Faculdade de Engenharia da Universidade do Porto

Controlo de um manipulador antropomórfico de 3 eixos

Relatório do Trabalho Prático

Unidade Curricular

Robótica Industrial

Autores

Alberto Santos **| up201708979**

Ana Alves | **up201506789**

Francisco Neves **| up201404576**

Rita Ferreira **| up201604728**

Junho 2020

Índice

1. Introdução

No âmbito da unidade curricular Robótica Industrial, foi proposto um trabalho prático que consistia no controlo de um manipulador antropomórfico de 3 eixos.

Para este efeito, utilizou-se o simulador SimTwo2020 com o cenário "Manipulator3DoF" disponibilizado pelos docentes.

O simulador disponibiliza um modelo 3D do braço robótico e um ambiente de programação em linguagem Pascal que permite a programação e visualização gráfica do controlo programado.

O manipulador robótico em estudo possui as seguintes características:

- 3 articulações rotativas
- Comprimento do 1º ligamento (da base para a 1ª articulação): 0.55m
- Comprimento do 2º ligamento (da 1ª articulação para a 2ª articulação): 0.4m
- Comprimento do 3º ligamento (da 2ª articulação para a 3ª articulação): 0.325m

O objetivo do trabalho é a implementação de três processos matemáticos usados em robótica para calcular variáveis de controlo:

- Cinemática direta
- Cinemática inversa
- Jacobiano

2. Cinemática Direta

2.1. Introdução e Cálculos

Neste capítulo, utilizar-se-á a cinemática direta para calcular a posição cartesiana da extremidade do manipulador.

A cinemática direta é um processo matemático que calcula a posição cartesiana do efetuador a partir dos valores dos parâmetros das articulações. Esses parâmetros dependem do tipo de articulações: são orientações angulares, posições cartesianas, ou velocidades angulares se forem articulações rotativas, prismáticas ou contínuas, respetivamente.

Assim, a partir da figura 1, com os eixos devidamente desenhados e com recurso ao método de *Denavit-Hartenberg*, obtemos a tabela 1.

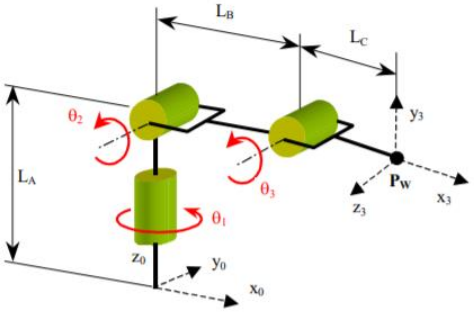

Figura 1 - Manipulador Antropomórfico de 3 eixos

i	θ	d	a	α
0	θ1	d1	0	-90º
1	θ2	0	d2	0
2	θ3	0	d3	0

Tabela 1

O método *Denavit-Hartenberg*, afirma que cada transformação homogénea tem a seguinte forma:

$$A_i = Rot_{z,\theta_i}\ Trans_{z,d_i}\ Trans_{x,a_i}\ Rot_{x,\alpha_i}$$

$$= \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_i & -s\alpha_i & 0 \\ 0 & s\alpha_i & c\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Que resulta:

$$A_i = \begin{bmatrix} c\theta_i & -s\theta_i\, c\alpha_i & s\theta_i\, s\alpha_i & a_i c\theta_i \\ s\theta_i & c\theta_i\, c\alpha_i & -c\theta_i\, s\alpha_i & a_i s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Logo,

${}^0A_1=\left[Rot_{z,\theta 1} * Trans_{z,d1} * Trans_{x,0} * Rot_{x,90}\right]$;

${}^1A_2=\left[Rot_{z,\theta 2} * Trans_{z,0} * Trans_{x,d2} * Rot_{x,0}\right]$;

${}^2A_3=\left[Rot_{z,\theta 3} * Trans_{z,0} * Trans_{x,d3} * Rot_{x,0}\right]$.

Assim, podemos obter a matriz ${}^0A_3$, representada a seguir, a partir da multiplicação de ${}^0A_1$ por ${}^1A_2$ por ${}^2A_3$.

$${}^0A_3 = \begin{bmatrix} \mathrm{C1C2C3} - \mathrm{C1S2S3} & -S3C1C2 - C3C1S2 & s1 & 0.325C1C2C3 - 0.325\mathrm{C1S2S3} + 0{,}4C1C2 \\ C2S1C3 - S1S2S3 & -S3C2S1 - C3S1S2 & -C1 & 0.325C3C2S1 - 0.325S3S1S2 + 0{,}4C2S1 \\ \mathrm{C3S2} + \mathrm{C2S3} & -S3S2 + C3C2 & 0 & 0.325C3S2 + 0.325S3C2 + 0{,}4S2 + 0.55 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Onde:

C1=cos(θ_1); C2=cos(θ_2); C3=cos(θ_3); S1=sin(θ_1); S2=sin(θ_2); S3=sin(θ_3).

2.2. Simulador

Em relação à implementação em código, a seguinte função DK3, foi criada e devidamente comentada:

```
function DK3(Thetas: matrix): matrix;
var
  A01, A12, A23, A03 :Matrix; // DH matrix variables
  XYZ_column_vector :Matrix;  // position of the end effector vector
begin
  // Compute the homogeneous DH transformation matrixes
  // using DHMat function:
  // parameters: a, alpha, d, theta
  // return: DH transformation matrix
  A01 := DHMat(0, -pi*0.5, dl , Mgetv(Thetas,0,0));
  A12 := DHMat(d2, 0, 0, Mgetv(Thetas,1,0));
  A23 := DHMat(d3, 0, 0, Mgetv(Thetas,2,0));

  // Compute the DH transformation matrix between base joint and end-effector
  A03 := MMult(MMult(A01,A12),A23);

  // Get the position of the end-effector from the last column of A03 matrix
  XYZ_column_vector := Mzeros(3,1);
  Msetv(XYZ_column_vector, 0, 0, Mgetv(A03, 0, 3));
  Msetv(XYZ_column_vector, 1, 0, Mgetv(A03, 1, 3));
  Msetv(XYZ_column_vector, 2, 0, Mgetv(A03, 2, 3));

  // return the position vector
  result := XYZ_column_vector;
end;
```

Como o braço antropomórfico em estudo consiste em 3 articulações rotativas, a função recebe como entrada um vetor coluna com os 3 ângulos das articulações em radianos (θ_1, θ_2, θ_3). Como output, a função retorna a posição do efetuador (x, y, z).

Abaixo demonstram-se dois exemplos de uso da função DK3.

As seguintes figuras são *snapshots* retirados à *sheet* e ao ambiente gráfico do programa SimTwo.

Exemplo 1:

Tendo como input:

- Ângulo da articulação 1: 20º
- Ângulo da articulação 2: 10º
- Ângulo da articulação 3: -20º

Direct
20
10
-20

A função responde com o output:

- Posição em X do efetuador: 0.671 m
- Posição em Y do efetuador: 0.244 m
- Posição em Z do efetuador: 0.537 m

DK
0.671
0.244
0.537

E a representação gráfica da posição do braço:

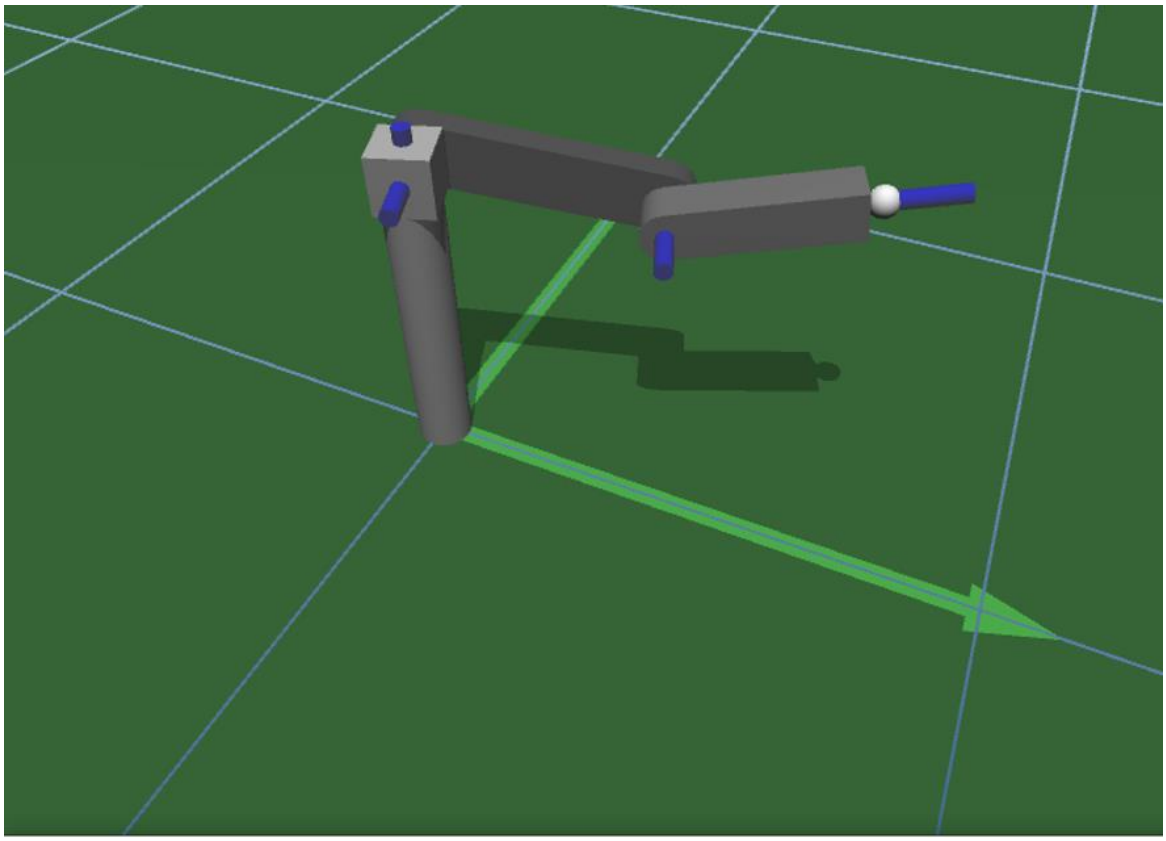

Figura 2 - Representação gráfica da posição do braço para o exemplo 1 da cinemática direta (o eixo X é o eixo verde-claro que aponta para a direita. O eixo Y é o eixo verde-claro o que aponta para cima)

Exemplo 2:

Tendo como input:

- Ângulo da articulação 1: -30º
- Ângulo da articulação 2: -10º
- Ângulo da articulação 3: 20º

Direct
-30
-10
20

A função responde com o output:

- Posição em X do efetuador: 0.618 m
- Posição em Y do efetuador: -0.357 m
- Posição em Z do efetuador: 0.563 m

DK
0.618
-0.357
0.563

E a representação gráfica da posição do braço:

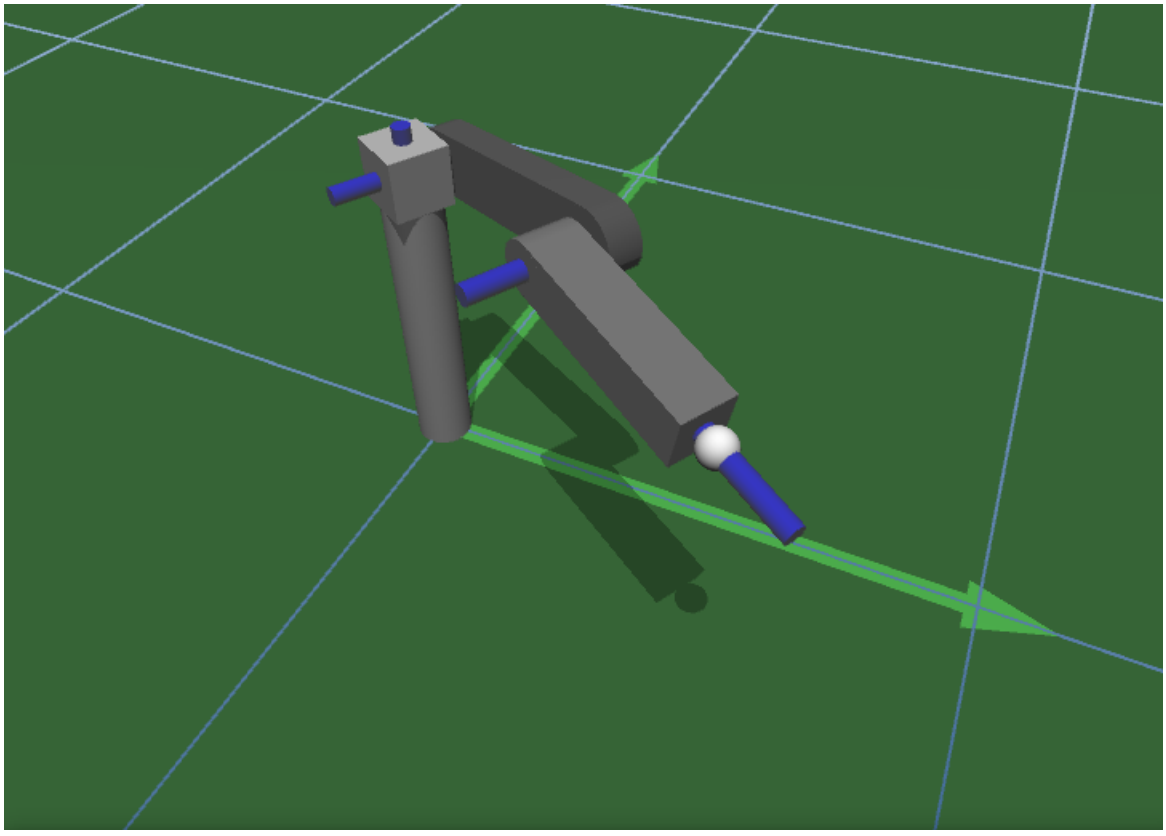

Figura 3 - Representação gráfica da posição do braço para o exemplo 2 da cinemática direta (o eixo X é o eixo verde-claro que aponta para a direita. O eixo Y é o eixo verde-claro o que aponta para cima)

3. Cinemática Inversa

3.1. Introdução e Cálculos

Neste capítulo, utilizar-se-á a cinemática inversa para calcular os valores dos parâmetros de cada articulação (no caso específico do problema, esses parâmetros são os ângulos das articulações rotativas) de modo a que a extremidade do manipulador se posicione num dado ponto cartesiano do espaço 3D. Ao contrário da cinemática direta, em que se teve como entrada os parâmetros das articulações e como saída a posição cartesiana do efetuador, neste método vamos ter como entrada as posições cartesianas do efetuador e como saída os ângulos das articulações.

Com recurso a trigonometria e à lei dos cossenos, demonstrar-se-á a seguir como obter cada ângulo de articulação.

- Comecemos pela determinação do 1º ângulo (o ângulo da 1ª articulação). Chamaremos a esse parâmetro θ_1. Pode observar-se na figura abaixo uma relação direta entre a posição x e y do efetuador e a altura z do efetuador para determinar o ângulo θ_1:

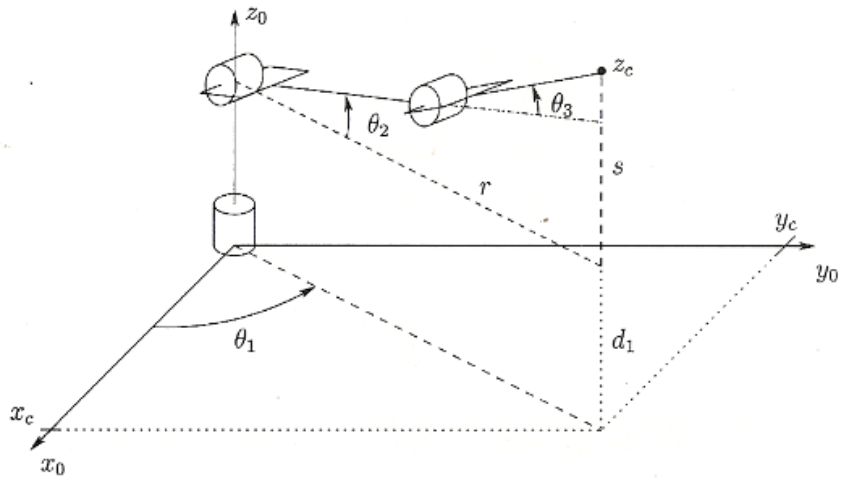

Figura 4 - Representação dos ângulos θ1, θ2 e θ3 no espaço

Usar-se-á a função *Atan2* com vista a questões de normalização de ângulos. θ_1 obtém-se então da seguinte forma:

$$z = z_c$$

$$y = y_c$$

$$x = x_c$$

$$s = z - d_1$$

$$c^2 = s^2 + r^2$$

$$r = \sqrt{x^2 - y^2}$$

$$\theta_1 = \mathrm{Atan2}\left(\frac{y}{x}\right)$$

- Passemos agora à determinação do 2º ângulo (o ângulo da 2ª articulação). Chamaremos a esse parâmetro θ_2. Pode observar-se na figura abaixo uma relação direta entre o ângulo Φ, o ângulo β e o ângulo θ_2. Tira-se a seguinte relação:

$$\theta_2 = \Phi - \beta$$

Para obter β, e tendo em conta a notação das duas figuras abaixo, usar-se-á a lei dos cossenos e obtém-se a seguinte relação:

$$d_3{}^2 = d_2{}^2 + c^2 - 2d_2 d_3 \cos(\beta)$$

$$\beta = \cos^{-1}\left(\frac{c^2 + d_2{}^2 - d_3{}^2}{2d_2 c}\right)$$

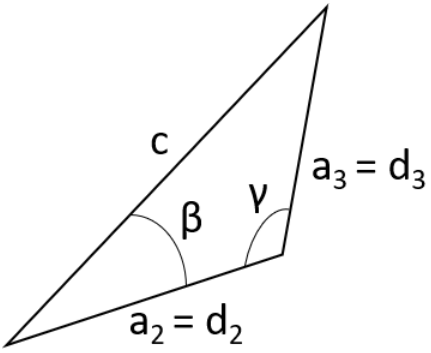

Figura 5 - Representação do ângulo β

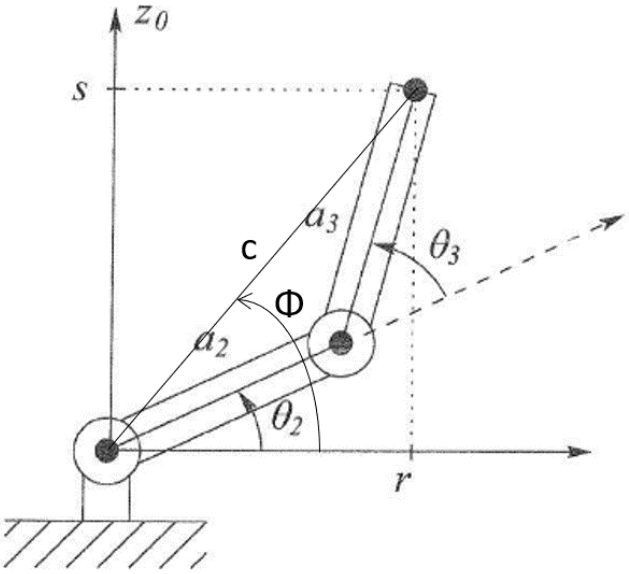

Figura 6 - Representação do ângulo Φ

Para obter Φ, usar-se à a relação direta entre s e r:

$$\Phi = \text{Atan2}\left(\frac{\text{s}}{\text{r}}\right) = \text{Atan2}\left(\frac{z - d_1}{\sqrt{x^2 - \text{y}}}\right)$$

- Concluiremos com o cálculo 3º ângulo (o ângulo da 3ª articulação). Chamaremos a esse parâmetro θ_3. Pode observar-se na figura abaixo uma relação direta entre o ângulo γ e θ_3. Tira-se a seguinte relação:

$$\theta_3 = \pi - \gamma$$

Para obter γ, usar-se-á a lei dos cossenos novamente:

$$c^2 = {d_2}^2 + {d_3}^2 - 2\text{d}_2\text{d}_3\cos(\gamma)$$

$$\gamma = \cos^{-1}\left(\frac{c^2 - {d_2}^2 - {d_3}^2}{2\text{d}_2\text{d}_3}\right)$$

Nota: A cinemática inversa tem várias soluções possíveis para diferentes configurações de braço. Escolheu-se a configuração "*Left Arm Elbow Down"*, logo todo o controlo funcionará para operações com o braço à esquerda e de cotovelo para baixo como o próprio nome da configuração indica.

3.2. Simulador

Em relação á implementação em código, a seguinte função IK3, foi criada e devidamente comentada:

```
function IK3(XYZ: matrix): matrix;
var
  beta, phi, s, r, c, d, x, y, z, num, den, Theta1, Theta2, Theta3: double;
  Thetas: Matrix;
begin
  // desired positions
  x := Mgetv(XYZ, 0, 0);
  y := Mgetv(XYZ, 1, 0);
  z := Mgetv(XYZ, 2, 0);

  // auxiliar terms
  s := z - d1;
  r := sqrt(sqr(x) + sqr(y));
  c := sqrt(sqr(s) + sqr(r));
  num := sqr(c) - Sqr(d2) - Sqr(d3);
  den := 2 * d2 * d3;
  phi := Atan2(s,r);
  beta := arccos((sqr(c) + Sqr(d2) - Sqr(d3)) / (2 * d2 * c));

  // angles computation
  Theta1 := Atan2(y,x);
  Theta2 := -(phi - beta);
  Theta3 := -arccos(num/den);
  Thetas := Mzeros(3, 1);
  Msetv(Thetas, 0, 0, Theta1);
  Msetv(Thetas, 1, 0, Theta2);
  Msetv(Thetas, 2, 0, Theta3);

  //return the angles vector
  result := Thetas;
end;
```

A função recebe como input um vetor coluna com as 3 posições cartesianas (x, y, z) do efetuador e como output retorna os ângulos das articulações pretendidos (θ_1, θ_2, θ_3) para obter a posição desejada do efetuador.

Abaixo demonstram-se dois exemplos de uso da função IK3.

Nota: A implementação de θ_2 e θ_3 no código é simétrica à representação matemática descrita no capítulo 3.1., visto que o simulador tem como convenção o crescimento dos ângulos no sentido horário enquanto que nas equações convencionou-se o crescimento dos ângulos no sentido anti-horário.

As seguintes figuras são *snapshots* retirados à *sheet* e ao ambiente gráfico do programa SimTwo.

Exemplo 1:

Tendo como input:

- Posição em X do efetuador: 0.685 m
- Posição em Y do efetuador: 0.121 m
- Posição em Z do efetuador: 0.470 m

indirect
0.685
0.121
0.470

A função responde com o output:

- Ângulo da articulação 1: 10º
- Ângulo da articulação 2: 20,1º
- Ângulo da articulação 3: -30,2º

IK
10.0
20.1
-30.2

E a representação gráfica da posição do braço:

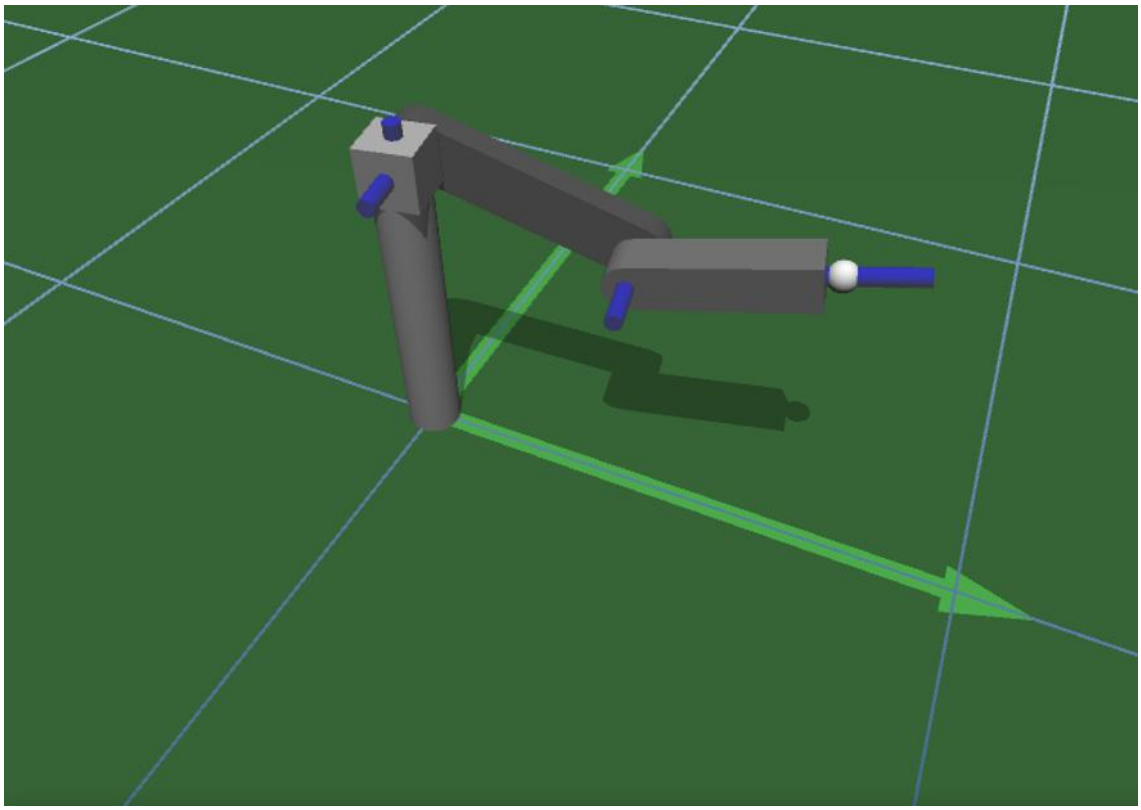

Figura 7 - Representação gráfica da posição do braço para o exemplo 1 da cinemática inversa *(o eixo X é o eixo verde-claro que aponta para a direita. O eixo Y é o eixo verde-claro o que aponta para cima)*

Exemplo 2:

Tendo como input:

- Posição em X do efetuador: 0.589 m
- Posição em Y do efetuador: -0.214 m
- Posição em Z do efetuador: 0.349 m

indirect
0.589
-0.214
0.349

A função responde com o output:

- Ângulo da articulação 1: -20º
- Ângulo da articulação 2: 40º
- Ângulo da articulação 3: -49,9º

IK
-20.0
40.0
-49.9

E a representação gráfica da posição do braço:

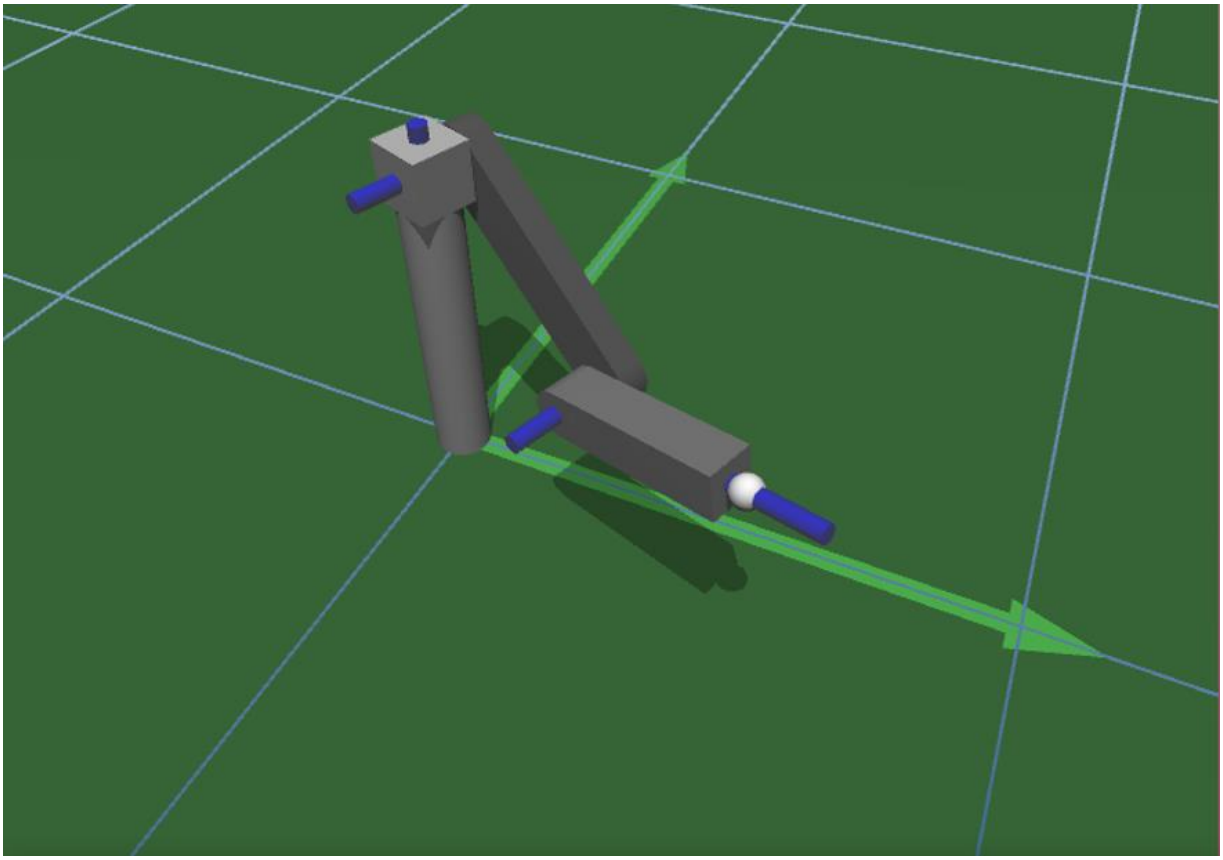

Figura 8 - Representação gráfica da posição do braço para o exemplo 2 da cinemática inversa *(o eixo X é o eixo verde-claro que aponta para a direita. O eixo Y é o eixo verde-claro o que aponta para cima)*

4. Jacobiano

4.1. Introdução e Cálculos

Neste capítulo, vamos utilizar o Jacobiano do controlador para calcular a velocidade a impor a cada articulação de modo a ter a extremidade do manipulador a mover-se com uma determinada velocidade linear.

O Jacobiano é dado por:

$$\begin{bmatrix} v_n^0 \\ w_n^0 \end{bmatrix} = \begin{bmatrix} J_v \\ J_w \end{bmatrix} \dot{q}\,(t)$$

Como apenas nos interessa a velocidade linear, vamos considerar apenas o Jv.

Assim:

$$J_v = [J_{v1}\ \ J_{v2} \ldots J_{vn}], \quad J_{vi} = \begin{cases} z_{i-1}^0 \times (o_n^0 - o_{i-1}^0) & \text{- articulação rotativa} \\ z_{i-1}^0 & \text{- articulação prismática} \end{cases}$$

Como temos apenas articulações rotativas consideramos apenas a primeira linha do sistema apresentado anteriormente.

A matriz de transformação homogénea obtida na cinemática direta está sob a forma de:

$$H = \begin{bmatrix} R & d \\ 0 & 1 \end{bmatrix}$$

A partir das ${}^0A_1$, ${}^0A_2$ e ${}^0A_3$ é possível retirar as matrizes ${}^0R_0$, ${}^0R_1$, ${}^0R_2$, ${}^0d_1$, ${}^0d_2$ e ${}^0d_3$, uma vez que a matriz Jacobiana irá ser calculada por:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} R_0^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} (d_3^0 - d_0^0) & R_1^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} (d_3^0 - d_1^0) & R_2^0 \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} (d_3^0 - d_2^0) \end{bmatrix} \begin{bmatrix} \theta 1 \\ \theta 2 \\ \theta 3 \end{bmatrix}$$

Obtendo-se:

$$R_0^0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad R_1^0 = \begin{bmatrix} C1 & 0 & -S1 \\ S1 & 0 & C1 \\ 0 & -1 & 0 \end{bmatrix} \quad R_2^0 = \begin{bmatrix} C1C2 & -C1S2 & -S1 \\ S1C2 & -S1S2 & C1 \\ -S2 & -C2 & 0 \end{bmatrix}$$

$$d_3^0 = \begin{bmatrix} 0{,}325C1C2C3 - 0.325C1S2S3 + 0.4C1C2 \\ 0.325C3C2S1 - 0.325S1S2S3 + 0.4C2S1 \\ 0.325S3C2 + 0.325C3S2 + 0.4S2 + 0.55 \end{bmatrix} \quad d_2^0 = \begin{bmatrix} 0.4C1C2 \\ 0.4S1C2 \\ -0.4S2 + 0.55 \end{bmatrix} \quad d_1^0 = \begin{bmatrix} 0 \\ 0 \\ 0.55 \end{bmatrix}$$

Efetuando os cálculos, é possível obter a seguinte matriz Jacobiana:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = [1ª\ Coluna\ (J1) \quad 2ª\ Coluna\ (J2) \quad 3ª\ Coluna\ (J3)] \begin{bmatrix} \theta 1 \\ \theta 2 \\ \theta 3 \end{bmatrix}$$

$$[J1] = \begin{bmatrix} -0.325C3C2S1 + 0.325S3S1S2 - 0.4C2S1 \\ 0.325C3C1C2 - 0.325C1S2S3 + 0.4C2C1 \\ 0 \end{bmatrix}$$

$$[J2] = \begin{bmatrix} 0.325C1C2S3 + 0.325S2C3C1 + 0.4S2C1 \\ 0.325S3C2S1 + 0.325C3S2S1 + 0.4S2S1 \\ -0.325C3C2S1S1 + 0.325S1S1S2S3 - 0.4C2S1S1 - 0.325C1C1C2C3 + 0.325C1C1S2S3 - 0.4C1C1C2 \end{bmatrix}$$

$$[J3] = \begin{bmatrix} 0.325S3C2C1 + 0.325C3S2C1 + 0.8S2C1 \\ 0.325S3C2S1 + 0.325C3S2S1 + 0.8S1S2 \\ -0.325C3C2S1S1 + 0.325S1S1S2S3 - 0.325C1C1C2C3 + 0.325C1C1S2S3 \end{bmatrix}$$

Sabendo que:
Velocidades do efetor = Matriz Jacobiana * Velocidades das articulações,
então:
Velocidades das articulações = Matriz Jacobiana Invertida * Velocidades do efetor

4.2. Simulador

Em relação à implementação de código, foram implementadas uma função principal e várias subfunções auxiliares de modo a tornar o código mais limpo e modular. Abaixo descrever-se-á e mostrar-se-á o corpo de cada função bem como o seu papel.

A função principal *ComputeJointVelocities* recebe como entrada um vetor coluna das velocidades lineares desejadas (V_x, V_y, V_z) para o efetuador e responde com duas possíveis saídas, dependendo de a condição de singularidade do manipulador ter sido atingida ou não.

Num breve resumo, a condição de singularidade atinge-se quando uma determinada configuração solicitada pelo controlador ao manipulador não pode ser atingida. Esta situação pode ser atingida, por exemplo, quando o braço do robô está completamente esticado e é solicitada uma posição no seu espaço de trabalho para o qual ele já não chega, porque atingiu os limites do seu comprimento.

Para converter velocidades do efetuador em velocidades de articulações é necessário inverter a matriz do jacobiano, como se pode verificar no capítulo 4.1., e isso implica uma divisão pelo determinante do jacobiano. Acontece que o determinante do jacobiano em configurações de singularidade aproxima-se de zero, o que implica que se tenha de fazer uma divisão por zero e se atinjam valores incomportáveis de velocidades nas articulações. Em conclusão, para controlar a situação de singularidade é necessário verificar o valor do determinante para que não aconteçam, por exemplo, num contexto real, danos na hardware.

As duas possíveis saídas da função são:

- Se atingida a condição singularidade é retornada uma matriz nula de dimensão 3 e o cálculo não é efetuado;
- Caso contrário, faz a inversão da matriz e retorna a multiplicação da matriz inversa pelo vetor coluna das velocidades, ou seja, a velocidade das articulações.

Abaixo um screenshot da função *ComputeJointVelocities*:

```
// Using the inverted jacobian computes and returns the joint velocities
// Takes as input the desired velocity of the end effector
function ComputeJointVelocities(endEffectorVelocity: matrix): matrix;
var
  Jacobian :matrix;
  invertedJacobian :matrix;
begin
   Jacobian := computeJacobian;
  //Check singularity
  if checkSingularity(Jacobian) then begin
    result := Mzeros(3,1);
  end
  else begin
    invertedJacobian := Minv(Jacobian);
    result := Mmult(invertedJacobian, endEffectorVelocity);
  end;
end;
```

No corpo da função *ComputeJointVelocities* encontramos mais duas funções que houve necessidade de implementar.

A função *checkSingularity* tem como papel a verificação do valor do determinante do jacobiano. Recebe como entrada a matriz do jacobiano e retorna um booleano que indica se a singularidade foi atingida ou não. Se o valor estiver muito próximo de zero, a variável booleana *singularity* é ativada, sinalizando que se atingiu a condição de singularidade. A verificação do valor efetua-se em função de um valor limite de tolerância. Este valor foi arbitrado por tentativa-erro e definido como constante no programa, tendo em atenção o robô conseguir sempre verificar a condição de singularidade a tempo (com tempo suficiente para ser processado pelo tempo do ciclo de controlo de cerca de 40ms).

Um snapshot da função *checkSingularity* abaixo:

```
function checkSingularity(jacobian :matrix): boolean;
begin
if abs(Mdet3x3(Jacobian)) < singularity_tolerance then begin
    singularity := true;
    result := true;
  end
  else begin
    singularity := false;
    result := false;
  end;
end;
```

Verificamos também a função *computeJacobian*, abaixo, que calcula e retorna o jacobiano:

```
function computeJacobian: matrix;
var
  Theta1, Theta2, Theta3: double;
  A01, A12, A23, A02 ,A03: matrix;
  R00,R01,R02 :matrix;
  d00, d01, d02, d03: matrix;
  ZColumnVector: matrix;
  Jv1, Jv2, Jv3: matrix;
  A, B: matrix;
  Jacobian :matrix;
  invertedJacobian :matrix;
begin
  //Z unit vector
  ZColumnVector := Mzeros(3,1);
  Msetv(ZColumnVector, 2, 0, 1);

  //Current angles of the joints
  Theta1 := GetAxisPos(irobot, 0);
  Theta2 := GetAxisPos(irobot, 1);
  Theta3 := GetAxisPos(irobot, 2);

  //Compute the Homogeneous Transformation Matrixes using DH method
  A01 := DHMat(0, -pi*0.5, d1 , Theta1);
  A12 := DHMat(d2, 0, 0, Theta2);
  A23 := DHMat(d3, 0, 0, Theta3);

 //Jacobian columns
 //Jv1
 R00 := Meye(3);
 A02 := MMult(A01,A12);
 A03 := MMult(A02,A23);
 d03 := MCrop(A03, 0, 3, 2, 3);
 d00 := Mzeros(3,1);
 A := MMult(R00, ZColumnVector);
 B := Msub(d03, d00);
 Jv1 := crossProduct(A, B);

 //Jv2
 R01 := Mcrop(A01, 0, 0, 2, 2);
 d01 := MCrop(A01, 0, 3, 2, 3);
 A := Mmult(R01, ZColumnVector);
 B := Msub(d03, d01);
 Jv2 := crossProduct(A, B);

 //Jv3
 R02 := MCrop(A02, 0, 0, 2, 2);
 d02 := MCrop(A02, 0, 3, 2, 3);
 A03 := Mmult(A02, A23);
 d03 := MCrop(A03, 0, 3, 2, 3);
 A := Mmult(R02, ZColumnVector);
 B := Msub(d03,d02);
 Jv3 := crossProduct(A, B);
```

```
  Jacobian := Mzeros(3,3);
  Msetv(Jacobian, 0, 0, Mgetv(Jv1, 0, 0));
  Msetv(Jacobian, 1, 0, Mgetv(Jv1, 1, 0));
  Msetv(Jacobian, 2, 0, Mgetv(Jv1, 2, 0));
  Msetv(Jacobian, 0, 1, Mgetv(Jv2, 0, 0));
  Msetv(Jacobian, 1, 1, Mgetv(Jv2, 1, 0));
  Msetv(Jacobian, 2, 1, Mgetv(Jv2, 2, 0));
  Msetv(Jacobian, 0, 2, Mgetv(Jv3, 0, 0));
  Msetv(Jacobian, 1, 2, Mgetv(Jv3, 1, 0));
  Msetv(Jacobian, 2, 2, Mgetv(Jv3, 2, 0));

  printValue(Mdet3x3(Jacobian),15,1);
  result := Jacobian;
end;
```

O produto cruzado de matrizes, o cálculo do determinante de matrizes de dimensão 3 e uma função de reinicialização do robô para o caso de singularidade foram também implementadas como funções auxiliares (*crossProduct*, *Mdet3x3* e *reset*), representadas abaixo:

```
function crossProduct(Matrix1, Matrix2: matrix): matrix;
var
a,b,c,d,e,f: Double;
crossProductMatrix: matrix;
begin
a := Mgetv(Matrix1,0,0);
b := Mgetv(Matrix1,1,0);
c := Mgetv(Matrix1,2,0);
d := Mgetv(Matrix2,0,0);
e := Mgetv(Matrix2,1,0);
f := Mgetv(Matrix2,2,0);

crossProductMatrix := Mzeros(3,1);
Msetv(crossProductMatrix, 0, 0, b*f - e*c);
Msetv(crossProductMatrix, 1, 0, c*d - a*f);
Msetv(crossProductMatrix, 2, 0, a*e - b*d);

result := crossProductMatrix;
end;
```

```
function Mdet3x3(mat: matrix): double;
var
  a11,a12,a13,a21,a22,a23,a31,a32,a33: double;
begin
  a11 := Mgetv(mat, 0,0);
  a12 := Mgetv(mat, 0,1);
  a13 := Mgetv(mat, 0,2);

  a21 := Mgetv(mat, 1,0);
  a22 := Mgetv(mat, 1,1);
  a23 := Mgetv(mat, 1,2);

  a31 := Mgetv(mat, 2,0);
  a32 := Mgetv(mat, 2,1);
  a33 := Mgetv(mat, 2,2);

  result := (a11*a22*a33 + a12*a23*a31 + a13*a21*a32) - (a31*a22*a13 + a32*a23*a11 + a33*a21*a12);
end;
```

```
//Reset the joints angles to a predefined position under constant values named:
// initial_angle_joint0
// initial_angle_joint1
// initial_angle_joint2
procedure reset;
var initialThetas: matrix;
begin
  initialThetas := Mzeros(3,1);
  Msetv(initialThetas, 0, 0,rad(initial_angle_joint0));
  Msetv(initialThetas, 1, 0,rad(initial_angle_joint1));
  Msetv(initialThetas, 2, 0,rad(initial_angle_joint2));
  SetThetas(initialThetas);
end;
```

Na *sheet* o utilizador define as velocidades lineares desejadas (neste exemplo: V_x = 1cm/s, V_y = -1cm/s, V_z = 0cm/s):

SetVelocity
1
-1
0

Na *sheet* o utilizador observa as velocidades das articulações calculadas pelo método do jacobiano inverso (neste exemplo: v1 = -2.12cm/s, v2 ~ 0cm/s v3, ~ 0cm/s):

Vjoints	
v1:	-2.121938
v2:	-0.000016347€
v3	0.000034041

Para terminar, abaixo mostram-se as constantes definidas para o problema do jacobiano:

- Duration_speed_simulation: a duração do tempo em que se simula a velocidade solicitada pelo utilizador. Neste exemplo são 10 segundos;
- initial_angle_joint0, initial_angle_joint1, initial_angle_joint2: Os ângulos iniciais da configuração do braço quando adota uma posição de repouso. Neste exemplo são 45º na articulação 0, 30º na articulação 1, -40º na articulação 2;
- singularity_tolerance: a tolerância que se atribui ao valor do determinante do jacobiano em que já se considera que se atingiu singularidade;

```
Duration_speed_simulation = 10;
initial_angle_joint0 = 45;
initial_angle_joint1 = 30;
initial_angle_joint2 = -40;
singularity_tolerance = 4e-2;
```

5. Conclusões

Considerou-se que o trabalho que nos foi proposto foi bem conseguido, uma vez que se atingiram todos os objetivos pretendidos.

Foi uma oportunidade de consolidar e de aplicar os conceitos relacionados com as cinemáticas e o jacobiano, já que, através do uso do simulador e da visualização gráfica da implementação, tornou-se clara e intuitiva a teoria por trás dos métodos.