Robótica Espacial
Aula prática nº 5

Cinemática diferencial inversa
Trajetórias no espaço operacional

Vitor Santos

Universidade de Aveiro

13 mar 2025

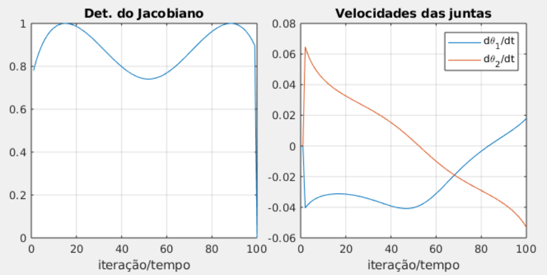
Sumário

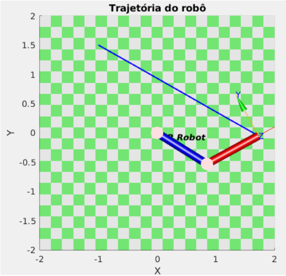
Exercício 1 - Criação da Função `RRjacobianInv()`

1. Criar a função `Ji=RRjacobianInv(Robot,q)` que devolve o jacobiano inverso do robô RR planar por cálculo analítico.
2. Os argumentos são `Robot` do tipo "SerialLink" e q que é o vetor das juntas.
3. A função deve invocar a função criada na aula anterior: `J=RRjacobian(Robot,q)` ...
4. ... e, quando possível, calcular o inverso e retornar esse inverso.
5. A função deve devolver `NaN` se o jacobiano direto for singular.

Exercício 2 - Trajetória linear de um RR planar

- Para um robô RR planar de elos iguais a 1, calcular e representar a trajetória da ponta ao longo de uma linha reta entre o ponto `A=[-1 ; 1.5]` e `B=[1.75; 0];`
- Pode-se usar a função `jacob0` da toolbox e inverter (usar só a parte do jacobiano que interessa!) ou usar diretamente o jacobiano inverso analítico com a função criada.
  - Os resultados devem ser iguais.
- Representar gráficos com as curvas das velocidades das duas juntas
- Representar a animação do movimento do robô com representação da trajetória





Exercício 2 - Detalhes para a resolução

1. O processo recomendado é o de começar por definir a equação paramétrica da trajetória:
   $$path = r(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = A + t * (B - A) \text{ com } 0 \leqslant t \leqslant 1$$
2. pode-se discretizar `t` em `NN` passos usando: `t = linspace(0, 1, NN);`
3. De seguida, e admitindo que a unidade de tempo é uma iteração de um ciclo em Matlab, definem-se os incrementos no espaço operacional como as diferenças entre valores sucessivos da trajetória $r(t)$:
   - Essas diferenças poderiam ser obtidas com o comando `diff` $\mathrm{d}r(t) \approx \Delta r(t)$, ...
   - ... mas é preferível usar o comando `gradient()` que tem a vantagem de manter o número de pontos da sequência e de ter valores mais rigorosos de uma diferenciação numérica: $\mathrm{d}x_i \approx \nabla x_i = \frac{x_{i+1} - x_{i-1}}{2h}$ para os pontos interiores, e $\nabla x_1 = \frac{x_2 - x_1}{h}$ e $\nabla x_N = \frac{x_N - x_{N-1}}{h}$, onde h é o passo ($= 1$ se não especificado)
4. Em matlab pode-se fazer toda esta operação do seguinte modo: `dr=gradient(path);`

Exercício 2 - Restantes detalhes para a resolução

- Para implementar o procedimento deve-se criar um ciclo para simular o movimento.
- Dento do ciclo devem-se fazer estas operações:
  - Calcular o jacobiano inverso para a posição corrente do robô
  - Calcular o incremento das juntas com base no incremento atual do espaço operacional
  - Atualizar o valor das juntas adicionando ao valor anterior o incremento corrente.
- Segue-se um código parcial para implementar o ciclo. Completar o que falta.

```
q = zeros(2, NN);  %initialize the array for the joints' values.
% Initial inverse kinematics to get starting position of robot
q(:, 1) = RR.ikine(transl(A(1), A(2), 0), 'mask', [1 1 0 0 0 0]);
for i = 2:NN
    J = RR.jacob0(***);          % Compute Jacobian
    Ji = pinv(J(1:2, :));        % ... and its inverse
    dq = Ji * dr(***);           % Compute joint increments
    q(:, i) = q(:, i-1) + ***; % update the joint values
end
```

- Ilustrar o movimento com um comando como:
  - RR.plot(q’, ’delay’, 0.01, ’trail’, {’b’, ’LineWidth’, 1.5});

Exercício 3 - Trajetória circular do RR planar

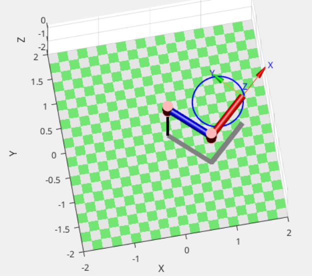
- Com o mesmo robô RR planar descrever uma trajetória circular com centro em `C=[1 ;0]` e raio `r=0.5`, começando o movimento no ponto mais à direita.
- Para descrever a equação paramétrica da trajetória sugere-se usar $0 \leqslant t \leqslant 2\pi$ e o seguinte:

$$path = r(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = C + r * \begin{bmatrix} \cos(t) \\ \sin(t) \end{bmatrix}$$

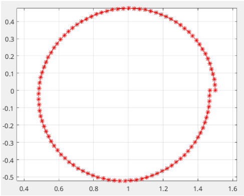
- O resultado porém não é muito preciso, como se pode ver na imagem de baixo que se sugere reproduzir com um código similar ao seguinte (explicar como funciona):

```
allMats=RR.fkine(q');
allP=[allMats.t];
figure(2)
plot(allP(1,:), allP(2,:),'-*r')
grid on, axis equal
```

- Porque acontece, e como se poderia melhorar?



External player



Exercício 3 - Considerações adicionais

- A cinemática inversa numérica pode ter limitações.
- Ela recorre ao jacobiano inverso e parte sempre de uma estimativa inicial
- Se essa estimativa for uma singularidade do jacobiano, em especial em robôs com poucos graus de liberdade, pode dar-se o caso de não convergir!
- Se não se der estimativa inicial, o algoritmo assume que os valores das juntas são zero. Isso pode ser um problema em robôs planares.
- Nesses casos recomenda-se dar estimativas iniciais fora de singularidades.
    - Isso pode-se fazer com uma opção adicional na função `ikine` que é `..., 'q0', qi, ...` onde `qi` é a estimativa inicial
    - No caso do exercício, a posição de partida seria calculada usando algo como:

```
qi=[0 0.1];
q(:, 1)=RR.ikine(transl(path(1,1),path(2,1),0),'q0',qi,'mask',[1 1 0 0 0 0]);
```

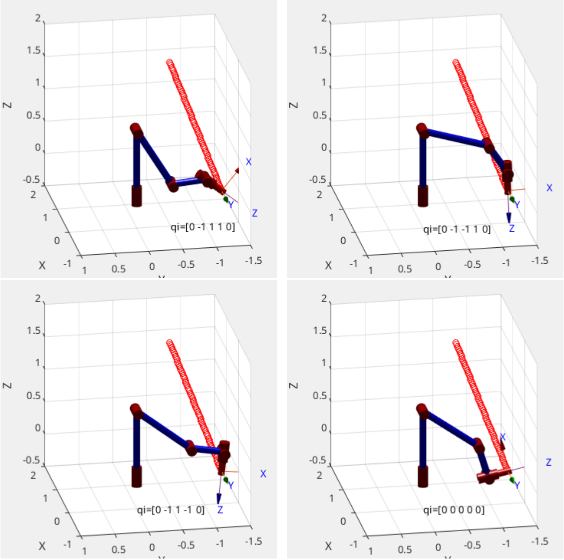
- Na verdade, esta é também a forma de procurar redundâncias alternativas na abordagem numérica da cinemática inversa (por vezes basta usar um `qi` em que o sinal +/- de uma junta altera a redundância obtida.)

Exercício 4 - Trajetória linear a 3D de um robô a 5 DOF

- Criar o robô com os seguintes parâmetros:

```
LA=1; LB=1; LC=0.5; LD=0.3;
DH=[ 0 LA 0  pi/2
     0 0  LB 0
     0 0  LC 0
     0 0  0  pi/2
     0 LD 0  0];
```

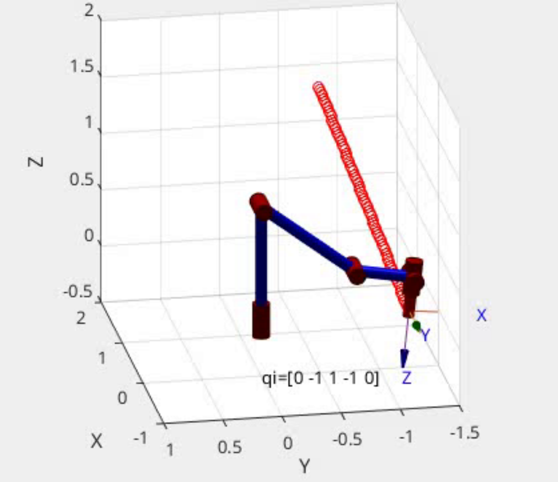
- Criar a trajetória linear de `A=[0 -1.25 0]'` até `B=[1.5 -0.75 1.5]'`.
- Sugere-se dar estimativas iniciais diversas `qi` para procurar diferentes redundâncias.
  - Como referido, dar valores negativos ou positivos para valores iniciais de juntas (sem pensar muito da magnitude) pode dar resultados muito diferentes.
  - Algumas sugestões de valores de estimativas iniciais para `qi`:



Exercício 4 - Ilustração do movimento linear do 5DOF

- Os parâmetros desta visualização foram:

```
R5DOF.plot(q', ...
'notiles',...
'noname',...
'nobase', ...
'noshadow', ...
'scale', 0.5, ...
'trail', {'b', 'LineWidth', 1.5}, ...
'delay', 0.001 ...
); % Plot full motion with trail
```



External player

Exercício 5a - Criação de objetos poliédricos em Matlab

Criar um objeto poliédrico - uma pirâmide quadrangular

1. Definir lista de vértices
2. Definir lista de faces
3. Definir lista de cores das faces

Código base

```
%Definition of vertices
V=[
   1 -1  0   %point 1
   1  1  0   %point 2
  -1  1  0   %point 3
  -1 -1  0   %point 4
   0  0  2   %point 5
  ];
```

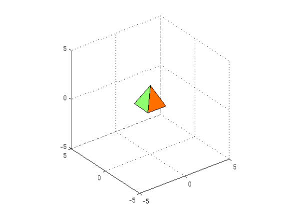
```
%definition of faces
F = [
1 2 5 5   %face1
2 3 5 5   %face2
3 4 5 5   %face3
4 1 5 5   %face4
1 2 3 4   %face5
];
%simple color index to paint the faces
fColor= [ 1 2 3 4 5 ]';
```

Exercício 5b - Representação de objetos poliédricos

Representar a pirâmide

- Representar a pirâmide anterior com o comando:
  - `h=patch('Vertices',V, 'Faces', F, 'FaceVertexCData', fColor,'FaceColor','flat');`
- Antes de fazer a representação pode-se alterar previamente a escala do objeto por exemplo com o comando: `V = 0.25*V` para reduzir a dimensão por 4.

Pirâmide



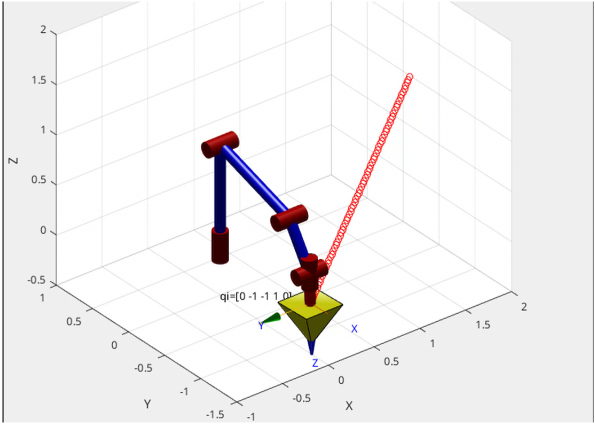
Observação

As transformações geométricas são aplicadas só aos vértices, ou seja, para alterar a posição/orientação do objeto desenhado (*handler* `h`), basta calcular as novas coordenadas dos vértices `V` e alterá-las diretamente no parâmetro `'Vertices'`. Por exemplo, se `V2` forem as novas coordenadas, usar-se-ia o comando: `set(h,'Vertices', V2)`.
Note-se que o comando `patch` espera as coordenadas `x,y,z` arranjadas como vetores linha e quando são manipuladas com transformações geométricas são vetores coluna!

Exercício 6 - Movimento de um robô com objeto na sua mão

- Combinar os exercícios anteriores e simular o movimento linear do robô com a pirâmide na sua extremidade
- Para se poder fazer a animação do objeto em simultâneo com o movimento do robô deve-se fazer a animação *frame* a *frame* da posição do robô e não fazer o plot de todas as posições, como anteriormente.
- Assim indica-se um excerto de código que poderá ajudar a construir a simulação pedida:

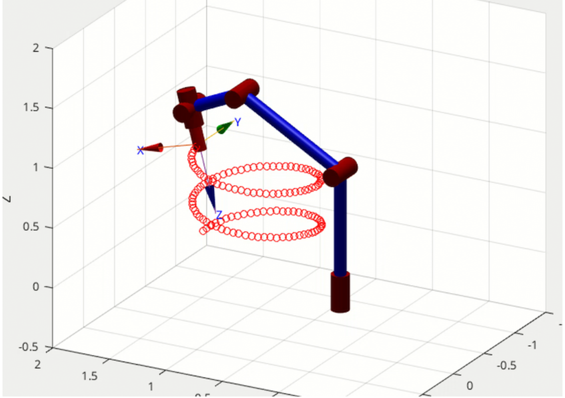
```
R5DOF.animate(q(:,i)'); %animate 1 frame
T=R5DOF.fkine(q(:,i));
Vn=T*V';
h.Vertices=Vn';
drawnow
```



External player

Exercício 7 - Movimento helicoidal de um robô a 5DOF

- Simular um movimento helicoidal da ponta do robô com os seguintes parâmetros:
  - centro inicial em C=[0.75 0 1]'
  - raio r=0.5
  - deve executar 2 espiras completas
  - O movimento ao longo do eixo z deve seguir esta lei: -0.25*t
  - O robô deve iniciar na parte de cima da helicóide e no ponto mais distante.
  - A redundância a usar deve ser tal que a junta 2 deve começar com ângulo positivo e a junta 3 com ângulo negativo.



External player