UNICAMP

Universidade Estadual de Campinas

Faculdade de Engenharia Elétrica e de Computação

IA903 – Introdução à Robótica Móvel 19 de setembro de 2024

Docente: Eric Rohmer

Discente:

- Gabriel Toffanetto França da Rocha – 289320

Exercise 1: Kinematics of a single robot leg

Sumário

1

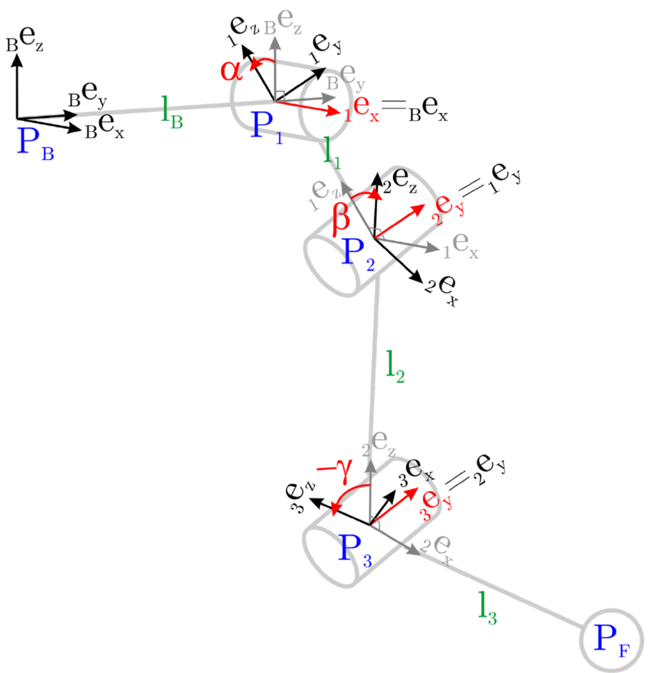

Figura 1: Sistemas de coordenadas da perna robótica.

Uma vez dados os sistemas de coordenadas da perna robótica, exibidos na Figura 1, é possível observar que o *frame* $\{P_1\}$ apresenta rotação em torno do eixo x, o *frame* $\{P_2\}$ em torno do eixo y, assim como o *frame* $\{P_3\}$. Considerando o vetor de coordenadas generalizadas, $\mathbf{q} = [\alpha\ \beta\ \gamma]^T$, sendo α, β e γ respectivamente os ângulos de rotação dos sistemas de coordenadas $\{P_1\}$, $\{P_2\}$ e $\{P_3\}$.

Com isso, têm-se que a rotação entre os sistemas de coordenadas da base e da junta 1 é dado pela matriz (1), a rotação entre o *frame* $\{P_1\}$ e o *frame* $\{P_2\}$ é dada por (2) e a rotação entre o sistema de coordenadas da junta 2 e o sistema da junta 3 é dado pela matriz (3).

$$\mathbf{R}_{B1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \tag{1}$$

$$\mathbf{R}_{12} = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) \\ 0 & 1 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) \end{bmatrix} \tag{2}$$

$$\mathbf{R}_{23} = \begin{bmatrix} \cos(\gamma) & 0 & \sin(\gamma) \\ 0 & 1 & 0 \\ -\sin(\gamma) & 0 & \cos(\gamma) \end{bmatrix} \tag{3}$$

2

Os sistemas de coordenadas $\{P_B\}$, $\{P_1\}$, $\{P_2\}$, $\{P_3\}$ e $\{P_F\}$ da perna robótica estão posicionados entre eles em distâncias l_B, l_1, l_2, l_3 e l_F, respectivamente. Considerando todas essas distâncias iguais e unitárias, obtêm-se os vetores de translação entre os referenciais mostrados em (4), (5), (6) e (7).

$$_B\mathbf{r}_{B1} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix}^T; \tag{4}$$

$$_1\mathbf{r}_{12} = \begin{bmatrix} 0 & 0 & -1 \end{bmatrix}^T; \tag{5}$$

$$_2\mathbf{r}_{23} = \begin{bmatrix} 0 & 0 & -1 \end{bmatrix}^T; \tag{6}$$

$$_3\mathbf{r}_{3F} = \begin{bmatrix} 0 & 0 & -1 \end{bmatrix}^T; \tag{7}$$

Permitindo a transformação entre os sistemas de coordenada, incluindo rotação e translação em uma só operação, a utilização de transformações homogêneas se faz interessante, sendo a mesma enunciada em (8).

$$\mathbf{H} = \begin{bmatrix} & \mathbf{R}_{[3\times3]} & & \mathbf{r}_{[3\times1]} \\ 0 & 0 & 0 & 1 \end{bmatrix}_{[4\times4]} \tag{8}$$

Dessa forma, a transformação homogênea entre o *frame* $\{P_B\}$ e $\{P_1\}$ é dada por (9), entre $\{P_1\}$ e $\{P_2\}$ é dada por (10), entre $\{P_2\}$ e $\{P_3\}$ é dada por (11) e entre $\{P_3\}$ e $\{P_F\}$ é dada por (12),

$$_B\mathbf{H}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 1 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{9}$$

$$_1\mathbf{H}_2 = \begin{bmatrix} \cos(\beta) & 0 & \sin(\beta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\beta) & 0 & \cos(\beta) & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{10}$$

$$_2\mathbf{H}_3 = \begin{bmatrix} \cos(\gamma) & 0 & \sin(\gamma) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\gamma) & 0 & \cos(\gamma) & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{11}$$

$$
{}_3\mathbf{H}_F = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{12}
$$

Para se obter a transformação homogênea entre *frames* de um conjunto consecutivo de transformações de referencial, basta multiplicar as matrizes de transformação homogênea, como feito em (13), obtendo assim a transformação da base para a terceira junta da perna robótica.

$$
{}_B\mathbf{H}_3 \;=\; {}_B\mathbf{H}_1\;{}_1\mathbf{H}_2\;{}_2\mathbf{H}_3 = \begin{bmatrix} [\mathrm{c}(\beta+\gamma) & 0 & \mathrm{s}(\beta+\gamma) & -\mathrm{s}(\beta) \\ \mathrm{s}(\beta+\gamma)\mathrm{s}(\alpha) & \mathrm{c}(\alpha) & -\mathrm{c}(\beta+\gamma)\mathrm{s}(\alpha) & \mathrm{s}(\alpha)+\mathrm{c}(\beta)\mathrm{s}(\alpha)+1 \\ -\mathrm{s}(\beta+\gamma)\mathrm{c}(\alpha) & \mathrm{s}(\alpha) & \mathrm{c}(\beta+\gamma)\mathrm{c}(\alpha) & -\mathrm{c}(\alpha)(\mathrm{c}(\beta)+1) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{13}
$$

$$
\begin{aligned}
{}_B\mathbf{H}_F \;&=\; {}_B\mathbf{H}_1\;{}_1\mathbf{H}_2\;{}_2\mathbf{H}_3\;{}_3\mathbf{H}_F \qquad (14)\\
&= \begin{bmatrix} \mathrm{c}(\beta+\gamma) & 0 & \mathrm{s}(\beta+\gamma) & -\mathrm{s}(\beta+\gamma)-\mathrm{s}(\beta) \\ \mathrm{s}(\beta+\gamma)\mathrm{s}(\alpha) & \mathrm{c}(\alpha) & -\mathrm{c}(\beta+\gamma)\mathrm{s}(\alpha) & \mathrm{s}(\alpha)[1+\mathrm{c}(\beta)+\mathrm{c}(\beta)\mathrm{c}(\gamma)-\mathrm{s}(\beta)\mathrm{s}(\gamma)]+1 \\ -\mathrm{s}(\beta+\gamma)\mathrm{c}(\alpha) & \mathrm{s}(\alpha) & \mathrm{c}(\beta+\gamma)\mathrm{c}(\alpha) & -\mathrm{c}(\alpha)[\mathrm{c}(\beta+\gamma)+\mathrm{c}(\beta)+1] \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
$$

Por meio da transformação de referencial entre o *frame* da base e da ferramenta, (14), é possível expressar o vetor da base para a ferramentam, pré-multiplicando o vetor da origem da base (${}_B\mathbf{r}_{BB} = [0\ 0\ 0]_B^T$) pela transformação homogênea em questão.

$$
{}_B\mathbf{r}_{BF} = \;{}_B\mathbf{H}_F\;{}_B\mathbf{r}_{BB} = \begin{bmatrix} -\sin(\beta+\gamma)-\sin(\beta) \\ \sin(\alpha)[\cos(\beta+\gamma)+\cos(\beta)+1]+1 \\ -\cos(\alpha)[\cos(\beta+\gamma)+cos(\beta)+1] \end{bmatrix} \tag{15}
$$

3

$$
{}_B\mathbf{J}_{BF} = \frac{\partial_B\mathbf{r}_{BF}}{\partial\mathbf{q}} = \begin{bmatrix} \frac{\partial_B\mathbf{r}_{BF}}{\partial q_1} & \frac{\partial_B\mathbf{r}_{BF}}{\partial q_2} & \frac{\partial_B\mathbf{r}_{BF}}{\partial q_3} \end{bmatrix} = \begin{bmatrix} \frac{\partial_B r_{xBF}}{\partial q_1} & \frac{\partial_B r_{xBF}}{\partial q_2} & \frac{\partial_B r_{xBF}}{\partial q_3} \\ \frac{\partial_B r_{yBF}}{\partial q_1} & \frac{\partial_B r_{yBF}}{\partial q_2} & \frac{\partial_B r_{yBF}}{\partial q_3} \\ \frac{\partial_B r_{zBF}}{\partial q_1} & \frac{\partial_B r_{zBF}}{\partial q_2} & \frac{\partial_B r_{zBF}}{\partial q_3} \end{bmatrix} \tag{16}
$$

$$
{}_B\mathbf{J}_{BF} = \begin{bmatrix} 0 & -\mathrm{c}(\beta+\gamma)-\mathrm{c}(\beta) & -\mathrm{c}(\beta+\gamma) \\ \mathrm{c}(\alpha)[\mathrm{c}(\beta+\gamma)+\mathrm{c}(\beta)+1] & \mathrm{s}(\alpha)[-\mathrm{s}(\beta+\gamma)-\mathrm{s}(\beta)] & -\mathrm{s}(\alpha)\mathrm{s}(\beta+\gamma) \\ \mathrm{s}(\alpha)[\mathrm{c}(\beta+\gamma)+\mathrm{c}(\beta)+1] & \mathrm{c}(\alpha)[\mathrm{s}(\beta+\gamma)+\mathrm{s}(\beta)] & \mathrm{c}(\alpha)\mathrm{s}(\beta+\gamma) \end{bmatrix} \tag{17}
$$

Onde $\mathrm{s}(\cdot) = \sin(\cdot)$ e $\mathrm{c}(\cdot) = \cos(\cdot)$.

Considerando uma posição inicial para a perna robótica, (18), e uma velocidade para a ferramenta (19), obtém-se um jacobiano instantâneo igual a (20).

$$\mathbf{q}_i = \begin{bmatrix} 0^\circ & 60^\circ & -120^\circ \end{bmatrix}^T \tag{18}$$

$$\mathbf{v} = \begin{bmatrix} 0 & 0 & -1 \end{bmatrix}^T \text{(m/s)} \tag{19}$$

$$\mathbf{J}_{BF} = \begin{bmatrix} 0 & -1 & -0,5 \\ 2 & 0 & 0 \\ 0 & 0 & 0,866 \end{bmatrix} \tag{20}$$

Com isso, uma vez que o jacobiano relaciona a velocidade no espaço cartesiano com a velocidade no espaço das juntas, $\mathbf{v} = \mathbf{J}\dot{\mathbf{q}}$, pode-se obter a velocidade instantânea das das juntas da perna robótica, $\mathrm{d}\mathbf{q}$, por meio de (21), para a velocidade desejada do efetuador e a posição inicial das juntas.

$$\mathrm{d}\mathbf{q} = \mathbf{J}_{BF}^{+}\mathbf{v} = \begin{bmatrix} 0 \\ -0,5774 \\ 1,1547 \end{bmatrix} \tag{21}$$

4

Para realizar o posicionamento do manipulador em um ponto desejado, deve-se fazer o uso da cinemática inversa, porém, sua obtenção de forma analítica é um procedimento por muitas vezes complexo e massante. Dessa forma, pode-se fazer o uso de métodos numéricos, como o de Newton, para convergir para a posição desejada do efetuador em n passos, até que o erro ζ seja menor que o desejado.

$$\mathbf{q}^{i+1} = \mathbf{q}^{i} + \mathbf{J}^{+}(\mathbf{r}^{goal} - \mathbf{r}^{i}) \tag{22}$$

Por meio do Algoritmo 1, é feita a resolução da cinemática inversa de forma numérica, atualizando a posição da perna robótica a cada interação para acompanhamento da convergência do algoritmo.

Algoritmo 1 Cinemática inversa numérica

$\zeta \leftarrow 0.01$; ▷ Stop error
$\mathbf{q}^i \leftarrow \mathbf{q}^0$;
while $||\mathbf{r}^{goal} - \mathbf{r}^i||_2 \geq \zeta$ **do** ▷ While solution does not converge...
    $\mathbf{q}^{i+1} \leftarrow \mathbf{q}^i + \mathrm{pinv}(\mathbf{J}(\mathbf{q}^i)) * (\mathbf{r}^{goal} - \mathbf{r}^i)$;
    UpdateRobotLegPosition($\mathbf{q}^{i+1}$);
    $\mathbf{q}^i \leftarrow \mathbf{q}^{i+1}$;
end while
$\mathbf{q}^{goal} \leftarrow \mathbf{q}^i$ ▷ Generalized coordinates for the solution

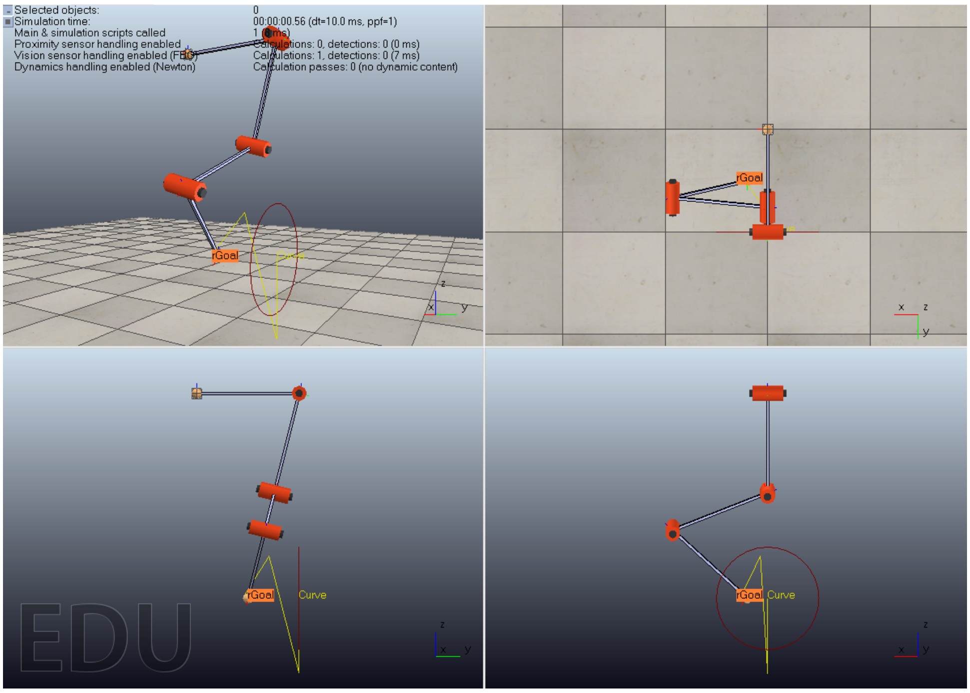

Figura 2: Posicionamento da perna robótica em uma posição desejada por meio da cinemática inversa numérica.

A convergência se deu em três passos, como mostrado no *log* abaixo.

```
i = 1 | rGoal = [0.200 0.500 -2.000]; r = [0.000 1.000 -2.732]; error = 0.478
i = 2 | rGoal = [0.200 0.500 -2.000]; r = [0.071 0.706 -1.589]; error = 0.072
i = 3 | rGoal = [0.200 0.500 -2.000]; r = [0.249 0.476 -1.953]; error = 0.003
```

5

Uma vez dada os *waypoints* da trajetória a ser seguida pela perna robótica, faz-se o uso novamente da relação (21) para obtenção da velocidade dos atuadores. Porém, dessa vez $\mathbf{v}$ será dado com base no erro de posição entre o efetuador da perna robótica e o ponto da trajetória atual, multiplicado por um ganho proporcional, como em (23).

$$\mathbf{v} = K_p(\mathbf{r}^{goal} - \mathbf{r}^i) \tag{23}$$

Considerando um ganho proporcional igual $K_p = 10$, obteve-se um bom rastreio da trajetória, com um transitório consideravelmente curto, e boa convergência ao *setpoint*, como pode ser visto nas Figuras 3 e 4.

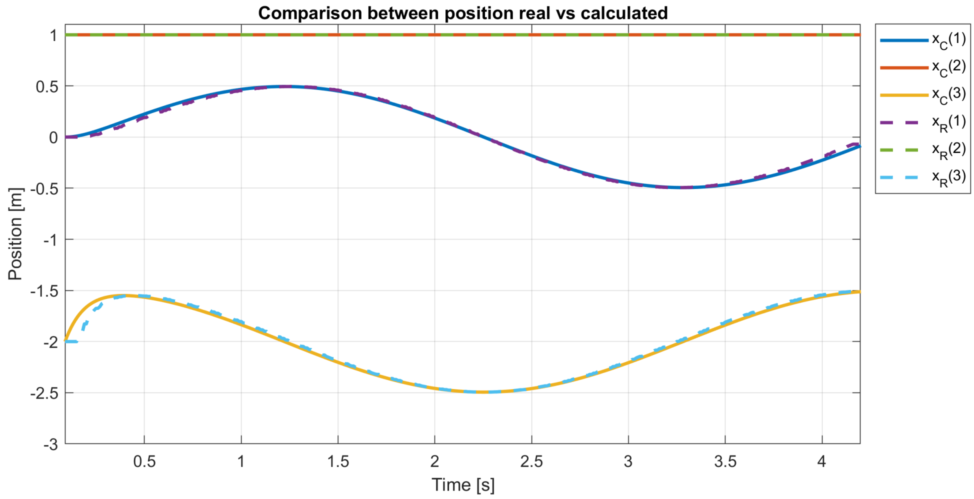

Figura 3: Rastreio da posição do efetuador da perna robótica em x, y e z.

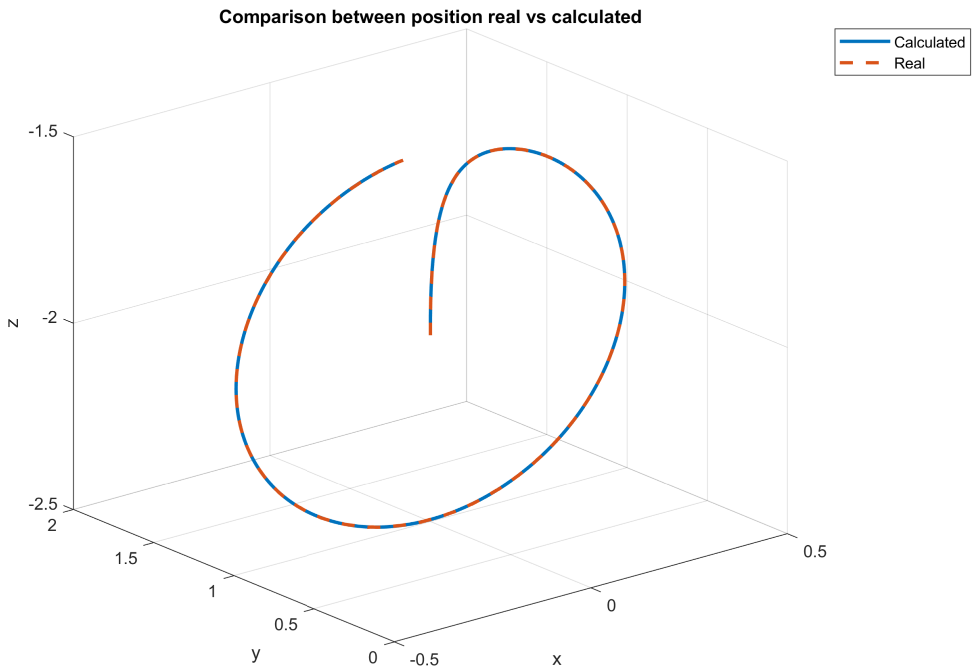

Figura 4: Rastreio da posição do efetuador da perna robótica no espaço.

Por fim, é possível observar na Figura 5 a trajetória realizada pela perna robótica partindo da posição inicial no centro do círculo.

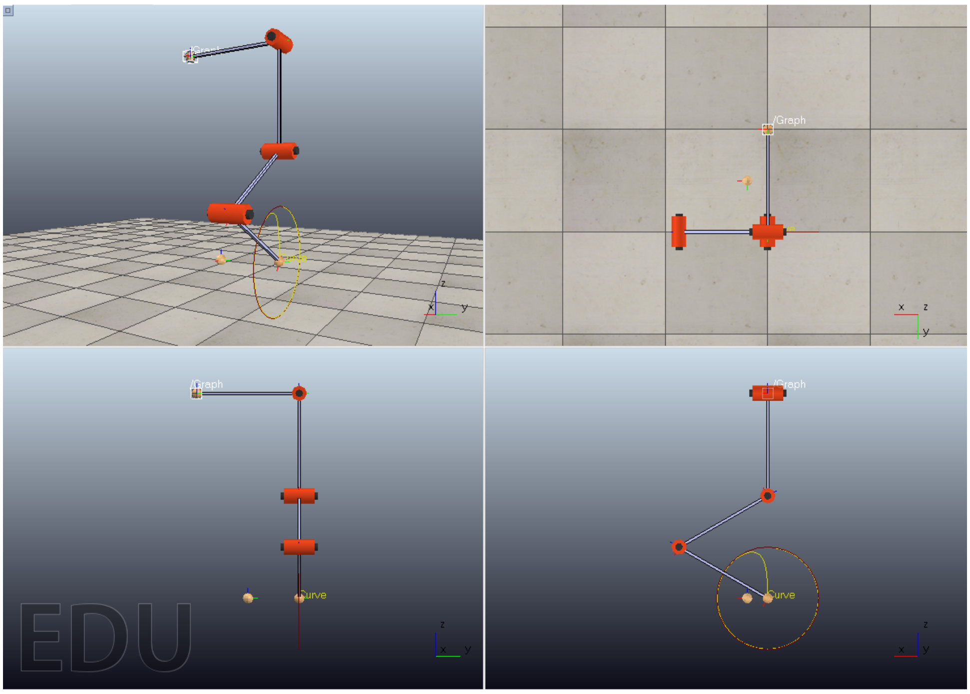

Figura 5: Trajetória realizada pela perna robótica.

Apêndices

Código fonte – Exercício 1

```
clc; clear all;

syms alpha beta gamma real

% write down the rotation matrices using the symbolic parameters alpha, beta
    , gamma

% Rotation arround 1_e_x
R_B1 = [1 0           0; ...
        0 cos(alpha) -sin(alpha); ...
        0 sin(alpha) cos(alpha) ;];

% Rotation arround 2_e_y
R_12 = [cos(beta)  0 sin(beta); ...
        0          1 0; ...
        -sin(beta) 0 cos(beta);];

% Rotation arround 3_e_y
R_23 = [cos(gamma)  0 sin(gamma); ...
        0           1 0; ...
        -sin(gamma) 0 cos(gamma);];

% answers
valid
```

Código fonte – Exercício 2

```
clc; clear all

syms alpha beta gamma

lb = 1; l1 = 1; l2 = 1; l3 = 1;

% rotational matrices calculated in previous problem set
R_B1 = [1,0,0;0,cos(alpha),-sin(alpha);0,sin(alpha),cos(alpha)];
R_12 = [cos(beta),0,sin(beta);0,1,0;-sin(beta),0,cos(beta)];
R_23 = [cos(gamma),0,sin(gamma);0,1,0;-sin(gamma),0,cos(gamma)];

% write down the 3x1 relative position vectors for link length l_i=1
r_B1_inB = [0; 1; 0];
r_12_in1 = [0; 0; -1];
r_23_in2 = [0; 0; -1];
r_3F_in3 = [0; 0; -1];

% write down the homogeneous transformation matrices
H_B1 = [R_B1 r_B1_inB ; [0 0 0 1]]
H_12 = [R_12 r_12_in1 ; [0 0 0 1]]
H_23 = [R_23 r_23_in2 ; [0 0 0 1]]
H_3F = [eye(3) r_3F_in3 ; [0 0 0 1]]

% create the cumulative transformation matrix
H_B3 = H_B1*H_12*H_23*H_3F

% find the foot point position vector
r_BF_inB = H_B3*[0; 0; 0; 1]
r_BF_inB = r_BF_inB(1:3)

valid
```

Código fonte – Exercício 3

```
clc; clear all;

syms alpha beta gamma real

q = [alpha;beta;gamma];

r_BF_inB = [...
    - sin(beta + gamma) - sin(beta);...
  sin(alpha)*(cos(beta + gamma) + cos(beta) + 1) + 1;...
  -cos(alpha)*(cos(beta + gamma) + cos(beta) + 1)]


% determine the foot point Jacobian J_BF_inB=d(r_BF_inB)/dq
dr_dalpha = [0; ...
             cos(alpha)*(cos(beta + gamma) + cos(beta) + 1); ...
             sin(alpha)*(cos(beta + gamma) + cos(beta) + 1)];

dr_dbeta = [-cos(beta + gamma) + cos(beta); ...
            sin(alpha)*(-sin(beta + gamma) -sin(beta)); ...
            cos(alpha)*(+sin(beta + gamma) +sin(beta))];

dr_dgamma = [-cos(beta + gamma); ...
             -sin(alpha)*sin(beta + gamma); ...
             cos(alpha)*sin(beta + gamma)];

J_BF_inB = [dr_dalpha dr_dbeta dr_dgamma]


% what generalized velocity dq do you have to apply in a configuration q = [0;60;-120]
% to lift the foot in vertical direction with v = [0;0;-1m/s];
v = [0; 0; -1]
qi = [0; 60*(pi/180); -120*(pi/180)];

% Determine the numerical value of the foot point jacobian for initial joint angles qi

alpha = qi(1);
beta = qi(2);
gamma = qi(3);

dr_dalpha = [0; ...
             cos(alpha)*(cos(beta + gamma) + cos(beta) + 1); ...
             sin(alpha)*(cos(beta + gamma) + cos(beta) + 1)];

dr_dbeta = [-cos(beta + gamma) - cos(beta); ...
            sin(alpha)*(-sin(beta + gamma) -sin(beta)); ...
```

```
            cos(alpha)*(sin(beta + gamma) +sin(beta))];

dr_dgamma = [-cos(beta + gamma); ...
             -sin(alpha)*sin(beta + gamma); ...
             cos(alpha)*sin(beta + gamma)];

JBF = [dr_dalpha dr_dbeta dr_dgamma]

% Determine the numerical value for dq
dq = inv(JBF)*v

valid
```

Código fonte – Exercício 4

```
% Make sure to have the simulation scene mooc_exercise.ttt running in V-REP!

% simulation setup, will add the matlab paths
connection = simulation_setup();

% the robot we want to interact with
robotNb = 0;

% open the connection
connection = simulation_openConnection(connection, robotNb);

% start simulation if not already started
simulation_start(connection);

vrep=connection.vrep;
% initialize connection
[err dt]=vrep.simxGetFloatingParameter(connection.clientID,vrep.sim_floatparam_simulation_time_step,vrep.simx_opmode_oneshot_wait);

% now enable stepped simulation mode:
simulation_setStepped(connection,true);

% given are the functions
%   r_BF_inB(alpha,beta,gamma) and
%   J_BF_inB(alpha,beta,gamma)
% for the foot positon respectively Jacobian

r_BF_inB = @(alpha,beta,gamma)[...
    -sin(beta + gamma) - sin(beta);...
  sin(alpha)*(cos(beta + gamma) + cos(beta) + 1) + 1;...
  -cos(alpha)*(cos(beta + gamma) + cos(beta) + 1)];

J_BF_inB = @(alpha,beta,gamma)[...
                                              0,               - cos(beta + gamma) - cos(beta),             -cos(beta + gamma);...
 cos(alpha)*(cos(beta + gamma) + cos(beta) + 1), -sin(alpha)*(sin(beta + gamma) + sin(beta)), -sin(beta + gamma)*sin(alpha);...
 sin(alpha)*(cos(beta + gamma) + cos(beta) + 1),  cos(alpha)*(sin(beta + gamma) + sin(beta)),  sin(beta + gamma)*cos(alpha)];

% write an algorithm for the inverse kinematics problem to
% find the generalized coordinates q that gives the endeffector position rGoal =
% [0.2,0.5,-2]' and store it in qGoal
q0 = pi/180*([0,-30,60])';
updatePos(vrep,connection.clientID,q0)
pause(0.5)
```

```

rGoal = [0.2,0.5,-2]';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% enter here your algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

k = 0;

zeta = 0.01; % Convergence error

while(norm(rGoal - r_BF_inB(q0(1),q0(2),q0(3))) > zeta)

    r = r_BF_inB(q0(1),q0(2),q0(3));

    qGoal = q0 + inv(J_BF_inB(q0(1),q0(2),q0(3)))*(rGoal - r);

    updatePos(vrep,connection.clientID,qGoal)
    pause(0.2)
    q0 = qGoal;
    k = k+1;

    fprintf('i = %d | rGoal = [%0.3f %0.3f %0.3f]; r = [%0.3f %0.3f %0.3f]; error = %0.3f \n', ...
            k, rGoal(1), rGoal(2), rGoal(3), r(1), r(2), r(3), norm(rGoal - r_BF_inB(q0(1),q0(2),q0(3))))

end

% now disable stepped simulation mode:
simulation_setStepped(connection,false);


pause(5)

% stop the simulation
simulation_stop(connection);

% close the connection
simulation_closeConnection(connection);
```

Código fonte – Exercício 5

```
dqMatrix = [];

% Make sure to have the simulation scene mooc_exercise.ttt running in V-REP!

% simulation setup, will add the matlab paths
connection = simulation_setup();

% the robot we want to interact with
robotNb = 0;

% open the connection
connection = simulation_openConnection(connection, robotNb);

% start simulation if not already started
simulation_start(connection);

vrep=connection.vrep;

% initialize connection
% do a function simulation_getDt to do the following
dt= simulation_getDt(connection)

% now enable stepped simulation mode:
simulation_setStepped(connection,true);

    % given are the functions
    %   r_BF_inB(alpha,beta,gamma) and
    %   J_BF_inB(alpha,beta,gamma)
    % for the foot positon respectively Jacobian

    r_BF_inB = @(alpha,beta,gamma)[...
        - sin(beta + gamma) - sin(beta);...
        sin(alpha)*(cos(beta + gamma) + cos(beta) + 1) + 1;...
        -cos(alpha)*(cos(beta + gamma) + cos(beta) + 1)];

    J_BF_inB = @(alpha,beta,gamma)[...
        0,              - cos(beta + gamma) - cos(beta),            -cos(beta + gamma);...
        cos(alpha)*(cos(beta + gamma) + cos(beta) + 1), -sin(alpha)*(sin(beta + gamma) + sin(beta)), -sin(beta + gamma)*sin(alpha);...
        sin(alpha)*(cos(beta + gamma) + cos(beta) + 1),  cos(alpha)*(sin(beta + gamma) + sin(beta)),  sin(beta + gamma)*cos(alpha)];

    % write an algorithm for the inverse differntial kinematics problem to
    % find the generalized velocities dq to follow a circle in the body xz plane
    % around the start point rCenter with a radius of r=0.5 and a
```

```
    % frequeny of 1Hz. The start configuration is q =  pi/180*([0,-60,120])'
    q0 = pi/180*([0,-60,120])';
    %q0 = pi/180*([0,-80,140])';

    updatePos(vrep,connection.clientID,q0)
   % pause(1.0)

    dq0 = zeros(3,1);
    rCenter = r_BF_inB(q0(1),q0(2),q0(3));
    radius = 0.5;
    f = 0.25;
    rGoal = @(t) rCenter + radius*[sin(2*pi*f*t),0,cos(2*pi*f*t)]';
    drGoal = @(t) 2*pi*f*radius*[cos(2*pi*f*t),0,-sin(2*pi*f*t)]';

    % define here the time resolution
    deltaT = dt;%0.01;
    timeArr = 0:deltaT:1/f;

    % q, r, and rGoal are stored for every point in time in the following arrays
    qArr = zeros(3,length(timeArr));
    rArr = zeros(3,length(timeArr));
    rGoalArr = zeros(3,length(timeArr));

    q = q0;
    dq = dq0;

    kp = 10;

    for i=1:length(timeArr)
        t = timeArr(i);
        % data logging, don't change this!
        q = q+deltaT*dq;
        qArr(:,i) = q;
        rArr(:,i) = r_BF_inB(q(1),q(2),q(3));
        rGoalArr(:,i) = rGoal(t);

        % controller:
        % step 1: create a simple p controller to determine the desired foot
        % point velocity
        v = kp*(rGoalArr(:,i) - rArr(:,i));
        % step 2: perform inverse differential kinematics to calculate the
        % gneralized velocities
        dq = inv(J_BF_inB(q(1),q(2),q(3)))*v;

        updateVels(vrep,connection.clientID,dq,q)

    end

```

```

% now disable stepped simulation mode:
simulation_setStepped(connection,false);

% stop the simulation
simulation_stop(connection);

% close the connection
simulation_closeConnection(connection);



%% Set up the Import Options and import the data
opts = delimitedTextImportOptions("NumVariables", 4);

% Specify range and delimiter
opts.DataLines = [3, Inf];
opts.Delimiter = ",";

% Specify column names and types
opts.VariableNames = ["Times", "DataMeters", "Data0Meters", "Data1Meters"];
opts.VariableTypes = ["double", "double", "double", "double"];

% Specify file level properties
opts.ExtraColumnsRule = "ignore";
opts.EmptyLineRule = "read";

% Import the data
coppeliagraph = readtable("./coppelia_graph.csv", opts);

%% Convert to output type
coppeliagraph = table2array(coppeliagraph);

%% Clear temporary variables
clear opts

%% Plot

close all

n = linspace(coppeliagraph(1244,1),coppeliagraph(end-1,1),401);

figure(1)
plot(n,rArr(1,:),'Linewidth',2)
hold on
plot(n,rArr(2,:),'Linewidth',2)
plot(n,rArr(3,:),'Linewidth',2)
plot(coppeliagraph(1244:end-1,1),coppeliagraph(1244:end-1,2),'--','Linewidth
    ',2)
```

```
plot(coppeliagraph(1244:end-1,1),coppeliagraph(1244:end-1,3),'--','Linewidth
    ',2)
plot(coppeliagraph(1244:end-1,1),coppeliagraph(1244:end-1,4),'--','Linewidth
    ',2)
axis([0.09 4.2 -3 1.1])
xlabel('Time [s]')
ylabel('Position [m]')
title('Comparison between position real vs calculated')
grid on
legend('x_C(1)','y_C(2)','z_C(3)','x_R(1)','y_R(2)','z_R(3)','Location','
    northeastoutside')
hold off

n = length(coppeliagraph(1244:end-1,3));
m = length(rArr(1,:));

x = interp1(linspace(1,n,m)',rArr(1,:),(1:n)');
y = interp1(linspace(1,n,m)',rArr(2,:),(1:n)');
z = interp1(linspace(1,n,m)',rArr(3,:),(1:n)');

figure(1)
plot3(x,y,z,coppeliagraph(1244:end-1,2),coppeliagraph(1244:end-1,3),
    coppeliagraph(1244:end-1,4),'--','Linewidth',2)
hold on
xlabel('x')
ylabel('y')
zlabel('z')
title('Comparison between position real vs calculated')
grid on
legend('Calculated','Real')
hold off

% exportgraphics(gca,'effector_position.png','Resolution',300)
```