Universidade Estadual de Campinas
UNICAMP

IA368N

Atividade 1

Robotic Leg

Vinicius Rodrigues Sanches – RA: 208976

Introdução Teórica

O sistema de locomoção robótica feita através de sistema de pernas apresenta uma série de vantagens quando o terreno em que se locomove o robô é acidentado. Copiando os sistemas biológicos naturais (os seres vivos) tentamos reproduzir suas capacidades de locomoção por oferecer vários graus de liberdade e facilidade, já que o membro manipulado precisa se apoiar em um único ponto no terreno. Claro que essas vantagens trazem algumas desvantagens por dificultar a construção com muitos sensores e atuadores bem como uma certa dificuldade de encontrar o ponto de equilíbrio do sistema dependendo de quantos atuadores e graus de liberdade estes apresentam. Em alguns casos fica impossível o cálculo de todas as possibilidades de movimento dos atuadores em tempo hábil, o que nos força a adotar solução probabilísticas e muitas vezes imperfeitas. Existem outras considerações a serem feitas também em relação à sua eficiência energética.

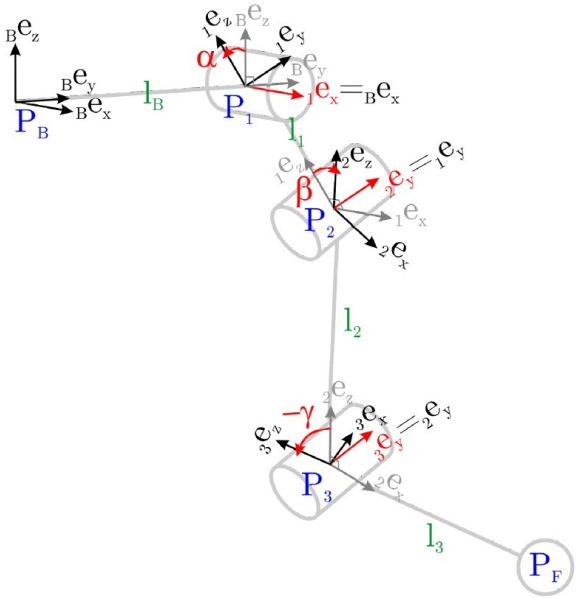
Objetivo

Nosso objetivo é demonstrar, por meio dos experimentos feitos em ambiente simulado, alguns dos cálculos e transformações necessárias ao uso de membros manipuladores e demonstrá-los.

Introdução às Atividades subsequentes.

Nesse exercício vamos executar algumas transformações de coordenadas generalizadas, rotações, translações bem como o Jacobiano do ponto de apoio P_f para observar a influência de cada variação angular do sistema como um todo. Tudo isso consiste em uma análise cinemática do sistema.

Vemos abaixo a figura mostrando o sistema do nosso manipulador, no caso, uma perna com três graus de liberdade.



Matrizes de Rotação Relativas

Dada a descrição cinemática de uma única perna com três graus de liberdade (q:= [α β γ]^T) determinamos as matrizes de rotação relativa RAC rodando um vetor r de um sistema arbitrário de coordenadas C a A:

$$ {}_A r = R_{AC}\, {}_C r $$

Tarefa: Como uma função das coordenadas generalizadas alfa, beta, gama, quais são as três matrizes de rotação relativas? Edite ex01.m de tal maneira que ele compute as matrizes de rotação relativas.

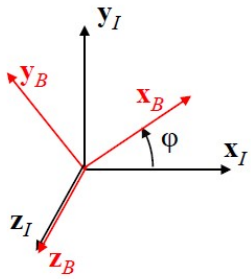
Seguem as equação para matrizes de rotação:

$$ \mathbf{R}_{IB} = \begin{bmatrix} {}_I\mathbf{x}_B & {}_I\mathbf{y}_B & {}_I\mathbf{z}_B \\ \cos(\varphi) & -\sin(\varphi) & 0 \\ \sin(\varphi) & \cos(\varphi) & 0 \\ 0 & 0 & 1 \end{bmatrix} $$

The x-axis of coordinate system B expressed in I

The y-axis of coordinate system B expressed in I

Rotational axis



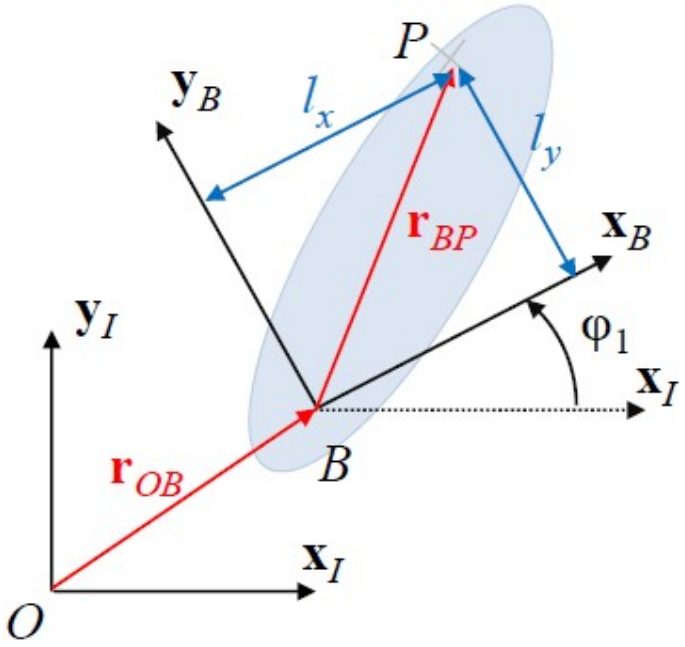
$$ {}_I\mathbf{r} = \mathbf{R}_{IB}\, {}_B\mathbf{r}, \quad {}_B\mathbf{r} = \mathbf{R}_{BI}\, {}_I\mathbf{r}, \; \mathbf{R}_{BI} = \mathbf{R}_{IB}^T $$

Transferindo para nosso manipulador, temos as três matrizes correspondentes aos três graus de liberdade no código fonte de solução:

```
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)]]
;
```

Matrizes de Transformação Homogêneas

Como já calculamos no tópico anterior as matrizes de rotação e agora escolhendo o comprimento de todos os seguimentos, nós determinaremos a matriz de transformação homogênea que transforma o ponto do pé representado no frame de coordenadas de l3 para o frame B (frame inicial do sistema). Para isso precisamos combinar a equação matricial de translação abaixo com as matrizes de rotação previamente resolvidas.



$$ {}_I\mathbf{r}_{OP} = {}_I\mathbf{r}_{OB} + {}_I\mathbf{r}_{BP} = {}_I\mathbf{r}_{OB} + \mathbf{R}_{IB}\,{}_B\mathbf{r}_{BP} $$

Matriz de Rotação

$$ \begin{pmatrix} {}_I\mathbf{r}_{OP} \\ 1 \end{pmatrix} = \begin{bmatrix} \mathbf{R}_{IB} & {}_I\mathbf{r}_{OB} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} {}_B\mathbf{r}_{BP} \\ 1 \end{pmatrix} $$

Vetor intermediário do próprio sistema

Tendo as matrizes de rotação previamente calculadas:

```
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)]]
;
```

Indicamos as matrizes dos vetores em relação ao seu próprio sistema de coordenadas:

```
r_B1_inB = [0; lb; 0];  %→ referente ao lb
r_12_in1 = [0; 0; -l1]; %→referente ao l1
r_23_in2 = [0; 0; -l2]; %→referente ao l2
r_3F_in3 = [0; 0; -l3]; %→ referente o l3
```

Agora temos que produzir a matriz homogênea para os 3 vetores intermediários e a cumulativa (composição de rotações) para o vetor final como o código abaixo:

```
% 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]];

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

Como obtivemos a matriz acumulativa H_B3, agora podemos ter o valor da posição do ponto final de P_f em relação ao frame inicial B.

Utilizando a mesma matriz de solução temos a equação final:

```
% find the foot point position vector
r_BF_inB = H_B3(1:3,:)*[r_3F_in3;1];
```

Jacobianos e Cinemática diferencial

Temos que calcular as velocidades e o Jacobiano do ponto final como também a velocidade generalizada para um movimento cartesiano desejado.

Como já nos foi dado a equação e seus coordenadas:

```
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)];
```

Apenas temos que calcular o seu Jacobiano em relação aos ângulos de q:

```
% determine the foot point Jacobian J_BF_inB=d(r_BF_inB)/dq
J_BF_inB = [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)];
```

Agora podemos inicializar os valores iniciais dos ângulos q_i e calcular o jacobiano pseudo-invertido:

```
% 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
JBF = double(subs(J_BF_inB,[alpha beta gamma],qi'));
pseudo_inverted_jacobian =  inv(JBF'*JBF)*JBF';
```

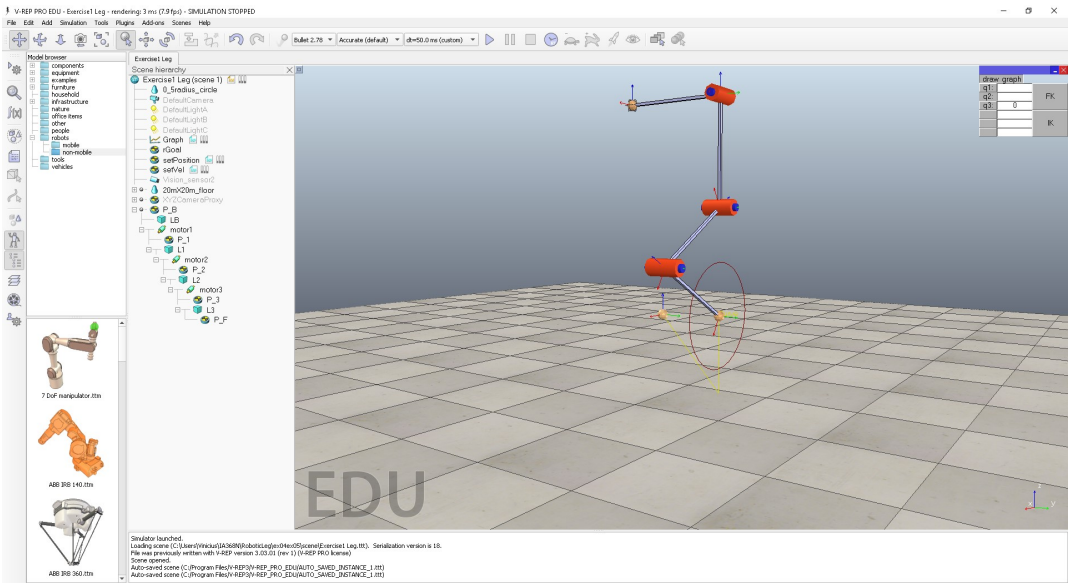
Para determinarmos o vetor de velocidades angulares dq é só multiplicarmos o Jacobiano pseudo-invertido pelo vetor de velocidades cartesianas v:

$$\dot{\mathbf{q}} = \mathbf{J}^{+}\dot{\mathbf{r}}_{F}^{goal}$$

```
% Determine the numerical value for dq
dq = pseudo_inverted_jacobian*v;
```

Cinemática Inversa Numérica

Vamos implementar uma solução para solução de cinemática inversa usando métodos numéricos (Método de Newton) para determinar a configuração do objetivo $\mathbf{q_{goal}}$.Utilizando o Octave e o V_REP, temos que fazer o ponto P_f chegar ao objeto **rGoal.**



Como já temos todas as equações generalizadas e seus Jacobianos como mostrado no código abaixo:

```
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)];
```

Primeiramente atualizamos a posição inicial dos ângulos da perna para encontrar a posição inicial:

```
q0 = pi/180*([0,-30,60])';
updatePos(vrep,connection.clientID,q0)
```

Escolhemos nosso rGoal :

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

Agora usamos um método iterativo para buscar um valor para todos os ângulos alpha beta e gamma:

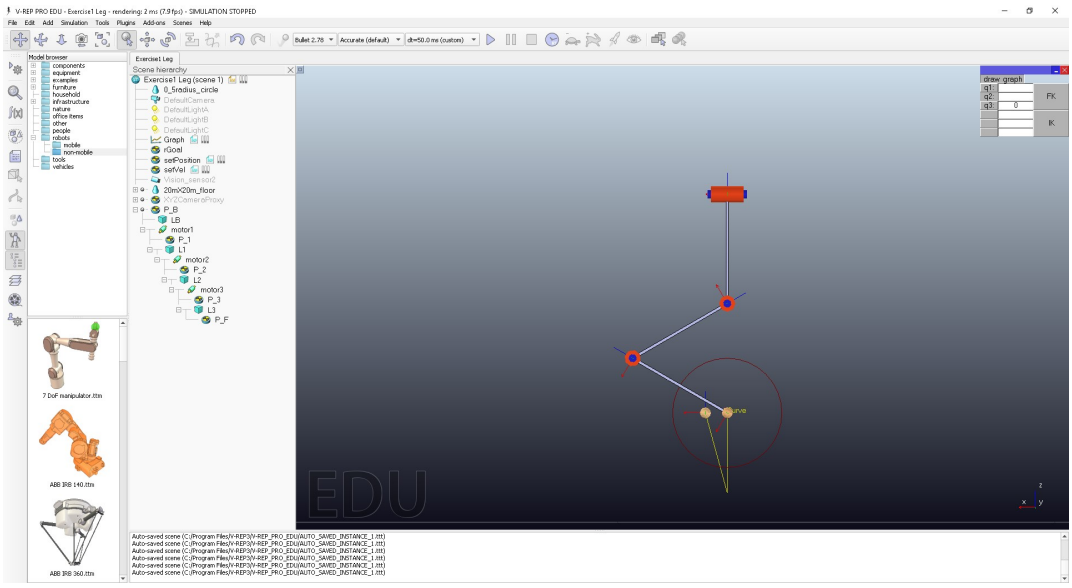
$$\mathbf{q}^{i+1} = \mathbf{q}^{i} + \mathbf{J}^{+}(\mathbf{r}^{goal} - \mathbf{r}^{i})$$

```
qi = q0;
error_valid = 1;

while error_valid >= 1E-8
qGoal = qi + pinv(J_BF_inB(qi(1),qi(2),qi(3)))*(rGoal-
r_BF_inB(qi(1),qi(2),qi(3)));
qi = qGoal;
valid
end

updatePos(vrep,connection.clientID,qGoal)
```

A função *valid* verifica e modifica o erro válido em relação ao rGoal para determinar a parada. Após isso podemos atuar nos motores da perna robótica com o valor correto dos ângulos.

Controle de Trajetória usando Cinemática diferencial inversa

Vamos implementar um controlador proporcional simples para determinar a velocidade cartesiana do ponto do pé e aplicar cinemática diferencial inversa para determinar as velocidades das juntas. Nesse caso nosso ponto de partida é o centro e o ponto do pé vai fazer uma trajetória circular partindo do centro da circunferência.



Definimos nosso ponto inicial velocidades inicial frequencia de atualização e equações do circulos (rGoal e drGoal) e suas velocidades:

```
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)]';
```

Definimos as equações de resolução de tempo e inicializamos os vetores que vao guardar os valores dos pontos calculados:

```
     % 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));
```

Agora inicializamos as velocidades das juntas e os parâmetros iniciais para os cálculos. Vamos iterar ponto a ponto nesse processo, encontrando sempre o próximo ponto e a próxima velocidade das juntas que deveram ser aplicadas:

```
q = q0;
    dq = dq0;
    updateVels(vrep,connection.clientID,dq)
    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 = drGoal(t)+10.0*(rGoal(t)-rArr(:,i));
        % step 2: perform inverse differential kinematics to calculate the
        % gneralized velocities
        dq = J_BF_inB(q(1),q(2),q(3))\v;
        dqMatrix=[dqMatrix dq];
        %updatePos(vrep,connection.clientID,q)
        updateVels(vrep,connection.clientID,dq)
    end
```

Podemos verificar que para cada fração de tempo i calculamos o próximo vetor de ângulos q usando o deltaT multiplicando pelo dq calculado;

```
t = timeArr(i);
% data logging, don't change this!
q = q+deltaT*dq;
```

Para tanto, temos que sempre recalcular a velocidade cartesiana v usando drGoal , rArr e rGoal:

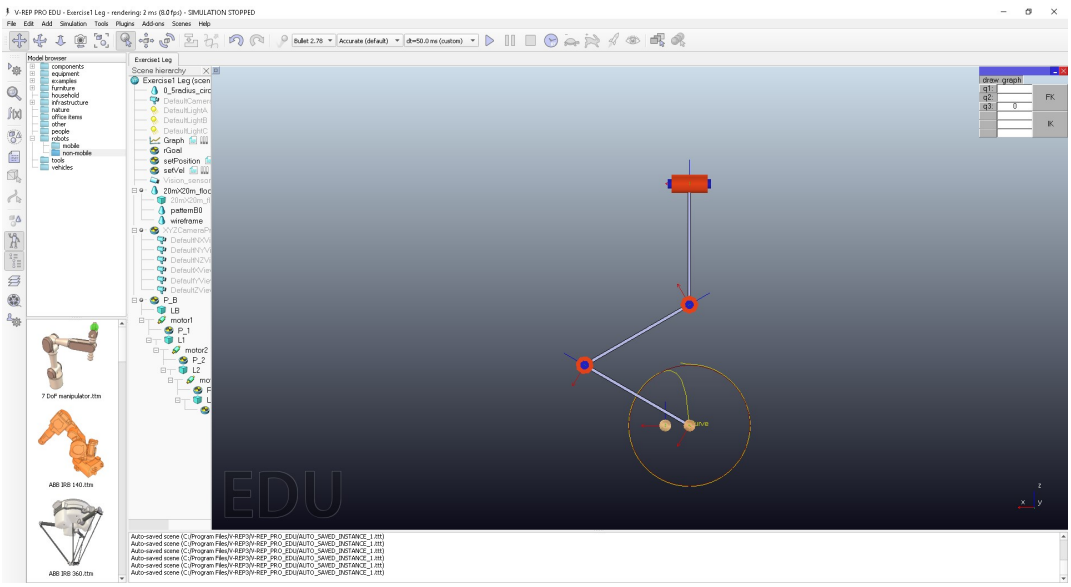
```
qArr(:,i) = q;
rArr(:,i) = r_BF_inB(q(1),q(2),q(3));
rGoalArr(:,i) = rGoal(t);

% controller:
% step 1: create a simple p controller to determine the desired foot
% point velocity
v = drGoal(t)+10.0*(rGoal(t)-rArr(:,i));
```

Podemos notar que rArr é sempre o ponto que conseguimos chegar aplicando a função de cinemática direta usando os ângulos calculados.

Ao final disso temos que recalcular o dq usando o Jacobiano invertido multiplicado pela velocidade cartesiana que acabamos de calcular, atualizar a dqMatrix e atuar nas velocidades das juntas:

```
% step 2: perform inverse differential kinematics to calculate the
 % gneralized velocities
 dq = J_BF_inB(q(1),q(2),q(3))\v;
 dqMatrix=[dqMatrix dq];
 %updatePos(vrep,connection.clientID,q)
 updateVels(vrep,connection.clientID,dq)
```

Podemos observar um movimento suave tentando acompanhar a circunferência descrita. Claramente que essa trajetória não se faz perfeitamente circular como podemos observar na figura abaixo. Mas tem uma resolução satisfatória.



Vamos agora comparar a circunferência ideal e os pontos realmente atingidos.

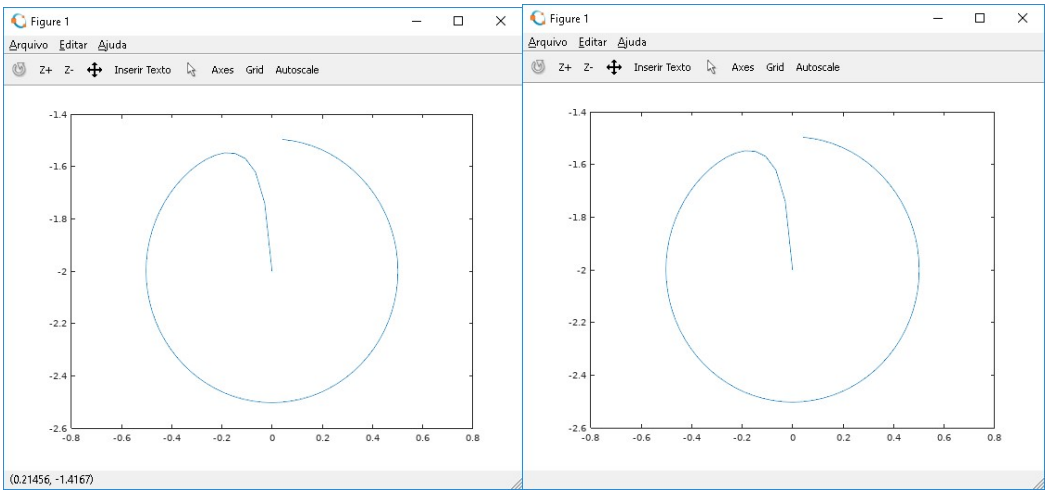
Exortando as curvas geradas por um arquivo CSV, temos 80 leituras (no csv temos um pouco mais devido atraso de inicio e de finalização do processo – removendo esses pontos podemos melhorar).

Differ 1= rArr' – positions_csv:

temos aqui os cálculos das diferenças do vetor r calculado e das posições que foram alcançadas na verdade:

```
-2.0000e-007  0.0000e+000  0.0000e+000
-3.2463e-007  0.0000e+000  2.7203e-007
-2.0884e-007  0.0000e+000  4.9734e-007
-1.8306e-007  0.0000e+000  -3.9915e-007
-3.4399e-007  0.0000e+000  -5.9460e-007
-6.3552e-007  0.0000e+000  3.4059e-007
-4.3832e-007  0.0000e+000  2.2530e-007
1.5992e-007  0.0000e+000  -9.9426e-008
-5.1137e-007  0.0000e+000  -1.9305e-007
-7.4722e-007  0.0000e+000  -1.0408e-007
-5.8746e-007  0.0000e+000  -5.5262e-007
1.3783e-007  0.0000e+000  1.8365e-007
6.0913e-008  0.0000e+000  3.5316e-008
-2.3936e-007  0.0000e+000  1.0345e-007
-3.6296e-007  0.0000e+000  -2.9958e-007
-1.6125e-007  0.0000e+000  4.1650e-007
2.1022e-007  0.0000e+000  -5.6194e-007
-5.0298e-007  0.0000e+000  2.8931e-007
-2.3304e-004  0.0000e+000  1.3579e-004
-2.3416e-004  0.0000e+000  1.3296e-004
-2.3475e-004  0.0000e+000  1.3224e-004
-2.3538e-004  0.0000e+000  1.3070e-004
-2.3615e-004  0.0000e+000  1.2979e-004
-2.3613e-004  0.0000e+000  1.3017e-004
-2.3674e-004  0.0000e+000  1.3063e-004
-2.3606e-004  0.0000e+000  1.3040e-004
-2.3543e-004  0.0000e+000  1.3125e-004
-2.3503e-004  0.0000e+000  1.3278e-004
-2.3446e-004  0.0000e+000  1.3449e-004
-2.3291e-004  0.0000e+000  1.3639e-004
-2.3207e-004  0.0000e+000  1.3796e-004
-2.3047e-004  0.0000e+000  1.4107e-004
-2.2868e-004  0.0000e+000  1.4423e-004
-2.2591e-004  0.0000e+000  1.4735e-004
-2.2435e-004  0.0000e+000  1.5086e-004
-2.2052e-004  0.0000e+000  1.5448e-004
-2.1809e-004  0.0000e+000  1.5822e-004
-2.1448e-004  0.0000e+000  1.6248e-004
-2.1145e-004  0.0000e+000  1.6801e-004
-2.0719e-004  0.0000e+000  1.7297e-004
-2.0295e-004  0.0000e+000  1.7817e-004
-1.9791e-004  0.0000e+000  1.8314e-004
-1.9258e-004  0.0000e+000  1.8859e-004
-1.8687e-004  0.0000e+000  1.9514e-004
-1.8078e-004  0.0000e+000  2.0010e-004
-1.7342e-004  0.0000e+000  2.0592e-004
-1.6722e-004  0.0000e+000  2.1148e-004
-1.5900e-004  0.0000e+000  2.1774e-004
-1.5117e-004  0.0000e+000  2.2348e-004
-1.4331e-004  0.0000e+000  2.2880e-004
-1.3423e-004  0.0000e+000  2.3404e-004
-1.2525e-004  0.0000e+000  2.3895e-004
```

```
-1.1549e-004  0.0000e+000  2.4398e-004
-1.0630e-004  0.0000e+000  2.4761e-004
-9.5588e-005  0.0000e+000  2.5220e-004
-8.5859e-005  0.0000e+000  2.5570e-004
-7.6148e-005  0.0000e+000  2.5885e-004
-6.5395e-005  0.0000e+000  2.6163e-004
-5.4720e-005  0.0000e+000  2.6388e-004
-4.4632e-005  0.0000e+000  2.6656e-004
-3.5445e-005  0.0000e+000  2.6757e-004
-2.4753e-005  0.0000e+000  2.6884e-004
-1.5721e-005  0.0000e+000  2.6932e-004
-6.8518e-006  0.0000e+000  2.6959e-004
1.5746e-006  0.0000e+000  2.7009e-004
1.0039e-005  0.0000e+000  2.6952e-004
1.6963e-005  0.0000e+000  2.6956e-004
2.2363e-005  0.0000e+000  2.6867e-004
2.7745e-005  0.0000e+000  2.6797e-004
3.2236e-005  0.0000e+000  2.6817e-004
3.3839e-005  0.0000e+000  2.6755e-004
3.4859e-005  0.0000e+000  2.6752e-004
3.3469e-005  0.0000e+000  2.6781e-004
3.0274e-005  0.0000e+000  2.6731e-004
2.4808e-005  0.0000e+000  2.6802e-004
1.6120e-005  0.0000e+000  2.6957e-004
4.9583e-006  0.0000e+000  2.6975e-004
-9.0016e-006  0.0000e+000  2.6933e-004
-2.6190e-005  0.0000e+000  2.6839e-004
-4.4898e-005  0.0000e+000  2.6608e-004
```



Como podemos notar, as diferenças são muito pequenas e imperceptíveis ao olho, o que demonstra que o controlador está bem regulado e suave.

Conclusões

Notamos uma grande importância do uso de tais ferramentas na construção de robôs manipuladores e pernas robóticas. Em certos aspectos ela se mostra de simples solução, porém em outras pode não conseguir solução satisfatória nos casos de encontrar da

cinemática inversa, quando temos que encontrar os ângulos a partir de uma posição ou velocidade.

Source code no GitHub

https://github.com/viniciusrsanches/IA368N

Bibliografia

Siegwart, Roland.

Introduction to autonomous mobile robots. - 2nd ed. / Roland Siegwart, Illah R. Nourbakhsh, and Davide

Scaramuzza.

p. cm. - (Intelligent robotics and autonomous agents series)

Slides de aula IA368N – Professor Eric Rohmer