PMR3502 - Elementos de robótica **Lista 03**

Bruno Scaglione NUSP: 10335812

Lucas Hattori Costa NUSP: 10335847

Data: 20 de setembro de 2021

Descrição do sistema

Um robô triciclo como o da Figura 1 se desloca no plano xy. As rodas motoras giram em uma velocidade determinada por um controlador. O ângulo de basculamento da roda dianteira também é controlado.

Nota: Este sistema, a menos da parte sensorial, é idêntico ao estudado no exercício de EKF.

Este robô anda por um ambiente no qual existem objetos puntuais idênticos entre si.

Adota-se neste problema o vetor de estado do sistema dado por:

$$X_t = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

acrescido do mapa do ambiente, representado por:

$$M = \begin{bmatrix} p_{1x} \\ p_{1y} \\ \vdots \\ p_{m_x} \\ p_{m_y} \end{bmatrix}$$

onde x, y são as coordenadas da posição do robô, p_{ix}, p_{iy} são as coordenadas do i-gésimo objeto e m é a quantidade total de objetos no ambiente.

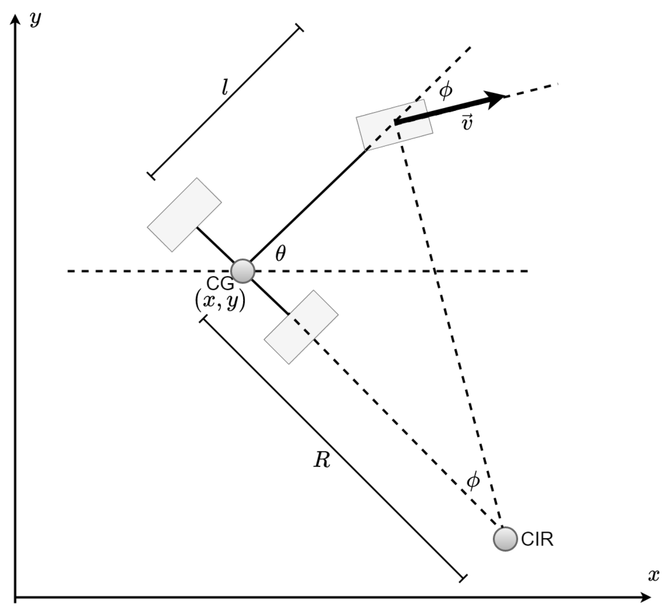


Figura 1: Diagrama do enunciado

A ação de controle do sistema é representada pelo vetor:

$$u_t = \begin{bmatrix} v \\ \phi \end{bmatrix}$$

onde v é a velocidade longitudinal imposta pelas rodas motoras e ϕ é o ângulo de basculamento da roda dianteira.

Sistemas de coordenadas

Neste problema serão adotados 3 sistemas de coordenadas:

1. Sistema *cartesiano global*: Sistema global de coordenadas. Coordenadas x, y em relação à à origem do sistema. Ângulos são medidos em relação ao eixo x. Este sistema é usado para descrever a posição e orientação do robô e as coordenadas de cada marco do mapa.

2. Sistema *polar local*: Medido em relação ao robô. Pontos são definidos pela sua distância até o centro do robô ρ e seu azimute em relação ao eixo longitudinal do robô ψ. Neste sistema são obtidas as leituras do sensor do robô.

3. Sistema *cartesiano local*: Medido em relação ao robô. Pontos são definidos pelas suas coordenadas lr em um sistema cartesiano com o eixo l paralelo ao eixo longitudinal do robô e o eixo r perpendicular a este. Este sistema é usado para evitar problemas de singularidade no sistema polar. As medidas em coordenadas polares são convertidas para este sistema.

Questões

Questão 1

O robô dispõe de um sensor de varredura capaz de detectar objetos à sua frente. Para cada objeto, o sensor oferece um par de valores ρ, ψ, onde ρ é a distância do objetos ao centro do robô e ψ o azimute do objetos, definido como o ângulo em que está o objetos, em relação ao eixo longitudinal do robô (ângulos positivos significam obstáculos à esquerda).

Esta medida neste sistema de coordenadas é feita com ruído Gaussiano de média nula e covariância $0,25^2$ em ρ e $(0.0436)^2$ radianos em ψ. Os dois ruídos são *independentes*.

Para simplificar o processamento, é conveniente transformar as coordenadas polares obtidas pelo sensor em coordenadas cartesianas no referencial do robô, definido como centrado no centro do robô, com a coordenada l paralela ao eixo longitudinal do robô e a coordenada r perpendicular a este e apontando para a esquerda.

a) Escreva as funções de transformação $t(\rho, \psi)$ e $n(\rho, \psi)$ que convertem uma medida de objetos ρ, ψ em coordenadas l, r.

b) Seja Q a matriz de covariância da medida de um objetos no sistema de coordenadas l, r. Escreva a expressão de $Q(\rho, \psi)$.

Solução:

a) As funções de transformação são simplesmente transformações do sistema polar local para o sistema cartesiano local. Logo:

$$\begin{bmatrix} l \\ \\ r \end{bmatrix} = \begin{bmatrix} t(\rho, \psi) \\ \\ n(\rho, \psi) \end{bmatrix} = \begin{bmatrix} \rho \cdot \cos(\psi) \\ \\ \rho \cdot \sin(\psi) \end{bmatrix} \tag{1}$$

b) Como ρ, ψ para l, r se trata de uma transformação não-linear, para utilizar as propriedades de covariância é necessário linearizar a função. Para isso utilizamos o Jacobiano.

Utilizando a propriedade de multiplicação por constante da covariância, a matriz de covariância Q_{lc} no referencial local cartesiano em função da covariância Q_{lp} no referencial local polar é dada por:

$$Q_{lc} = J_{lc,\,lp} \cdot Q_{lp} \cdot J_{lc,\,lp}^T \tag{2}$$

onde J é o Jacobiano de l, r em função de ρ, ψ e é dado por:

$$J_{lc,\,lp} = \begin{bmatrix} \frac{\partial l}{\partial \rho} & \frac{\partial l}{\partial \psi} \\ \frac{\partial r}{\partial \rho} & \frac{\partial r}{\partial \psi} \end{bmatrix} = \begin{bmatrix} \cos(\psi) & -\rho\sin(\psi) \\ \sin(\psi) & +\rho\cos(\psi) \end{bmatrix} \tag{3}$$

Assim, temos:

$$Q_{lc} = Q(\rho,\psi) = \begin{bmatrix} \rho^2 s_\psi^2 \sin^2(\psi) + s_\rho^2\cos^2(\psi) & 0.5\left(s_\rho^2 - \rho^2 s_\psi^2\right)\sin(2\psi) \\ 0.5\left(s_\rho^2 - \rho^2 s_\psi^2\right)\sin(2\psi) & \rho^2 s_\psi^2\cos^2(\psi) + s_\rho^2\sin^2(\psi) \end{bmatrix} \tag{4}$$

onde $s_\rho = 0.0436^2$ e $s_\psi = 0.25^2$ são as covariâncias de ρ e ψ respectivamente.

Questão 2

Considere o robô no estado $[x, y, \theta]^T$. Suponha que o robô detectou um objeto nas coordenadas locais l, r com matriz de covariância Q (neste mesmo referencial). Não há nenhum conhecimento prévio sobre o objeto.

a) Qual a posição esperada no referencial global?

b) Qual a covariância no referencial global em função de θ e Q ?

Solução:

a) A posição esperada no referencial global se da pela transformação do objeto nas coordenadas locais cartesianas para as coordenadas globais. A matriz homogênea A_g^{lc} realiza a transformação e está descrita abaixo:

$$A_g^{lc} = \begin{bmatrix} Rot_{g_{(2\times2)}}^{lc} & & P_{g_{(2\times1)}}^{lc} \\ 0 & 0 & 1 \end{bmatrix} \tag{5}$$

onde: $Rot_g^{lc}(\theta)$ é a matriz de rotação do referencial local cartesiano para o global que depende do ângulo θ do estado X do robô e dada pela Equação 6; e $P_g^{lc}(x,\ y) = \begin{bmatrix} P_{g_x}^{lc} \ P_{g_y}^{lc} \end{bmatrix}^T$ é o vetor de translação do referencial local cartesiano para o global que depende das posições x e y do estado X do robô.

$$Rot_g^{lc}(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix} \tag{6}$$

Então, de coordenadas locais cartesianas para globais temos:

$$\begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} = A_g^{lc} \cdot \begin{bmatrix} l \\ r \\ 1 \end{bmatrix} \tag{7}$$

b) Sabendo a transformação linear que transforma coordenadas locais cartesianas em globais, é possível obter a relação de covariância pela propriedade de multiplicação por constante. Como a covariância independe da translação, utiliza-se a matriz de rotação. Assim temos:

$$Q_g = Rot_g^{lc}(\theta) \cdot Q_{lc} \cdot \left(Rot_g^{lc}(\theta)\right)^T \tag{8}$$

Questão 3
Considere o robô no estado $[x, y, \theta]^T$ e um obstáculo na posição p_x, p_y (em coordenadas globais).

a) Escreva a expressão do valor esperado da medida do objeto nas coordenadas l, r.

b) Escreva a matriz C, gradiente do vetor do item a) em relação às variáveis p_x, p_y.

c) Se este objeto fosse observado pelo sensor, quais seriam os seus valores de ρ, θ? Para simplificar a resposta, escreva em função de l, r do item a)).

d) Escreva a matriz de covariância Q de tima medida do objeto no referencial l, r considerando que a mesma é feita com as incertezas do sensor ρ, θ descritas na introdução (Novamente, por simplicidade, pode escrever em função de valores de ρ, θ do item c). Sugestão: Vide a resposta ao item 1 b.

Solução:

a) Como A_g^{lc} já foi obtida anteriormente, é necessário apenas inverter a matriz para obter A_{lc}^g, portanto:

$$A_{lc}^g = \left(A_g^{lc}\right)^{-1} \tag{9}$$

e a mudança de coordenadas globais para locais cartesianas, utilizando a expressão da inversa de matriz de transformação homogênea, é dada por:

$$\begin{bmatrix} l \\ r \\ 1 \end{bmatrix} = A_{lc}^g \cdot \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} = \begin{bmatrix} \left(Rot_{g_{(2\times2)}}^{lc}\right)^T & -\left(Rot_{g_{(2\times2)}}^{lc}\right)^T \cdot P_{g_{(2\times1)}}^{lc} \\ 0 \quad 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} \tag{10}$$

b) Pela Equação 10 é possível ver que a matriz C, gradiente do vetor $[l\ r\ 1]^T$, é a matriz de rotação de coordenadas locais cartesianas para globais transposta $\left(Rot_g^{lc}\right)^T$, pois os outros elementos apenas contribuem com constantes na expressão de $[l\ r\ 1]^T$. Assim temos:

$$C = \left(Rot_g^{lc}\right)^T \tag{11}$$

c) Para fazer a transformação de coordenadas globais para locais polares, fazemos a transformação de globais para locais cartesianas seguida da transformação de locais cartesianas para locais polares. Como a primeira transformação foi feita na questão anterior, é preciso somente realizar a segunda, que é dada por:

$$\begin{bmatrix} \rho \\ \psi \end{bmatrix} = \begin{bmatrix} \sqrt{l^2 + r^2} \\ arctan2(r, l) \end{bmatrix} \tag{12}$$

d) A resposta é a mesma do Item b), ou seja:

$$Q_{lc} = Q(\rho, \psi) = \begin{bmatrix} \rho^2 s_\psi^2 \sin^2(\psi) + s_\rho^2 \cos^2(\psi) & 0.5\left(s_\rho^2 - \rho^2 s_\psi^2\right)\sin(2\psi) \\ 0.5\left(s_\rho^2 - \rho^2 s_\psi^2\right)\sin(2\psi) & \rho^2 s_\psi^2 \cos^2(\psi) + s_\rho^2 \sin^2(\psi) \end{bmatrix} \tag{13}$$

Questão 4
O robô só consegue detectar objetos distantes de no máximo 3 metros e à sua frente com ângulos de $\pm 60°$. Suponha o robô na posição $x = 1, y = 2, \theta = 30°$. Determine para os seguintes obstáculos qual pode ser detectado e qual não:

1. $(2, 2)$
2. $(1, 0)$
3. $(3, 4)$
4. $(5/2, 5)$

***Solução**:*
Para verificar se o objeto pode ser detectado, é útil transformar as coordenadas globais dadas para as coordenadas locais polares, pois a base local polar é a nativa das restrições do robô. Para isso primeiro é feita a transformação para locais cartesianas e em seguida para locais polares, como abaixo:

$$\begin{bmatrix} l \\ r \\ 1 \end{bmatrix} = \begin{bmatrix} \cos(\theta) & \sin(\theta) & -\cos(\theta)\cdot x - \sin(\theta)\cdot y \\ -\sin(\theta) & \cos(\theta) & \sin(\theta)\cdot x - \cos(\theta)\cdot y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix} = \begin{bmatrix} .8660254 & 0.5 & -1.8660254 \\ -0.5 & .8660254 & -1.23205081 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} p_x \\ p_y \\ 1 \end{bmatrix}$$

$$\begin{bmatrix} \rho \\ \psi \end{bmatrix} = \begin{bmatrix} \sqrt{l^2 + r^2} \\ arctan2(r, l) \end{bmatrix}$$

Também é importante transformar os ângulos de graus para radianos, para estarem na mesma medida de ψ. Os ângulos $\pm 60°$ em radianos são $\pm 1.0471975512\ rad$

Caso 1: (2, 2)

Em coordenadas locais cartesianas:

$$\begin{bmatrix} 0.8660254 \\ -0.5 \end{bmatrix}$$

Em coordenadas locais polares:

$$\begin{bmatrix} 1 \\ -0.52359878 \end{bmatrix}$$

Assim, como $1 < 3$ e $-1.0471975512 < -0.52359878 < 1.0471975512$, o **objeto neste ponto pode ser detectado.**

Caso 2: (1, 0)

Em coordenadas locais cartesianas:

$$\begin{bmatrix} -1 \\ \\ -1.73205081 \end{bmatrix}$$

Em coordenadas locais polares:

$$\begin{bmatrix} 2 \\ \\ -2.0943951 \end{bmatrix}$$

Assim, como $-1.0471975512 \not< -2.0943951 < 1.0471975512$, **o objeto neste ponto não pode ser detectado.**

Caso 3: (3, 4)

Em coordenadas locais cartesianas:

$$\begin{bmatrix} 2.73205081 \\ \\ 0.73205081 \end{bmatrix}$$

Em coordenadas locais polares:

$$\begin{bmatrix} 2.82842712 \\ \\ 0.26179939 \end{bmatrix}$$

Assim, como $2.82842712 < 3$ e $-1.0471975512 < 0.26179939 < 1.0471975512$, **o objeto neste ponto pode ser detectado.**

Caso 4: (5/2, 5)

Em coordenadas locais cartesianas:

$$\begin{bmatrix} 2.79903811 \\ \\ 1.84807621 \end{bmatrix}$$

Em coordenadas locais polares:

$$\begin{bmatrix} 3.35410197 \\ \\ 0.58354994 \end{bmatrix}$$

Assim, como $3.35410197 \not< 3$, o **objeto neste ponto não pode ser detectado.**

Questão 5

Suponha que a partícula $\bar{P}$ foi produzida na etapa 1 do FastSlam e contém a seguinte hipótese sobre o estado do robô:

$$X = \begin{bmatrix} 4 \\ 3 \\ -\pi/2 \end{bmatrix}$$

O mapa da partícula a partir da qual $\bar{P}$ foi gerada é:

$$M = \left\{ \left\{ \begin{bmatrix} 2 \\ 1 \end{bmatrix}, \begin{bmatrix} 1/9 & 0 \\ 0 & 1 \end{bmatrix} \right\} \right\}, \left\{ \left\{ \begin{bmatrix} 3 \\ -1 \end{bmatrix}, \begin{bmatrix} 5/9 & 4/9 \\ 5/9 & 4/8 \end{bmatrix} \right\}, \left\{ \begin{bmatrix} 5/2 \\ 2 \end{bmatrix}, \begin{bmatrix} 1/9 & 0 \\ 0 & 1 \end{bmatrix} \right\} \right\}$$

O robô realiza a observação de 3 objetos: $z_1 = [3, 0], z_2 = [4, -0.646]$ e $z3 = [\sqrt{2}, \pi/8]$. Estas observações são feitas em coordenadas polares ρ, ψ. Lembre-se que as covariâncias do erro de medição são de $0,25^2$ em ρ e $0,0436^2$ em ψ. Suponha neste caso que a probabilidade de se observar um novo objeto α vale 0,05.

a) Calcule para cada observação a posição equivalente no sistema de coordenadas locais l, r.

b) Determine os valores de $p_{i,j}$ como definidos na equação (5) para $i = 1, 2, 3$ e $j = 1, 2, 3, 4$.

c) Determine as associações de cada observação.

d) Calcule o mapa atualizado da partícula $\bar{P}$.

e) Calcule o peso total da partícula $\bar{P}$

***Solução**:*

a) Transformando as observações de locais polares para locais cartesianas , através da Equação 14, ficamos com:

$$Z_{lc} = \begin{bmatrix} z_{1_{lc}} \\ \\ z_{2_{lc}} \\ \\ z_{3_{lc}} \end{bmatrix} = \begin{bmatrix} [3,\ 0]^T \\ \\ [3.19399268,\ -2.40798895]^T \\ \\ [1.30656296,\ 0.541196]^T \end{bmatrix}$$

b) Temos dois objetos atualizados, um novo objeto e um objeto que está no mapa mas não foi detectado neste passo. A matriz de pesos p_{ij}, para o passo em questão, é dada por:

$$\begin{bmatrix} 4.52984211e-08 & 2.36284194e-01 & 1.01550775e-05 \\ \\ 1.22385832e-01 & 1.42121782e-09 & 3.27372620e-03 \\ \\ 1.07703834e-12 & 1.36139070e-04 & 2.08799560e-08 \\ \\ \end{bmatrix}$$

c) Assim, como $2.36284194e-01 = p_{1,2} = \max(p_{i,j}) > \alpha = 0.05\%$ o objeto 2 é associado à observação 1. Após a primeira associação, a linha e coluna correspondentes ao $\max(p_{i,j})$ são retiradas da matriz, que passa a ter uma dimensão a menos. Nessa etapa é retirada a linha 1 e coluna 2 correspondentes ao elemento $p_{1,2}$. O processo continua para a próxima associação: como $1.22385832e-01 = p_{2,1} = \max(p_{i,j}) > \alpha = 0.05\%$, o objeto 1 é associado à observação 2, em seguida a linha 2 e coluna 1 são retiradas. Por fim, como $\max(p_{i,j}) < \alpha = 0.05\%$ o objeto 3 não é associado à nenhuma observação. A observação 3 é associada à um novo objeto 4. Sintetizando, têm-se:

$$z_1 \longrightarrow \text{Objeto 2}$$
$$z_2 \longrightarrow \text{Objeto 1}$$
$$z_3 \longrightarrow \text{Objeto 4 - novo objeto}$$

d) O mapa atualizado da partícula $\bar{P}$ é resultado de:

- **para objetos com associação (*objetos 1* e *2* nesse caso):** atualizar (corrigir) o estado do objeto, realizando um passo de *update* do EKF.
- **para objetos sem associação (*objeto 3* nesse caso):** copiar os valores do estado anterior.
- **para objetos novos (*objeto 4* nesse caso):** inicializar o estado do objeto com: μ_j tal que $[\mu_j,\ 1] = A_g^{lc} \cdot [l,\ r,\ 1]^T$ onde o objeto está descrito em coordenadas cartesianas locais $l,\ r$ através da Equação 14; e $\Sigma_j = Q_g$ da Equação 8.

Assim, com número aproximados para três casas decimais, temos o mapa atualizado:

$$M = \left\{ \left\{ \begin{bmatrix} 1.716 \\ -0.099 \end{bmatrix}, \begin{bmatrix} 0.030 & 0.010 \\ 0.010 & 0.047 \end{bmatrix} \right\}, \left\{ \begin{bmatrix} 3.973 \\ -0.014 \end{bmatrix}, \begin{bmatrix} 0.015 & 0.006 \\ 0.008 & 0.033 \end{bmatrix} \right\}, \right.$$
$$\left. \left\{ \begin{bmatrix} 5/2 \\ 2 \end{bmatrix}, \begin{bmatrix} 1/9 & 0 \\ 0 & 1 \end{bmatrix} \right\}, \left\{ \begin{bmatrix} 4.541 \\ 1.693 \end{bmatrix}, \begin{bmatrix} 0.012 & -0.020 \\ -0.021 & 0.054 \end{bmatrix} \right\} \right\}$$

e) O peso total é simplesmente a multiplicação dos pesos computados para cada objeto. Assim, o peso w total da partícula é dado por:

$$\begin{aligned} w &= \prod_i p_{i,c(i)} \\ &= p_{1,2} \cdot p_{2,1} \cdot p_{3,4} \\ \therefore w &= 0.9140482050108684 \end{aligned}$$

Questão 6

Implemente uma previsão simples de estado utilizando a linguagem de programação que achar mais adequada (Sugestão: Python com o pacote `numpy`). Neste passo você deve estimar o estado recursivamente com a equação:

$$\bar{\mu}_t = F\left(\mu_{t-1}, u_t\right)$$

Como o filtro de partículas é não-paramétrico, não é necessário calcular covariâncias. Para o estado inicial use:

$$\mu_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

Neste problema, $\Delta t = 1$.

O seu programa deve processar uma entrada no formato descrito pela seção "Dados". Processe os parâmetros u_t no arquivo *csv* anexado neste enunciado. Plote os valores de x_t, y_t de $\bar{\mu}_t$

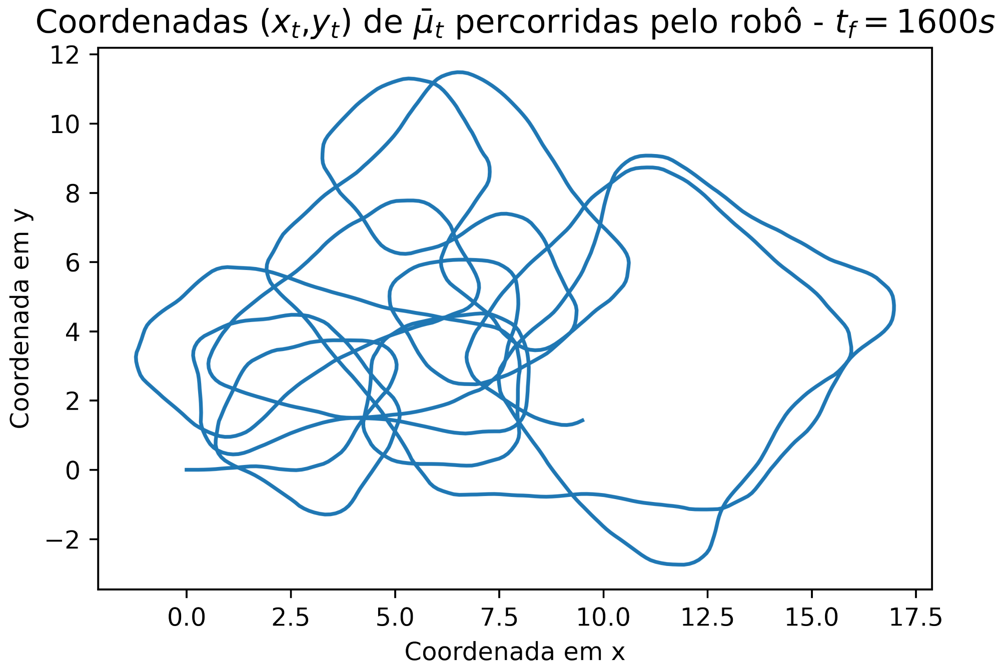
no plano x, y.

Solução: Utilizando a modelagem realizada no relatório anterior (realizada a devida correção pontuada pelo professor), têm-se a seguinte função matricial $F(\mu_{t-1}, u_t)$:

$$X_{d_t} = F(X_{t-1}, u_{t-1}) = \begin{bmatrix} X_{1_{t-1}} \\ \\ X_{2_{t-1}} \\ \\ X_{3_{t-1}} \end{bmatrix} + \begin{bmatrix} u_{1_{t-1}} \cdot \cos(X_{3_{t-1}}) \cdot \cos(u_{2_{t-1}}) \\ \\ u_{1_{t-1}} \cdot \sin(X_{3_{t-1}}) \cdot \cos(u_{2_{t-1}}) \\ \\ \frac{u_{1_{t-1}} \cdot \tan(\phi_{t-1})}{l} \end{bmatrix} \cdot \Delta t \tag{14}$$

onde $X = [x,\ y,\ \theta]^T$ é o vetor de estado do robô, X_d é a parcela determinística da extrapolação do estado para o próximo passo (única usada nessa questão) e $u = [v,\ \phi]^T$ é o vetor de ação de controle.

Com isso, processa-se os dados fornecidos, obtendo valores das coordenadas de $\bar{\mu}$ ($\bar{X}$ pela notação adotada na modelagem) para cada instante de tempo, gerando a seguinte figura:



Questão 7

Implemente o FastSlam.

O passo de previsão deve para cada partícula somar o estado previsto a uma perturbação aleatória.

Considere para isso que ao final do arco de circunferência percorrido pelo robô soma-se um deslocamento aleatório longitudinal, um deslocamento radial e um deslocamento angular. Estes três deslocamentos são independentes, Gaussianos de média nula e covariâncias $s^2_{l_t}, s_{r^2_t}$ e $s_{\theta^2_t}$ respectivamente. As covariâncias $s_{l^2_t}$ e $s_{r^2_t}$ são de medidas de perturbação em um referencial ideal que faz um ângulo de $\hat{\theta}_t$ com o eixo x. Esta orientação seria a do eixo longitudinal do veículo se este descrevesse um arco perfeito de circunferência no instante t de modo que $\hat{\theta}_t = F(X_{t-1}, u_t)_3$ (ou seja, a orientação determinada pela função F sem ruído). No entanto, a matriz R, de covariância do ruído do processo, está escrita no referencial *Global*. Neste referencial, as perturbações em x e y podem não ser independentes.

Os valores das covariâncias $s^2_{l_t}, s^2_{r_t} e s^2_{\theta_t}$ dependem do parâmetro v_t (o primeiro coeficiente de

u_t) de acordo com as seguintes expressões:

$$s_{l_t}^2 = \left(\frac{v_t}{6}\right)^2$$
$$s_{r_t}^2 = \left(\frac{v_t}{12}\right)^2$$
$$s_{\theta_t}^2 = \left(\frac{v_t}{8l}\right)^2$$

Onde l é a distância entre os eixos do veículo e vale 0,3.

Nota: Este é exatamente o mesmo modelo dinâmico do exercício de EKF.

Sugestão: Lembre-se que os ruídos nas coordenadas globais não são independentes, não basta gerar 3 variáveis aleatórias independetes para cada partícula. Em Python existe a função `numpy.random.multivariate_normal` para gerar variáveis aleatórias Gaussianas multidimensionais. Você pode alternativamente gerar as perturbações no referencial orientado de acordo com $\hat{\theta}_t$, onde são resultado de 3 variáveis independentes, e transformá-las no referencial global.

Use $0,05\%$ como valor de α e ao menos 200 partículas.

Durante os passos de correspondência e atualização, novos objetos podem ser criados nos mapas de cada partícula. A medida que o processo progride o mapa tende a ficar excessivamente poluído.

Para mitigar este efeito, adote um procedimento de limpeza. A cada objeto no mapa de cada partícula está associado um contador, inicializado com 1.

A cada vez que um objeto no mapa é associado a uma observação, este contador deve ser incrementado.

A cada vez que um objeto deveria ter sido observado (vide questão 4) e não o foi, este contador deve ser decrementado. Quando o contador associado a um objeto chegar a zero, este deve ser retirado do mapa.

Plote a trajetória no plano x, y do robô, usando para isso a média dos estados das partículas em S_t.

Plote o mapa final.

Para tanto, sorteie aleatoriamente uma partícula do valor final de S_t e plote os objetos para os quais o contador vale mais do que 10.

Observação: A determinação de associações de acordo com valores da equação (5) tem complexidade bilinear em N (número de observações) e M (número de partículas no mapa). Isso significa que, a medida que a quantidade de pontos no mapa cresce, também cresce o tempo para corrigir um mapa. Existem estruturas de dados que mitigam este efeito, como as $k-d$ trees. Embora você não precise se preocupar com tais estruturas neste exercício, tenha em mente que a busca exaustiva por associações vai deixar o algoritmo lento. Uma solução em Python, sem otimizações específicas, leva uma hora para processar todas as entradas em um processador Intel Xeon a $2,30$GHz para 200 partículas.

Solução: Novamente, utilizando o desenvolvimento do modelo dinâmico realizado no relatório anterior, a matriz R_g, de covariância do ruído do processo, escrita no referencial *Global*, é dada por:

$$R_g = M \cdot R_l \cdot M^T$$

onde R_l é a matriz de covariância no referencial local dada pelo enunciado:

$$R_{l_{t-1}} = \begin{bmatrix} \left(\frac{v_{t-1}\cdot\Delta t}{6}\right)^2 & 0 & 0 \\ 0 & \left(\frac{v_{t-1}\cdot\Delta t}{12}\right)^2 & 0 \\ 0 & 0 & \left(\frac{v_{t-1}\cdot\Delta t}{8\cdot l}\right)^2 \end{bmatrix}$$

e a matriz M é dada por:

$$M_{t-1} = \begin{bmatrix} \cos(\theta_{d_{t-1}}) & -\sin(\theta_{d_{t-1}}) & 0 \\ \sin(\theta_{d_{t-1}}) & \cos(\theta_{d_{t-1}}) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

FastSLAM com 10 partículas

Rodamos o *FastSLAM* com 10 partículas. O caminho médio do robô está ilustrado na Figura 2.

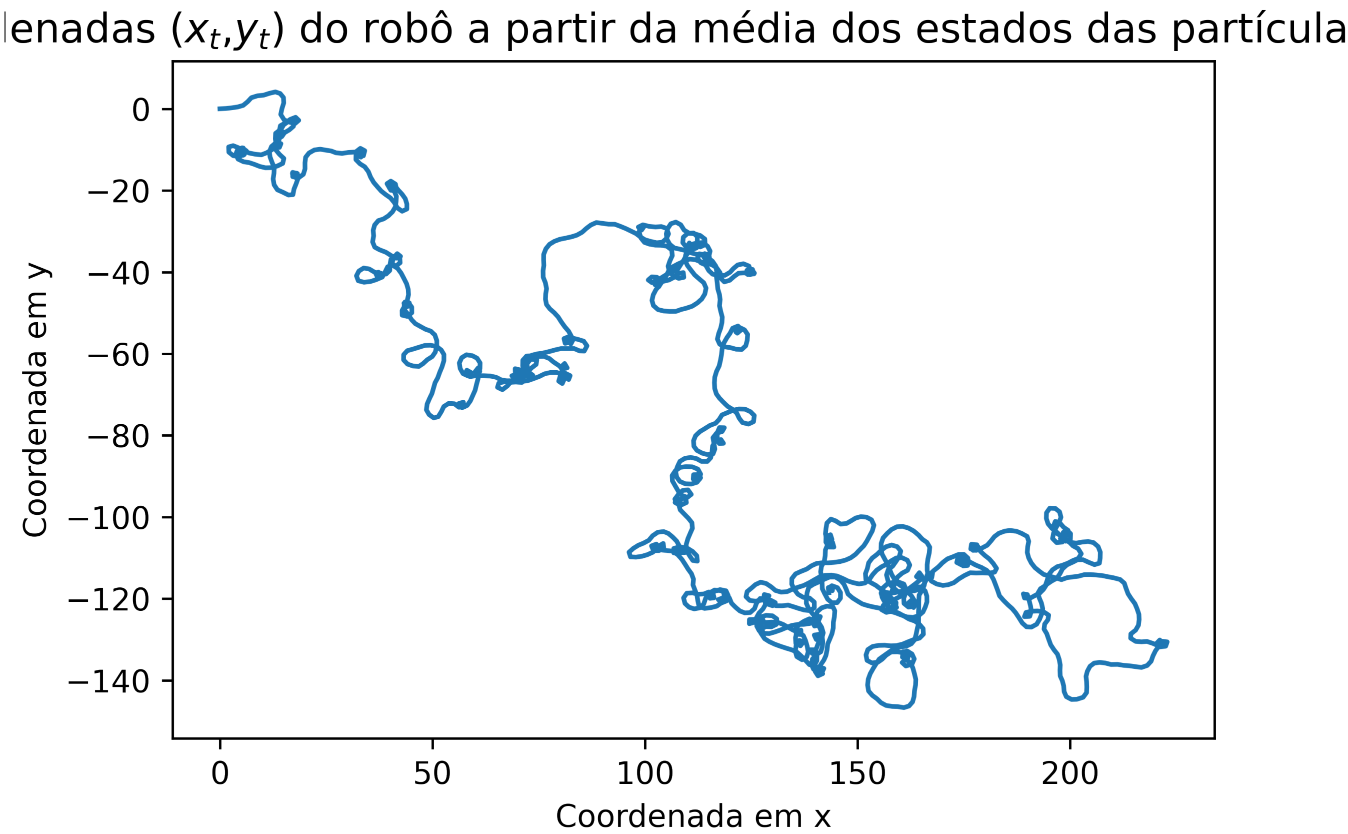


Figura 2: Caminho médio do robô, utilizando 10 partículas

Sorteamos uma partícula e plotamos os objetos. A posição dos objetos para esta partícula está ilustrada na Figura 3.

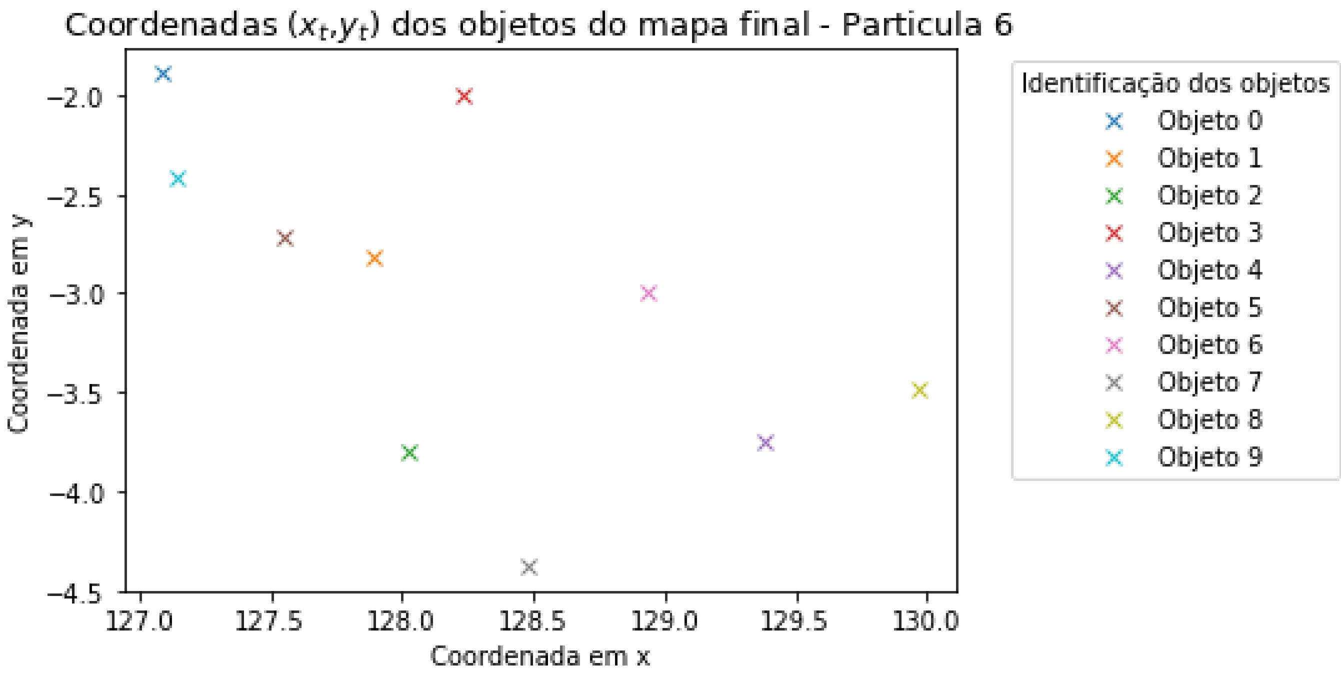


Figura 3: Posição dos objetos de uma partícula sorteada, utilizando 10 partículas

ep2

August 3, 2021

[1]:
```
import numpy as np
import numpy.linalg as LA
from tqdm import tqdm
```

[116]:
```
srho = 0.25**2
spsi = 0.0436**2
alpha = 0.05**0.01
l = 0.3
dt = 1
```

0.1 Questão 1

[3]:
```
def lp2lc(coords) -> np.array:
    rho, psi = coords
    return np.array([
        [rho*np.cos(psi)],
        [rho*np.sin(psi)],
    ])

def Q_lp2lc(obs) -> np.array:
    rho, psi = obs
    return np.array([
        [rho**2 * spsi * np.sin(psi)**2 + srho * np.cos(psi)**2, 0.5*(-rho**2 *␣
↪spsi + srho)*np.sin(2*psi)],
        [0.5*(-rho**2 * spsi + srho)*np.sin(2*psi), rho**2 * spsi * np.
↪cos(psi)**2 + srho * np.sin(psi)**2]
    ])
```

0.2 Questão 2

[4]:
```
def R_lc2g(theta: float) -> np.array:
    '''
    Rotation matrix from Local Cartesiano to Global
    '''
    return np.array([
            [ np.cos(theta),-np.sin(theta)],
```

```
            [ np.sin(theta), np.cos(theta)]
    ])

def A_lc2g(robot: np.array) -> np.array:
    '''
    Homogenous matrix from Local Cartesiano to Global
    '''
    x, y, theta = robot
    return np.array([
        [ np.cos(theta), -np.sin(theta), x],
        [ np.sin(theta), np.cos(theta), y],
        [0, 0, 1]
    ])

def lc2g(coords_lc: np.array, robot: np.array) -> np.array:
    x_lc, y_lc = coords_lc
    return (A_lc2g(robot) @ np.array([[x_lc[0]], [y_lc[0]], [1]]))[:2]

def Q_lc2g(theta: float, Q_lc: np.array) -> np.array:
    return R_lc2g(theta) @ Q_lc @ R_lc2g(theta).T
```

0.3 Questão 3

[5]:
```
def g2lc(coords_g: np.array, robot: np.array) -> np.array:
    A_g2lc = LA.inv(A_lc2g(robot))
    gx, gy = coords_g
    return (A_g2lc @ np.array([[gx], [gy], [1]]))[:2]

def C(theta):
    return R_lc2g(theta).T

def lc2lp(coords_lc: np.array) -> np.array:
    l, r = coords_lc
    return np.array([
        [np.sqrt(l**2+r**2)],
        [np.arctan2(r,l)]
    ])
```

0.4 Questão 4

[6]:
```
def check_sensor(x: float, y: float, theta_robot: float = np.pi/6) -> bool:
    print(f'X: {x}    Y: {y}')
    D_MAX = 3
    ANG_MAX = np.pi/3
    x_robot = 1
    y_robot = 2
```

```
    l, r = g2lc([x,y], [x_robot, y_robot, theta_robot])
    rho, psi = lc2lp([l,r])
    print(rho,psi)

    if rho <= D_MAX and (-ANG_MAX <= psi) and (psi <= ANG_MAX):
        print(True,'\n')
    else:
        print(False,'\n')
```

[7]:
```
check_sensor(2,2)
```

```
X: 2   Y: 2
[[1.]] [[-0.52359878]]
True
```

0.5 Questão 5

[38]:
```
class EKF_PARTICLE:
    def __init__(self, landmarks, robot_t0):
        self.landmarks = landmarks
        self.robot = robot_t0

    @staticmethod
    def Lj(land, obs, robot_t):
        '''
        land = [mu, sigma]
        mu = [x_bar, y_bar]
        obs = [rho, psi]
        '''
        return C(robot_t[2]) @ land['sigma'] @ C(robot_t[2]).T + Q_lp2lc(obs)

    def compute_W(self, observations_t, robot_t):
        '''
        Observations_t = todas as obs de um mesmo timestamp
        Robot_t = estado do robo no timestamp
        '''
        m = len(self.landmarks)
        n = len(observations_t)
        if m == 0:
            #necessario para o caso inicial, onde nao existem landmarks
            self.W = np.zeros((n,1))
        else:
            W = np.zeros((n,m))
            for i, obs in enumerate(observations_t):
                for j, land in enumerate(self.landmarks):
                    d = lp2lc(obs) - g2lc(land['mu'], robot_t)
```

```
                L = self.Lj(land, obs, robot_t)
                W[i][j] = 1 / np.sqrt(LA.det(2*np.pi*L)) * np.exp(-(d.T @ LA.
↪inv(L) @ d)/2)

        self.W = W

    def attribute_landmarks(self, observations_t, robot_t):
        W = self.W.copy()
        c = []
        new_landmarks = []
        n, m = W.shape
        while True:
            i,j = np.unravel_index(np.argmax(W), W.shape)
            max_W = np.max(W)
            if max_W == -1:
                break
            elif max_W >= alpha:
                c.append({'observation':i, 'landmark': j, 'weight': max_W})
                W[i,:] = -1
                W[:,j] = -1
            else:
                #create new landmark
                obs = observations_t[i]
                initial_mu = lc2g(lp2lc(obs), robot_t).T[0]
                initial_sigma = Q_lc2g(robot_t[2], Q_lp2lc(obs))
                new_landmarks.append({'mu':initial_mu, 'sigma':initial_sigma,␣
↪'count':1})
                c.append({'observation':i, 'landmark': m+len(new_landmarks)-1,␣
↪'weight': alpha})
                W[i,:] = -1

        self.attributions = c
        self.new_landmarks = new_landmarks
        return c, new_landmarks

    def Kalman_Gain(self,land, robot_t, obs):
        K = land['sigma'] @ C(robot_t[2]).T @ LA.inv(self.Lj(land, obs, robot_t))
        return K

    def update_landmarks(self, observation_t, robot_t):
        landmarks_seen = []
        for i, att in enumerate(self.attributions):
            if att['landmark'] >= len(self.landmarks):
                # if new landmark
                pass
            else:
                land = self.landmarks[att['landmark']]
```

```python
                obs = observation_t[att['observation']]
                d = lp2lc(obs) - g2lc(land['mu'], robot_t)
                K = self.Kalman_Gain(land,robot_t,obs)
                land['mu'] += (K @ d).T[0]
                land['sigma'] = (np.eye(K.shape[0]) - K @ C(robot_t[2])) @ land['sigma']
                self.landmarks[att['landmark']] = land

                # só preciso atualizar o count dos landmarks antigos
                landmarks_seen.append(att['landmark'])

        idx2pop = []
        for i, land in enumerate(self.landmarks):
            # atualiza counters dos landmarks
            if i not in landmarks_seen:
                land['count'] -= 1
                if land['count'] == 0:
                    idx2pop.append(i)
            else:
                land['count'] += 1

        # deletar landmarks com count=0
        for i in sorted(idx2pop, reverse=True):
            del self.landmarks[i]

    def compute_particle_weight(self):
        total_weight = 1
        for el in self.attributions:
            total_weight *= el['weight']
        self.particle_weight = total_weight

    def extrapolate(self, u_t):
        """
        Extrapola a posição da partícula
        u_t : controle daquele timestamp
        """
        self.robot = np.random.multivariate_normal(F(self.robot, u_t), R(self.robot, u_t))

    def fastSLAM_t(self, observation_t):
        self.compute_W(observation_t, self.robot)
        self.attribute_landmarks(observation_t, self.robot)
        self.update_landmarks(observation_t, self.robot)
        self.landmarks += self.new_landmarks
        self.compute_particle_weight()

        return self.attributions, self.landmarks
```

[117]:
```python
Z = [[3, 0],
     [4, -0.646],
     [np.sqrt(2), np.pi/8]]
Z = np.array([Z])
robot_t = np.array([4, 3, -np.pi/2])
landmarks = [
    [[2.,1], [[1/9,0],
              [0,1.]]],
    [[3.,-1], [[5/9,4/9],
               [5/9,4/8.]]],
    [[5/2,2], [[1/9,0],
              [0,1.]]],
]
# aqui os landmarks sao inicializados como 2 apenas para facilitar o codigo e␣
↪possibitar
# acesso ao mapa atualizado incluindo o landmark que nao foi atribuido a nenhuma␣
↪observacao
landmarks = [{'mu':np.array(l[0]), 'sigma':np.array(l[1]), 'count':2} for l in␣
↪landmarks]

ekf_q5 = EKF_PARTICLE(landmarks, robot_t)
attributions, landmarks = ekf_q5.fastSLAM_t(Z[0])
ekf_q5.compute_particle_weight()
```

[118]:
```python
f'Peso da particula {ekf_q5.particle_weight}'
```

[118]: 'Peso da particula 0.9140482050108684'

[12]:
```python
print('Mapa atualizado')
for i, land in enumerate(landmarks):
    print(f'Landmark {i}')
    print(f'mu: {land["mu"]}')
    print(f'sigma: {land["sigma"]}')
```

```
Mapa atualizado
Landmark 0
mu: [ 1.71630326 -0.09871928]
sigma: [[0.03038221 0.01066319]
 [0.01066319 0.04700132]]
Landmark 1
mu: [ 3.97314494 -0.01421739]
sigma: [[0.01491948 0.00631884]
 [0.00789855 0.03275703]]
Landmark 2
mu: [2.5 2. ]
sigma: [[0.11111111 0.        ]
 [0.         1.        ]]
```

```
Landmark 3
mu: [4.5411961  1.69343704]
sigma: [[ 0.01239805 -0.02075291]
 [-0.02075291  0.05390387]]
```

0.6 Questão 6

[13]:
```python
import pandas as pd
import matplotlib.pyplot as plt
```

[14]:
```python
data = pd.read_csv('./data/valoresEP2.csv', usecols=np.arange(14), header=None).
 ↪to_numpy()
control_hist = data[:,:2]
observations = []
for observation_t in data[:,2:]:
    observation_t = observation_t[~np.isnan(observation_t)]
    observation_t = observation_t.reshape(int(observation_t.shape[0]/2), 2)
    observations.append(observation_t)
```

[15]:
```python
def F(X_prev: np.array, u_prev: np.array, dt=1) -> np.array:
    '''
    Calculada analiticamente.
    '''
    l=0.3
    return np.array([
        X_prev[0] + u_prev[0] * np.cos(u_prev[1]) *  np.cos(X_prev[2]) * dt,
        X_prev[1] + u_prev[0] * np.cos(u_prev[1]) *  np.sin(X_prev[2]) * dt,
        X_prev[2] + u_prev[0] * dt * np.tan(u_prev[1])/ l
    ])
```

[16]:
```python
X = [[0,0,0]]
for i, u in enumerate(control_hist):
    X.append(F(X[-1], u))
```

[17]:
```python
X = np.array(X)
```

[76]:
```python
plt.plot(X[:,0], X[:,1])
plt.ylabel('Coordenada em y')
plt.xlabel('Coordenada em x')
plt.title(r'Coordenadas ($x_t$,$y_t$) de $\bar{\mu}_{t}$ percorridas pelo robô -␣
 ↪$t_f=1600s$', fontsize=13)
plt.savefig('./images/caminhoEP2Q6.png', dpi=600)
plt.show()
```

output_22_0.png

0.7 Questão 7

[20]:
```
def R(x_prev, u):
    '''
    DE ESTADO
    '''
    cos = np.cos(x_prev[2])
    sin = np.sin(x_prev[2])
    M =  np.array([
        [cos, -sin, 0],
        [sin,  cos, 0],
        [0, 0, 1]
    ])

    sl = (dt * u[0] / 6)**2
    sr = (dt * u[0] / 12)**2
    stheta = (dt * u[0] / (8*l))**2

    # Matriz R no referencial local
    Rl = np.array([
        [sl, 0, 0],
        [0, sr, 0],
        [0, 0, stheta,]
    ])

    return M @ Rl @ M.T
```

[102]:
```
N_PARTICLES = 10


def ressample_particles(particles):
    """
    Recebe as partículas e ressampleia novas 200 com base nos pesos das␣
 ↪partículas.
    """
    weights = np.array([p.particle_weight for p in particles])
    weights /= weights.sum()
    new_particles = np.random.choice(particles,
```

```
                                           size=(N_PARTICLES,),
                                           p = weights)
    return new_particles


def particles_mean(particles):
    """
    Pega a média dos estados das partículas
    """
    return np.mean(
        np.array([p.robot for p in particles]),
        axis=0
    )


# def Q7(observations, control_history):
particles = []
robot_mean_history = []
for _ in range(N_PARTICLES):
    particle = EKF_PARTICLE(landmarks=[], robot_t0=np.zeros(3))
    particles.append(particle)

for t, observation_t in tqdm(enumerate(observations)):
    for particle in particles:
        particle.extrapolate(control_hist[t])
        particle.fastSLAM_t(observation_t)
    particles = ressample_particles(particles)
    robot_mean = particles_mean(particles)
    robot_mean_history.append(robot_mean)

# return robot_mean_history, particles
```

```
1600it [02:14, 11.90it/s]
```

[103]:
```
plt.plot(np.array(robot_mean_history)[:,0], np.array(robot_mean_history)[:,1])
plt.ylabel('Coordenada em y')
plt.xlabel('Coordenada em x')
plt.title(r'Coordenadas do robô a partir da média das partículas em $S_t$',␣
 ↪fontsize=13)
plt.savefig(f'./images/caminhoEP2Q7_N{N_PARTICLES}.png', dpi=600)
plt.show()
```

output_26_0.png

[115]:

```
random_particle = np.random.randint(10)
final_map = particles[random_particle].landmarks
for i, land in enumerate(final_map):
    plt.plot(land['mu'][0], land['mu'][1],'x', label=f'Objeto {i}')
plt.ylabel('Coordenada em y')
plt.xlabel('Coordenada em x')
plt.legend(title='Identificação dos objetos', bbox_to_anchor=(1.05, 1), loc='upper left')
plt.title(r'Coordenadas ($x_t$,$y_t$) dos objetos do mapa final'+f' - Particula {random_particle}', fontsize=13)
plt.savefig(f'./images/caminhoEP2Q7_2_n{N_PARTICLES}.png', dpi=600)
plt.show()
```

output_27_0.png