configuração 1

Na segunda validação, foi feito uma rotação de $\pi/2$ na quarta junta (qz = [0 0 0 pi/2 0 0 0]).

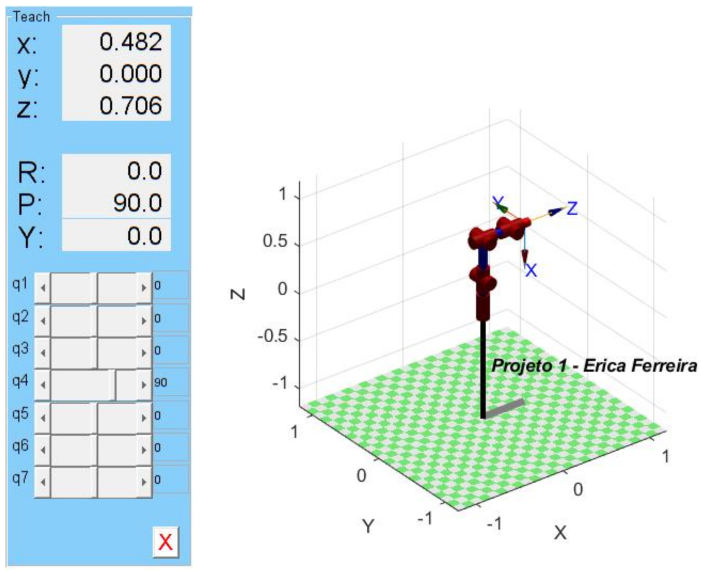

Figura 10: Plot da Configuração 2

```
ans =
          0         0         1    0.4817
          0         1         0         0
         -1         0         0    0.7056
          0         0         0         1
```

Figura 11: Matriz de Transformação Homogênea da Configuração 2

Já na terceira validação, foi feita uma rotação de $\pi/2$ na quarta junta e de $-\pi/2$ na sexta junta (qz = [0 0 0 pi/2 0 -pi/2 0]).

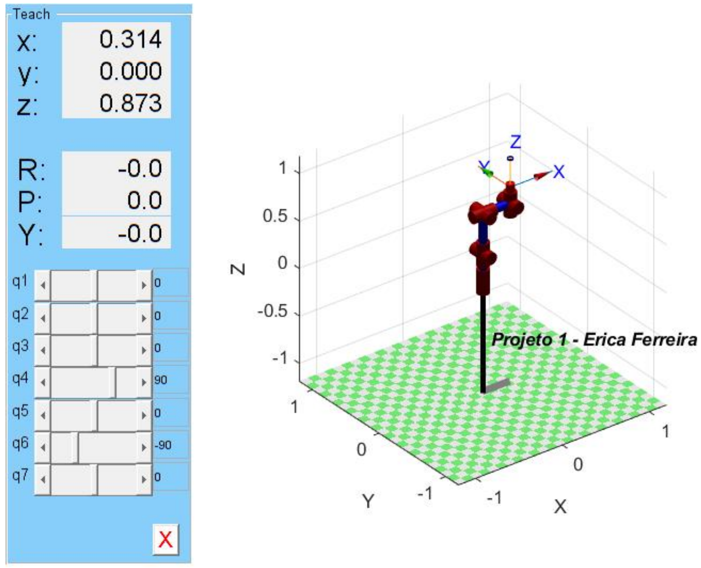

Figura 12: Plot da Configuração 3

```
ans =
          1          0          0     0.3143
          0          1          0          0
          0          0          1      0.873
          0          0          0          1
```

Figura 13: Matriz de Transformação Homogênea da Configuração 3

Para a última validação, foi feita uma rotação de $\pi/2$ na primeira e quarta juntas e uma rotação de $-\pi/2$ na sexta junta (qz = [pi/2 0 0 pi/2 0 -pi/2 0]).

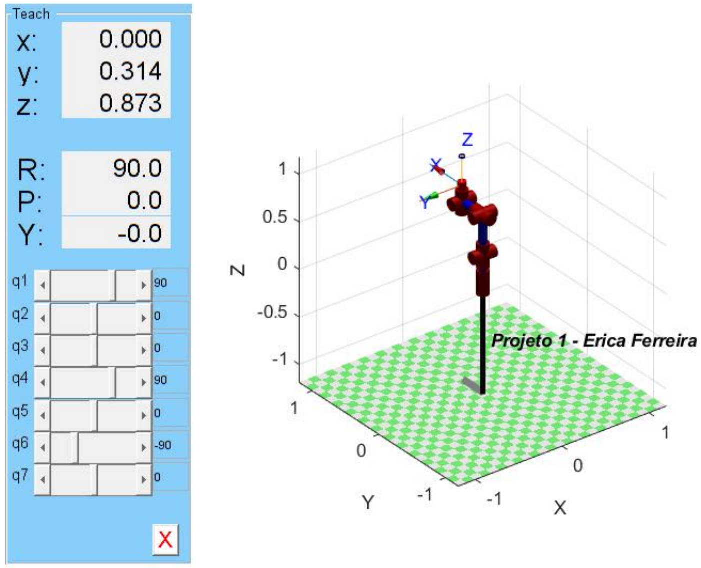

Figura 14: Plot da Configuração 4

```
ans =
          0        -1         0         0
          1         0         0    0.3143
          0         0         1     0.873
          0         0         0         1
```

Figura 15: Matriz de Transformação Homogênea da Configuração 4

3.2 Cinemática Inversa

Agora que a simulação foi validada utilizando cinemática direta, a mesma será validada por cinemática inversa. Para isso, vamos inserir uma matriz de transformação homogênea criada a partir da seguinte configuração de do Efetuador (End Effector Frame) na posição $p_{be} = [0.58, -0.2, 0.4]^T$ e a orientação mostrada na matriz abaixo:

$$T_{be}(\theta) = \begin{bmatrix} 0 & 0 & 1 & 0.58 \\ 1 & 0 & 0 & -0.2 \\ 0 & 1 & 0 & 0.4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Essa matriz é então utilizada com a função *kinova.ikine()*, que retorna os ângulos de cada junta em radianos. Estes ângulos são então inseridos na função *kinova.fkine()*, que irá nos retornar a mesma matriz que foi inserida na função de cinemática inversa, validando assim o resultado.

```
>> % Matriz de Transformação Homogênea
>> matriz

matriz =

         0         0    1.0000    0.5800
    1.0000         0         0   -0.2000
         0    1.0000         0    0.4000
         0         0         0    1.0000

>> % Calculando os ângulos de cada junta
>> % a partir da matriz de transformação
>> qz = kinova.ikine(matriz)

qz =

   -1.2041    1.0222    1.4427    1.7093    0.1513   -0.4547    0.4222

>> % Calculando a Matriz de Transformação Homogênea
>> % a partir dos ângulos de cada junta
>> kinova.fkine(qz)


ans =
         0         0         1      0.58
         1         0         0      -0.2
         0         1         0       0.4
         0         0         0         1
>> % Mesma matriz foi retornada, o que valida os nossos resultados
```

Figura 16: Código do *MATLAB* utilizado para a validação por cinemática inversa

3.3 Cinemática Diferencial

Com todas as variáveis definidas, esses valores serão o *input* do seguinte código no *MATLAB* para o cálculo do Jacobiano de Posição em relação à base:

```
>> % Apontando para z
h1 = [0; 0; 1];
h3 = [0; 0; 1];
h5 = [0; 0; 1];
h7 = [0; 0; 1];

% Apontando para y
h2 = [0; 1; 0];
h4 = [0; 1; 0];
h6 = [0; 1; 0];

% vetor posição em relação à junta anterior
p01 = [0; 0; 0];
p12 = [0; 0; 0.1564 + 0.2104];
p23 = [0; 0; 0.2104 + 0.2104];
p34 = [0; 0; 0];
p45 = [0; 0; 0.2084 + 0.1059];
p56 = [0; 0; 0];
p67 = [0; 0; 0.1059 + 0.0615];

%ângulo das juntas na posição zero ("empinado")
q = [0 0 0 0 0 0 0];
```

Figura 17: Inserção de h_i e $p_{i-1,i}$

```
%Matriz rotação
R01 = [cos(q(1)) -sin(q(1)) 0; sin(q(1)) cos(q(1)) 0; 0 0 1];
R23 = [cos(q(3)) -sin(q(3)) 0; sin(q(3)) cos(q(3)) 0; 0 0 1];
R45 = [cos(q(5)) -sin(q(5)) 0; sin(q(5)) cos(q(5)) 0; 0 0 1];
R67 = [cos(q(7)) -sin(q(7)) 0; sin(q(7)) cos(q(7)) 0; 0 0 1];

R12 = [cos(q(2)) 0  sin(q(2));  0 1 0; -sin(q(2)) 0 cos(q(2))];
R34 = [cos(q(4)) 0  sin(q(4));  0 1 0; -sin(q(4)) 0 cos(q(4))];
R56 = [cos(q(6)) 0  sin(q(6));  0 1 0; -sin(q(6)) 0 cos(q(6))];

R02 = R01 * R12;
R03 = R01 * R12 * R23;
R04 = R01 * R12 * R23 * R34;
R05 = R01 * R12 * R23 * R34 * R45;
R06 = R01 * R12 * R23 * R34 * R45 * R56;
R07 = R01 * R12 * R23 * R34 * R45 * R56 * R67;
```

Figura 18: Cálculo da Matriz Rotação

Então temos como resultado, a partir desse código:

```
p45_0 = R04*p45;
p35_0 = R03*p34 + p45_0;
p25_0 = R02*p23 + p35_0;
p15_0 = R01*p12 + p25_0;

h_10 = (R01*h1);
h_20 = (R02*h2);
h_30 = (R03*h3);
h_40 = (R04*h4);
h_50 = (R05*h5);
h_60 = (R06*h6);
h_70 = (R07*h7);

h1_0 = [0 -h_10(3) h_10(2); h_10(3) 0 -h_10(1) ; -h_10(2) h_10(1) 0];
h2_0 = [0 -h_20(3) h_20(2); h_20(3) 0 -h_20(1) ; -h_20(2) h_20(1) 0];
h3_0 = [0 -h_30(3) h_30(2); h_30(3) 0 -h_30(1) ; -h_30(2) h_30(1) 0];
h4_0 = [0 -h_40(3) h_40(2); h_40(3) 0 -h_40(1) ; -h_40(2) h_40(1) 0];


% Jacobiano de posição
J5p= [h1_0*p15_0 h2_0*p25_0  h3_0*p35_0  h4_0*p45_0]
```

Figura 19: Cálculo do Jacobiano de Posição J_{5p}

```
% Jacobiano de posição
J5p= [h1_0*p15_0 h2_0*p25_0  h3_0*p35_0  h4_0*p45_0]

J5p =

         0    0.7351         0    0.3143
         0         0         0         0
         0         0         0         0

>> kinova.jacob0(q)
```

Figura 20: Jacobiano de Posição gerado pelo código das Figuras 16,17,18

3.3.1 Validação do Jacobiano

O Jacobiano de posição foi validado usando a função *kinova.jacob*0() a partir do modelo construído segundo os parâmetros de Denavit-Hartenberg. Uma modificação, foi feita, entretanto, como pode ser vista na figura abaixo. Como foi considerada que as 3 últimas juntas tem a mesma origem, elas passam a ter distância 0.

```
%All link lenghts and offsets are measured in cm
clear L
%            theta    d          a       alpha  type mdh  offset
L(1) = Link([0        0.2848     0 -pi/2 0 0], 'standard');
L(2) = Link([0        0.0        0  pi/2 0 0], 'standard');
L(3) = Link([0        0.4208     0 -pi/2 0 0], 'standard');
L(4) = Link([0        0.0        0  pi/2 0 0], 'standard');
L(5) = Link([0        0.3143     0 -pi/2 0 0], 'standard');
L(6) = Link([0        0          0  pi/2 0 0], 'standard');
L(7) = Link([0        0          0  0    0 0], 'standard' );

kinova=SerialLink(L, 'name', 'Projeto 1 - Erica Ferreira');

q = [0 0 0 0 0 0 0 ];

kinova.teach(q)
```

Figura 21: Código no MATLAB modificado

Foram comparadas as seguintes posições:

$$q = [0\ 0\ 0\ 0\ 0\ 0\ 0]$$
$$q = [0\ 0\ \pi/2\ 0\ 0\ 0\ 0]$$
$$q = [0\ 0\ 0\ \pi/2\ 0\ 0\ 0]$$

Para a configuração 1, temos os seguintes resultados, podemos ver que são compatíveis, assim como os das configurações 2 e 3.

```
% Jacobiano de posição
J5p= [h1_0*p15_0 h2_0*p25_0  h3_0*p35_0  h4_0*p45_0]

J5p =

         0    0.7351         0    0.3143
         0         0         0         0
         0         0         0         0

>> kinova.jacob0(q)
```

Figura 22: Jacobiano de posição da Configuração 1

```
>> kinova.jacob0(q)

ans =

         0    0.7351         0    0.3143         0         0         0
         0         0         0         0         0         0         0
         0         0         0         0         0         0         0
         0         0         0         0         0         0         0
         0    1.0000         0    1.0000         0    1.0000         0
    1.0000    0.0000    1.0000    0.0000    1.0000    0.0000    1.0000
```

Figura 23: Jacobiano da Configuração 1

```
% Jacobiano de posição
J5p= [h1_0*p15_0 h2_0*p25_0  h3_0*p35_0  h4_0*p45_0]

J5p =

         0    0.0000         0    0.0000
    0.7351         0         0         0
         0   -0.7351         0   -0.3143
```

Figura 24: Jacobiano de posição da Configuração 2

```
>> kinova.jacob0(q)

ans =

   -0.0000    0.0000         0    0.0000         0         0         0
    0.7351    0.0000         0    0.0000         0         0         0
    0.0000   -0.7351         0   -0.3143         0         0         0
         0         0    1.0000         0    1.0000         0    1.0000
    0.0000    1.0000    0.0000    1.0000    0.0000    1.0000    0.0000
    1.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000
```

Figura 25: Jacobiano da Configuração 2

```
% Jacobiano de posição
J5p= [h1_0*p15_0 h2_0*p25_0  h3_0*p35_0  h4_0*p45_0]

J5p =

         0    0.4208         0    0.0000
    0.3143         0    0.3143         0
         0   -0.3143         0   -0.3143
```

Figura 26: Jacobiano de posição da Configuração 3

```
>> kinova.jacob0(q)

ans =

   -0.0000    0.4208   -0.0000    0.0000         0         0         0
    0.3143    0.0000    0.3143    0.0000         0         0         0
    0.0000   -0.3143    0.0000   -0.3143         0         0         0
         0         0         0         0    1.0000         0    1.0000
    0.0000    1.0000    0.0000    1.0000    0.0000    1.0000    0.0000
    1.0000    0.0000    1.0000    0.0000    0.0000    0.0000    0.0000
```

Figura 27: Jacobiano da Configuração 3

3.4 Torque

Agora que foi definido o torque ele é calculado utilizando o Jacobiano gerado por $kinova.jacob0()$ para a posição home com a configuração original por DH (sem juntar as três últimas juntas), como podemos ver no código abaixo.

Vamos calcular o torque para uma força de $-25N$ na vertical, como queremos no final o torque realizado pelas juntas para se manter nessa posição, que é o oposto do torque infligido nas juntas, será calculado para uma força de $25N$ na vertical, o que já retornará o resultado esperado.

```
>> q_home = [0 15 180 230 0 55 90]* pi /180;

jacobiano_no_home = kinova.jacob0(q_home);

% Força de 25N na terceira junta
torque = (jacobiano_no_home') * [0; 0; 25; 0; 0; 0;];
```

Figura 28: Código no *MATLAB* para o cálculo do τ

Então temos como resultado:

```
disp(torque)
   -0.0000
  -11.4147
   -0.0000
    8.6919
    0.0000
    4.1850
         0
```

Figura 29: τ calculado

4 Conclusão

Neste relatório calculamos a cinemática direta, inversa, diferencial e o torque para o modelo Kinova Gen3 Ultra lightweight, durante esses cálculos foram testadas e decorrente disso, entendidas, funções do *MATLAB*, como $.fkine()$ para cinemática direta, $.ikine()$, para a inversa e $.jacob0()$ para o jacobiano geométrico (cinemática diferencial).