Robótica Espacial
Aula prática nº 10
Robôs móveis - Odometria e Sensores Inerciais

Vitor Santos

Universidade de Aveiro

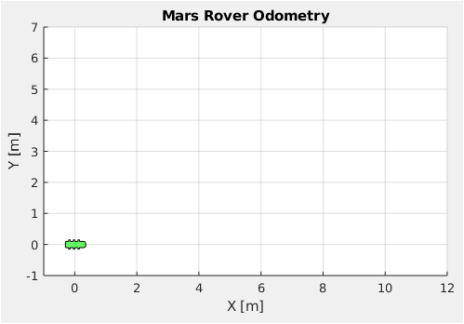
8 maio 2025

Sumário

Exercício 1 - Modelo simples de Rover

- Criar um robô similar a um rover para simular o seu movimento por controlo da sua velocidade linear e angular (independentemente da respetiva estrutura cinemática).
- Usar a função fornecida `robotShape()` para criar a geometria do robô.
- Estabelecer os parâmetros gerais da simulação como indicado.
- Um código base possível será o seguinte que deve ser completado (***)

```
rover_shape = robotShape();
% Preparation and initial representation
figure;
axis equal; grid on; hold on
% Indicative initial limits
xlim([-1 12]); ylim([-1 7]);
xlabel('X [m]'); ylabel('Y [m]');
title('Mars Rover Odometry');
% Create graphic handle for the rover
h_true = fill(***, ***, 'g', 'FaceAlpha', 0.6);
% General simulation parameters
dt = 0.1;           % Time step [s]
T = 20;             % Total time [s]
N = round(T/dt);    % Number of steps
```

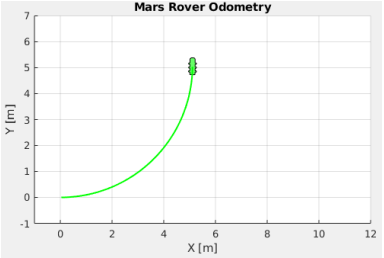


Exercício 2 - Movimento e odometria sem erros

- A trajetória será visualizada com a função `animatedline()` criada inicialmente vazia.
- À medida que se calcularem novos pontos será atualizada com a função `addpoints()`.
- Também deve ser criada uma matriz para guardar as poses ao longo da trajetória (x, y, θ)
- Devem ser definidas as velocidades lineares e angulares iniciais

- Uma sugestão de código é a seguinte:

```
% Assume initial constant velocities
v_cmd0 = 0.4;        % linear  [m/s]
w_cmd0 = pi/40;      % angular [rad/s]
% Matrix to store the states [x; y; theta]
true_pose = zeros(3, N);
% Create the empty trajectory as an animatedline
traj_true = animatedline('Color','g','LineWidth',1.5);
```



- A trajetória resultante será como a ilustrada e ver mais detalhes adiante de como implementar a simulação.

Ex. 2 - Implementação do ciclo de simulação do movimento

- Sugestão para o ciclo de simulação onde se deve completar os ***:

```
for k = 2:N  % Why starting in 2?
    w_cmd = w_cmd0;
    v_cmd = v_cmd0;
% Calculate current position after previous
    true_pose(3,k)=true_pose(***)+w_cmd*dt;
    theta=true_pose(3,k);
    true_pose(1,k)=true_pose(***)+v_cmd*cos(***)*dt;
    true_pose(2,k)=true_pose(***)+v_cmd*sin(***)*dt;
% Update visuals
    R_true = [cos(***) -sin(***))
              sin(***)  cos(***)];
    shape_true = R_true * rover_shape + true_pose(***);
    h_true.XData=shape_true(1,:);
    h_true.YData=shape_true(2,:);
% Upate trajectory
    addpoints(traj_true, true_pose(***), true_pose(***));
    pause(0.01)
end
```

Assumindo que $\Delta\theta_k$ é pequeno no intervalo de medida Δt, a versão discreta do cálculo da odometria é dada por:

$$\theta_k = \theta_{k-1} + \Delta\theta_k$$
$$x_k = x_{k-1} + \Delta l_k \cos\theta_k$$
$$y_k = y_{k-1} + \Delta l_k \sin\theta_k$$

Sendo:

$$\Delta\theta_k \approx \omega_k \Delta t$$
$$\Delta l_k \approx V_k \Delta t$$

- Ajustar o código para inverter a velocidade angular a meio da trajetória, isto é: `w_cmd` passar a ser `-w_comd0`
- Para uma simulação mais longa recomenda-se também aumentar T para 30.

Exercício 3 - Movimento e odometria com erros

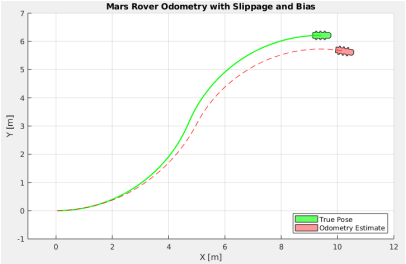
- O objetivo é simular a existência de erros e comparar os dois resultados com e sem erros.
- Isso deve ser ilustrado com a simulação/animação simultânea das duas situações.
- Há duas fontes de erros:
  - Erros sistemático nas velocidades impostas (constantes)
  - Erros aleatórios por exemplo com origem em derrapagem ou patinagem das rodas (por exemplo erros Gaussianos de média nula e uma dada variância)
- Sugerem-se os seguintes valores iniciais para esses erros:

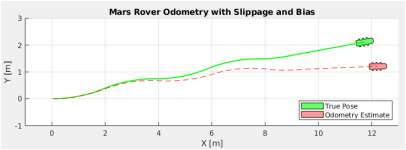
```
bias_v =  0.015;    % Systematic bias in linear velocity (e.g., worn wheel)
bias_w = -0.005;    % Systematic bias in angular velocity (id.)
sigma_v = 0.02;     % Random noise std dev for linear velocity (emulate slippage)
sigma_w = 0.01;     % Random noise std dev for angular velocity (idem)
```

- Estes erros devem ser incorporados (adicionados) às respetivas velocidades para calcular a variante de simulação com erro
- Sugestão da operação para a velocidade linear na situação com erro (`v_meas`):
  - `v_meas = v_cmd + bias_v + sigma_v*randn();`

Ex.3 - Indicações adicionais para a simulação com erro

- Para efetuar as duas simulações é preciso criar código adicional para a variante de simulação com erro.
- Fora do ciclo:
  - Matriz para guardar as poses (`odom_pose`)
  - Criar outro modelo do rover (Sugere-se outra cor `'r'`)
  - Criar outra trajetória (e.g. linha vermelha tracejada)
- Dentro do ciclo:
  - Fazer o cálculo de atualização da pose/movimento para a variante com erro;
  - Atualizar a posição do rover com erro (vermelho);
  - Atualizar a sua trajetória.
- É possível alterar dinamicamente as velocidades ainda mais. Por exemplo, se a velocidade angular se tornar simétrica a cada quarto da trajetória, o resultado final seria o indicado na figura de baixo.





Observa-se que erros nas velocidades impostas ou na interação com o chão podem levar a desvios grandes, sendo este o problema principal da **odometria.**

Exercício 4 - Ligação a smartphone com Matlab Mobile

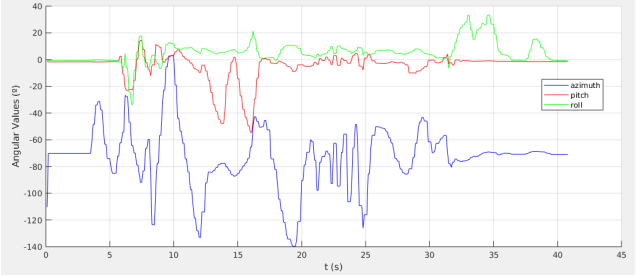
- Instalar o pacote de suporte Matlab para Android (ou iOS) no computador pessoal
- Instalar o Matlab Mobile (Android ou IoS) no smartphone pessoal
- Executar a aplicação no smartphone e entrar na conta pessoal da Mathworks
- Selecionar "sensors" na aplicação do smartphone
- Ativar os sensores que se pretendem monitorizar (orientação, aceleração, etc.)
- No computador criar um objeto para aceder ao dispositivo móvel:
  - `m=mobiledev`
- No computador também é possível ativar ou desativar os sensores do smartphone:
  - `m.AccelerationSensorEnabled=1;`, `m.AngularVelocitySensorEnabled=1;`, etc.
  - Para ativar o envio de dados pode-se carregar no botão "Start"no smartphone ou fazer `m.Logging=1` no programa do computador.
- Verificar os dados disponíveis nos campos de `m` como:
  - `m.Acceleration, m.AngularVelocity, m.Orientation, etc.`

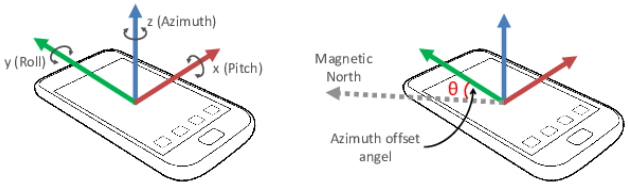
Mais detalhes sobre a metodologia em:
`https://www.mathworks.com/help/matlabmobile/ug/sensor-data-collection-with-matlab-mobile.html`

Exercício 5 - Traçar gráficos das leituras do smartphone

- Com `animatedline()` mostrar em contínuo os últimos 500 valores das 3 orientações
- Completar o código em falta ***

```
close
m=mobiledev
h=figure;
m.OrientationSensorEnabled=1;
m.Logging = 1;  %start logging
ang1=animatedline('Color', 'b', *** ,500);
ang2= ***; ang3= ***;
t=0; dt=0.1;
grid on; ylim=[-180 180];
legend('azimuth', 'pitch', 'roll');
while isvalid(h)
    t=t+dt;
    if numel(m.Orientation) > 2
        addpoints(***);
        ***
    end
    pause(***)
end
m.Logging = 0; %stop logging
```

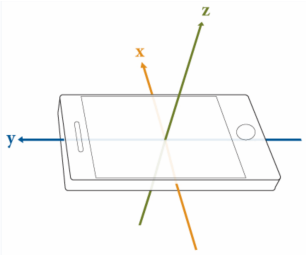




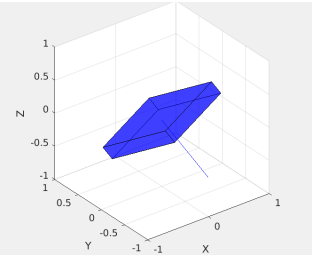
NB: Orientações dadas pelo magnetómetro interno do smartphone

Exercício 6 - Orientação gravítica do smartphone

- Ativar os acelerómetros do smartphone
- Assumir o smartphone estático para que as acelerações medidas sejam **apenas** as da gravidade.
- Obter o vetor de gravidade normalizado:
  - `g=accel/norm(accel)` (sendo `g=[gx gy gz]`)
- Obter os ângulos de orientação do smartphone:
  - Pitch: em torno de y: `atan2(-gx, sqrt(gy^2+gz^2));`
  - Roll: em torno de x: `atan2(gy,gz)`
- Desenhar um paralelepípedo emulando o smartphone e ilustrar a sua orientação em relação à gravidade (e.g. uma linha de comprimento 1) com dados da aquisição remota
- **NB.** O sistema pode não ser muito reativo dependendo das condições da rede.



Eixos dos acelerómetros do smartphone



Exemplo de orientação medida no exercício

Ex. 6 (cont.) - Sugestões auxiliares de código

Definir o sólido para representar o smartphone

```
% Vertices of the phone: top face (1-4), bottom face (5-8), sides (9-12)
vertices = [0.5  0.5 -0.5 -0.5  0.5  0.5 -0.5 -0.5  % X
            0.8 -0.8 -0.8  0.8  0.8 -0.8 -0.8  0.8  % Y
            0.1  0.1  0.1  0.1 -0.1 -0.1 -0.1 -0.1];% Z
% Faces of phone: front (1), back (2), left (3), right (4), top (5), bottom (6)
faces = [1 2 6 5  % front
         4 3 7 8  % back
         1 4 8 5  % left
         2 3 7 6  % right
         1 2 3 4  % top
         5 6 7 8];% bottom
% Plot the initial orientation of the rectangular prism
h=patch('Vertices', vertices', 'Faces', faces, 'FaceColor', 'b', 'FaceAlpha', 0.5);
view(3)
```

Dentro do ciclo atualizar os vértices com transformações geométricas a calcular:

```
% Compute rotated vertices
rotated_vertices = R_roll * R_pitch * vertices;
% Update plot with rotated prism
h.Vertices=rotated_vertices';
```

Exercício 7 - Medição de orientação de smartphone com IMU

- A leitura dos sensores inerciais (Acelerómetros e giroscópios) pode ser usada para estimar o movimento.
- Os dados são muito ruidosos e é necessário usar ferramentas de filtragem (EKF, etc.)
- Dada a sensibilidade ao tempo, recomenda-se adquirir os dados localmente no aparelho e depois processá-los no computador para maior rigor.
- Também se recomenda usar frequências de amostragem elevadas (100 Hz ou mais)
- Propõe-se executar o seguinte comando com um exemplo da mathworks:
  - `openExample('shared_positioning/EstimateIPhoneOrientationUsingSensorFusionExample')`
- que tem um video de apoio que pode ser encontrado aqui:
  - `https://youtu.be/J-9Z3pMUrdI?feature=shared`
- Há muitos outros exemplos nesta linha, como este: `https://www.mathworks.com/help/nav/ug/estimate-position-and-orientation-of-a-ground-vehicle.html`
- Pode ser necessário instalar toolboxes adicionais como a Navigation Toolbox.