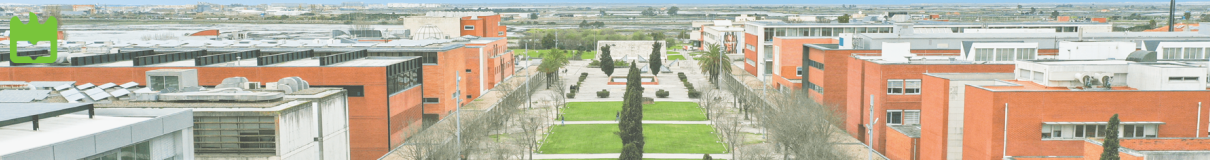


Robótica Espacial
Aula prática nº 4
Ângulos de Euler e Quaternions
Cinemática diferencial direta

Vitor Santos

Universidade de Aveiro

6 mar 2025

Sumário

Ângulos de Euler – RPY

$$\mathrm{RPY}(\phi,\theta,\psi) = \mathrm{rotz}(\phi) \times \mathrm{roty}(\theta) \times \mathrm{rotx}(\psi) =$$
$$\begin{bmatrix} C\phi\,C\theta & C\phi\,S\psi\,S\theta - C\psi\,S\phi & S\phi\,S\psi + C\phi\,C\psi\,S\theta & 0 \\ C\theta\,S\phi & C\phi\,C\psi + S\phi\,S\psi\,S\theta & C\psi\,S\phi\,S\theta - C\phi\,S\psi & 0 \\ -S\theta & C\theta\,S\psi & C\psi\,C\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Impondo $\theta \in]-\pi/2, +\pi/2[$, e sendo $\mathrm{RPY}(\phi,\theta,\psi) = \begin{bmatrix} r_{11} & r_{12} & r_{13} & 0 \\ r_{21} & r_{22} & r_{23} & 0 \\ r_{31} & r_{32} & r_{33} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$, virá:

$$\begin{cases} \phi = \arctan\left(r_{21}, r_{11}\right) \\ \theta = \arctan\left(-r_{31}, \sqrt{r_{32}^2 + r_{33}^2}\right) \\ \psi = \arctan\left(r_{32}, r_{33}\right) \end{cases}$$

Se $\theta = \pm\pi/2$, ϕ ou ψ podem ser arbitrários. Nesse caso: $\psi = 0$ e $\phi = \arctan\left(-r_{12}, r_{22}\right)$.

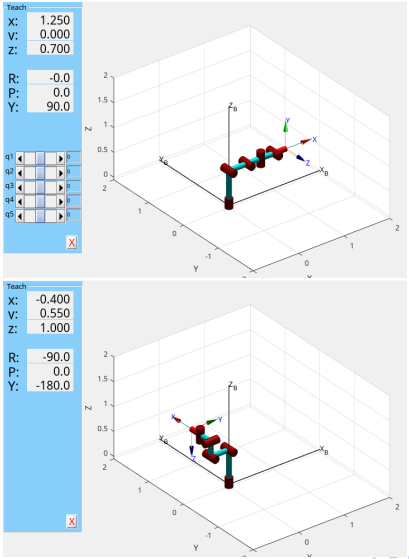
Cálculo dos ângulos de Euler

- Em matlab há várias formas de obter os ângulos de Euler:
  - A partir da matriz de transformação geométrica global `Th`:
    - `tform2eul(Th)`
  - A partir da sub-matriz de orientação `R`
    - `rotm2eul(R)`
- Se se obtiver a transformação (do tipo SE3) da cinemática direta com por exemplo a expressão: `T = robot.fkine(q)`, obtém-se os seguintes elementos úteis para o cálculo:
  - `T.R` – a matriz de rotação
  - `T.T` – a matriz de transformação geométrica global
  - `T.t` – o vetor de translação

Nota: A **Robotics System Toolbox** da MathWorks e a **Robotics Toolbox for MATLAB** de Peter Corke são ambas necessárias para os exercícios.

Exercício 1 - Observação e alteração dos ângulos de Euler

- Representar o robô indicado e ativar o modo `teach()` (figura em cima)
- Criar um referencial na base para comparar os ângulos
- Identificar as orientações da ponta em relação à base quando se movem as juntas (na posição inicial os ângulos de Euler são (0°,0°,90°))
- Observar os ângulos quando as juntas variam
- Procurar encontrar os valores das juntas para obter os seguintes ângulos de Euler (figura de baixo)
  - (R)oll, em torno de z: $-90°$
  - (P)itch, em torno de y: $0°$
  - (Y)aw, em torno de x: $180°$
- Usar e completar o código proposto na página seguinte.



Exercício 1 - Código parcial

Completar o código substituindo os * pelo código adequado**

```
LA=0.7; LB=0.4; LC=0.3; LD=0.3; LE=0.25;        % Robot main measurements
%   q  d   a    alpha
DH=[
    0, LA, 0,   pi/2
    0, 0,  LB,  0
    0, 0,  LC,  -pi/2
    0, 0,  LD,  pi/2
    0, 0,  LE,  0
];
robot = SerialLink(***, 'name', 'Robot5DOF');  % Create robot kinematics chain
q = zeros(1, ***);                             % Initial joint configuration
wSpace=[-1 2 -2 2 -0.05 2];                    % Suggested workspace
fig=figure('Name', 'Euler Angles and Quaternions', 'NumberTitle', 'off'); % Create figure
% Display robot
robot.plot(***, 'workspace', wSpace, 'scale',0.75, 'notiles', 'nobase', 'noshadow',...
           'linkcolor','c', 'noname');
hold on  % to allow multiple graphics and plots in the sme figure
% Plot fixed reference frame at the base
trplot(***, 'frame', 'B', 'length', 2, 'thick', 1, 'color', 'k');
% use in teach mode for interactive modification of joints
robot.teach();
```

Ângulos de Euler

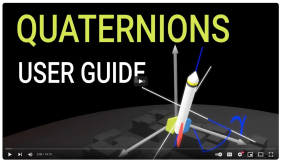
- Os ângulos de Euler são intuitivos e válidos, mas podem apresentar limitações:
  - Pode haver situações de ambiguidade (*Gimbal Lock*)
  - Há múltiplas formas de se obter a mesma orientação em 3D
  - Há múltiplas possibilidades de ângulos conforme a sequência dos eixos para as rotações
  - Não se adequam bem a interpolação para certos movimentos contínuos de rotação
  - Computacionalmente mais exigentes porque usam muitas operações trigonométricas
- Os ângulos de Euler usados nestes exercício são os do tipo RPY com as expressões apresentadas antes
- Traduzem as rotações nesta ordem vistas todas do referencial global
  - 1ª rotx()
  - 2ª roty()
  - 3ª rotz()
- Porém há uma variante popular designada ZYZ que se traduz na seguinte ordem
  - 1ª rotz()
  - 2ª roty(), mas vista do novo referencial após a 1ª rotação
  - 3ª rotz(), mas vista do novo referencial após a 2ª rotação
- Como é natural, as expressões são diferentes das da abordagem RPY

Angulos de Euler e quaternions

- Um alternativa aos ângulos de Euler pode ser usar quaternions
  - Definem-se com uma única rotação em torno de um eixo conveniente.
  - Numero complexo a 4 dimensões: $q = (s, x, y, z)$
    - s é a parte real - indica o ângulo a rodar: $s = \cos\theta/2$ (por vezes usa-se w)
    - x, y, z é a parte imaginária - Componentes do vetor que define o eixo de rotação
    - $q = s + xi + yj + zk = \cos\frac{\theta}{2} + xi + yj + zk$
  - Os quaternions para expressar orientação são ditos unitários, i.e., de norma 1: $||q|| = \sqrt{s^2 + x^2 + y^2 + z^2} = 1$
  - Para isso, é usual normalizar um quaternion antes de o usar em operações ligadas à orientação: $q \rightarrow \frac{q}{||q||}$
  - Em matlab há diversas funções para operar e converter quaternions

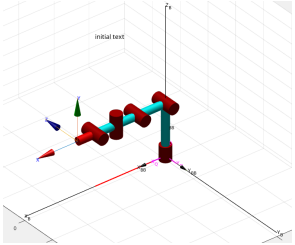
Para uma explicação comparativa mais completa entre ângulos de Euler e quaternions para expressar orientações e rotações, recomenda-se o seguinte tutorial:

`https://youtu.be/bKd2lPjl92c?si=ODpmoE6IMkJYyahd`



Exercício 2 - Preparação para visualização com quaternions

- Criar 4 entidades auxiliares para ilustrar os quaternions:
  - Um texto para visualizar o valor corrente do quaternion
  - Uma linha para ilustrar o eixo de rotação corrente do quaternion
  - Dois sistemas de coordenadas auxiliares:
    - Um que com a orientação final para colocar na ponta do eixo de rotação e que será similar ao que está na ponta do robô
    - Um outro que é a réplica do referencial base, mas que será colocado ao longo do eixo para mais tarde animar e ilustrar o efeito de rotação associado ao quaternion.




```
%Text for quaternion info -- Initial dummy text to update later
hQuatText=text(1,0,2,'initial text','HorizontalAlignment', 'left','FontSize',12);
% Create a line for quaternion axis representation. Use a red line with a thickness of 2
axisLength=1.5; %length of the red line that represents the quaternion rotation axis
quaternionAxisLine = plot3([0, axisLength],[0, 0],[0, 0],'r-','LineWidth',2);
% Create a handle for the quaternion axis frame
hAxisFrame=trplot(***,'frame','Q','length',0.2,'thick',2,'color','m');
% Create a handle for a replica of the base frame to later animate
hBaseFrame=trplot(***,'frame','BB','arrow','width',0.5,'length',0.4,'thick',1,'color','k');
```

Exercício 3 - Representação dos valores do quaternion

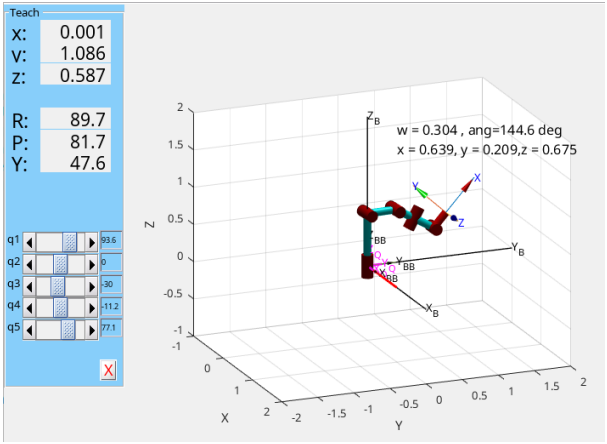
- Criar um ciclo `while` onde se atualize continuamente os valores do quaternion
- Operações importantes:
  - Obter o valor corrente das juntas: `q = robot.getpos();`
  - Obter a matriz de transformação da ponta (cinemática direta): `T = robot.fkine(q);`
  - Obter o quaternion unitário correspondente: `quat = UnitQuaternion(T);`
  - Atualizar o texto com informação do quaternion; o objeto `quat` tem dois campos: `s` e `v`, onde `v` é o vetor de 1x3 com a parte imaginária do quaternion.

```
% Continuously update Euler angles when joints change
while isvalid(robot)
    q = robot.getpos();        % Get current joint angles
    T = robot.fkine(q);        % Compute forward kinematics
    quat = UnitQuaternion(T); % Create corresponding UnitQuaternion from T
    quatAngle=2*acos(quat.s); % Calculate rotation angle after the real part of quaternion
    %Update the string with the complete information about the quaternion
    hQuatText.String = sprintf('w = %.3f , ang=%.1f deg \nx = %.3f, y = %.3f,z = %.3f', ...
                               quat.s, 180/pi*quatAngle, quat.v(1), quat.v(2), quat.v(3));

    pause(0.25);               % Small delay to reduce CPU usage
end
```

Exercício 3 - Ilustração da representação

- Alterando os ângulos das juntas observa-se as variações no quaternion

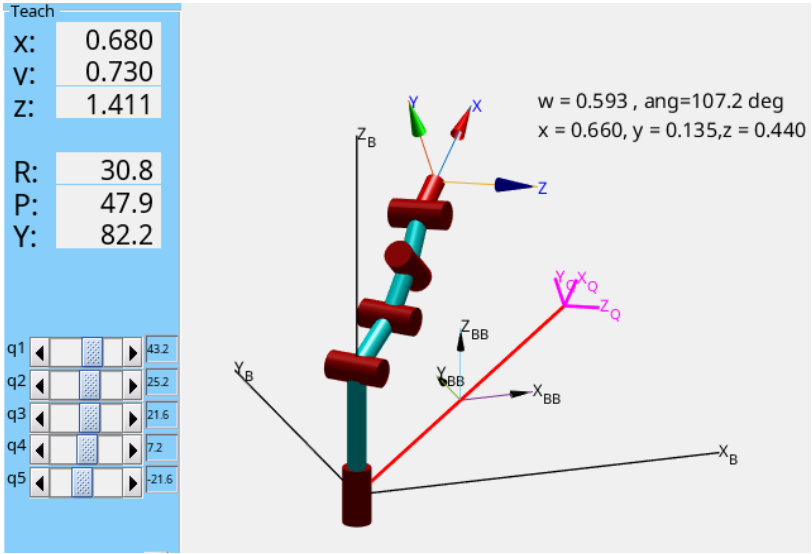


- Porém, ainda não se atualizou o eixo de rotação e os sistemas de coordenadas auxiliares

Exercício 4 - Representação gráfica usando o quaternion

- Representar o eixo do quaternion e os dois sistemas de coordenadas auxiliares
- Operações importantes
  - Obter o vetor do quaternion e normalizá-lo se necessário:
    - `qaxis = quat.v;`
  - Ajustar a representação do eixo criado anteriormente
    - Operações como: `set(quaternionAxisLine, 'XData', [0, axisLength*qaxis(1)], ...)`
  - Obter a matriz de transformação para posicionar o sistemas de coordenadas auxiliar. Além da orientação (o foco deste exercício), inclui-se uma translação para melhor visualização do sistema de eixos.
    - A matriz de orientação obtém-se diretamente como `quat.T`
    - A matriz de translação obtém-se com `transl(axisOffset)` onde `axisOffset` não é mais do que uma escala do vetor do eixo de rotação do quaternion, ou seja, `axisOffset = axisLength * qaxis`.
  - O resultado da matriz é: `quatAxisMatrix = transl(axisOffset) * quat.T;`
- As operações finais dizem respeito à alteração da propriedade `'Matrix'` de ambos os sistemas de eixos `hAxisFrame` e `hBaseFrame` com a respetiva matriz
- Segue uma ilustração e o código deste bloco para completar.

Exercício 4 - Imagem do sistema com ilustração completa



Exercício 4 - Exemplo parcial de código

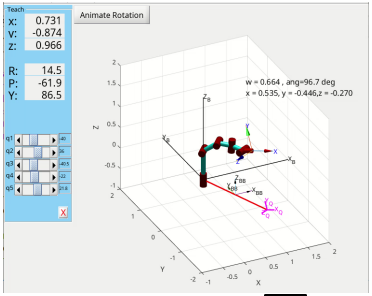
Substituir os "*"pelo código apropriado**

```
while isvalid(robot)
    %... code from previous exercises
    qaxis = quat.v;  % extract the quaternion vector part (x, y, z)
    if any(quat.v)   % normalize only if non null vector (in case no rotation exists)
        qaxis = qaxis/norm(***); % Normalize to unit vector
    end
    % Update the axis line representing the quaternion axis of rotation
    set(quaternionAxisLine, 'XData', [0, axisLength*qaxis(1)], ...
                            'YData', [0, axisLength*qaxis(***)], ...
                            'ZData', [0, axisLength*qaxis(***)]);
    axisOffset = axisLength * qaxis;
    quatAxisMatrix = transl(axisOffset) * quat.T;
    % Update the quaternion axis frame location and orientation
    set(hAxisFrame, 'Matrix', quatAxisMatrix);
    % Update the base reference frame location to haf way along the quaternion axis
    set(hBaseFrame, 'Matrix', transl(***/2));

    pause(0.25); % Small delay to reduce CPU usage
end
```

Exercício 5 (opcional) - Animação da rotação em torno do eixo

- Para uma ilustração mais completa, pode-se fazer a animação do referencial de base e verificar como ele se alinha com o referencial da orientação da ponta.
- A classe de objetos `UnitQuaternion` tem um método que faz a animação do sistema de coordenadas base, basta fazer: `quat.animate`.
- Porém, essa animação é feita na origem do sistema de eixos, e tem algumas limitações.
- Assim, propõe-se fazer a animação do sistema de coordenadas de base criado ('BB')
- Em termos de metodologia propõe-se a criação de um botão na interface gráfica que despoleta a animação quando for pressionado.
- Seguem sugestões de código de apoio.



External player

Exercício 5 - Sugestões de código a incorporar no programa

No programa principal antes do ciclo *while*

```
%create button
btn = uicontrol('Style', 'pushbutton',...
'String', 'Animate Rotation', ...
'FontSize', 12, 'Position',...
[145 figHeight-40 160 40], 'Callback',...
@(src, event) setappdata(fig, ...
'buttonPressed', true));
% Initialize button flag
setappdata(fig, 'buttonPressed', false);
% Adjust position if figure is resized
addlistener(fig, 'SizeChanged', ...
@(src, event) set(btn, 'Position', ...
[145, src.Position(4)-40, 160, 40]));
```

Dentro do ciclo *while* antes da *pause* final

```
if getappdata(fig, 'buttonPressed')
% Reset flag after action
setappdata(fig, 'buttonPressed', false);
animateFrame(quat,qaxis,axisOffset,...
hBaseFrame)
end
```

Função *animateFrame()*

```
function animateFrame(quat,qaxis,axisOffset,bFrame)
numSteps = 50; % Number of animation frames
pauseTime = 0.05; % Pause duration per frame
% Animate rotation around the quaternion axis
ang=2*acos(quat.s);
allangs=linspace(0, ang, numSteps);
%Add some more frames :-)
allangs=[allangs  repmat(allangs(end),1,10)...
         fliplr(allangs(1:2:end))];
for m = allangs
% Compute incremental rotation matrix
  % Convert axis-angle to 3x3 rotation matrix
  R_inc = axang2rotm([qaxis, m]);
       % Convert to homogeneous 4x4
  T_inc = [R_inc, [0; 0; 0]; 0, 0, 0, 1];
%Update base reference frame using the transform.
 set(bFrame, 'Matrix', transl(axisOffset/2)*T_inc);
 pause(pauseTime); % Pause forsmooth animation
end
pause(1)
end
```

Exercício 6 - Jacobiano direto do RR planar

Criar um robô RR planar e o seu jacobiano na forma simbólica

- Criar variáveis simbólicas para juntas e comprimentos de elos:
  - `syms q1 q2 LA LB real`
- Definir a tabela de DH com os valores simbólicos:

elo i	θ_i	d_i	a_i/l_i	α_i
1	θ_1	0	L_A	0
2	θ_2	0	L_B	0

- Com `SerialLink` criar a cadeia cinemática do robô dando-lhe o nome 'RR planar'
- Obter a cinemática direta do robô usando o método `fkine` com argumento `[q1 q2]`.
- Extrair a posição (x,y) da cinemática direta anterior (campo `t` da matriz da cinemática)
- Com a função `jacobian` obter o jacobiano direto da posição em função de `[q1 q2]`.

Exercício 7 - Função para calcular o Jacobiano do RR planar

- A partir do resultado simbólico do exercício anterior criar uma função que retorne o valor do Jacobiano para um robô RR planar e para uma qualquer posição:
- `function J=RRjacobian(robot,Q)`
  - `robot` é do tipo `SerialLink`
  - `Q` é um vetor com o valor das juntas do robô
- A função deve analisar a variável `robot` para extrair os comprimentos dos elos associados ao robô em causa.
- Sugestão parcial da função onde se deve substituir "***"pelo código adequado

```
function J=RRjacobian(robot,Q)
LA=robot.a(***);
LB=robot.a(***);
q1=Q(***);
q2=Q(***);
J =[
    ***, ***
    ***, ***];
end
```

Exercício 8 - Testar a função `RRjacobian()` num robô concreto

- Impondo $LA = 2$ e $LB = 1$, criar uma versão concreta (não puramente simbólica) do robô do exercício anterior:

elo i	θ_i	d_i	a_i/l_i	α_i
1	θ_1	0	LA	0
2	θ_2	0	LB	0

- Não esquecer de não incluir q1 e q2 (que foram definidas antes com simbólicas) na tabela de DH porque isso forçará a sua representação simbólica modificando o aspeto do resultado final (embora esteja correto, será menos legível).
- Calcular o jacobiano para Q=[pi/4 pi/3] de duas formas:
  - Com a função criada `RRjacobian()`
  - Com o método `jacob0` associado à classe `SerialLink` que, na verdade, usa a abordagem do Jacobiano Geométrico e não a via analítica que deu origem à função criada `RRjacobian()`.
- Os resultados pelos dois métodos deverão ser os seguintes:

```
-2.3801    -0.9659
1.1554    -0.2588
```

```
-2.3801    -0.9659
1.1554    -0.2588
0           0
0           0
0           0
1.0000     1.0000
```

- A segunda variante tem 6 linhas, porquê?
- Como se pode utilizar?

Exercício 9 - Velocidade das juntas para o robô RR planar

Obter a seguinte trajetória nas juntas (`jtraj()`)

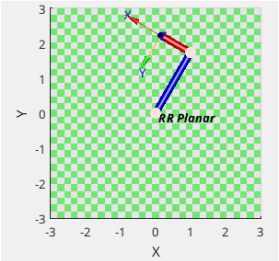
- Ponto de partida `Qi=[0 0]`
- Ponto de chegada `Qf=[pi/3 pi/2]`
- Número de passos `NN=100`

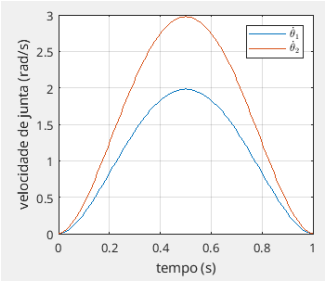
Calcular a velocidade das juntas em cada instante

- Usar a função `diff`
- Colocar os resultados na variável `vq`
- Duração da trajetória: `Dt=1 s`

Numa janela dupla mostrar:

- A animação do movimento robô entre Qi e Qf
- Um gráfico com as curvas das velocidade das juntas (e com as escalas e unidades corretas)





Exercício 10 - Cálculo das velocidades da ponta

Calcular e guardar as velocidades da ponta ao longo da trajetória

- Inicializar um array para guardar as velocidades da ponta
- Fazer um ciclo `for` para o cálculo do jacobiano em cada ponto e daí calcular as velocidades da ponta
- Atender às dimensões do jacobiano dado por `jacob0` e dos vetores das velocidades;

$$vr(i) = \begin{bmatrix} v_x(i) \\ v_y(i) \end{bmatrix} = J(Q)vq(i) = J(Q) \begin{bmatrix} vq_1(i) \\ vq_2(i) \end{bmatrix}$$

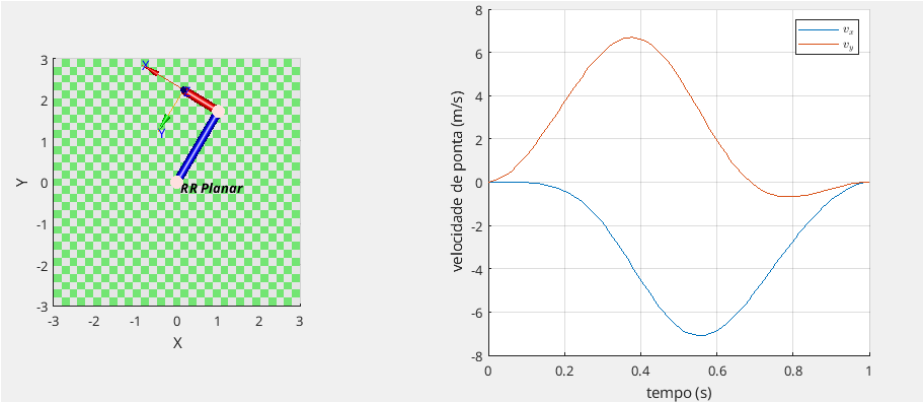
- Representar as velocidades da ponta e ilustrar com a animação do robô

Notas e observações

- A inicialização prévia do array de velocidades é opcional mas pode melhorar o desempenho do Matlab
- Em casos críticos de simulação ou de controlo exigente pode ser útil pré-calcular os jacobianos de antemão para manter a cadência do ciclo, embora isso não seja em geral problemático.
- O jacobiano geométrico tem por defeito 6 linhas (pelas 6 variáveis no espaço operacional) mas em robôs de poucos graus de liberdade vários termos são nulos e podem ser descartados para evitar as multiplicações inúteis por zero.

Exercício 10 - Resultado

As curvas das velocidades da ponta refletem o movimento na simulação



- Observe-se as curvas e interprete-se o resultado
- Porque é a velocidade v_x sempre negativa?
- Porque é que a velocidade v_y fica ligeiramente negativa perto do fim da trajetória?