Implementação do Filtro de Kalman em Localização Robótica

Inicialmente vamos considerar um caso particular: o robô pára após realizar um percurso de comprimento desconhecido e irá se localizar nesta posição (parado).

Neste caso a única informação que dispomos sobre a pose do robô é a fornecida pela sua odometria.

Podemos implementar o Filtro de Kalman para fundir a odometria e o laser (sem utilizar o modelo cinemático do robô).

Implementação do Filtro de Kalman

Para $\Delta s = 0$ e $\Delta\theta = 0$ (robô parado):

$$G_t = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = I \qquad V_t = \begin{bmatrix} \frac{1}{2}\cos(\theta_{t-1}) & \frac{1}{2}\cos(\theta_{t-1}) \\ \frac{1}{2}sen(\theta_{t-1}) & \frac{1}{2}sen(\theta_{t-1}) \\ \frac{1}{2b} & -\frac{1}{2b} \end{bmatrix}$$

$$\overline{\Sigma}_t = \Sigma_{t-1} + \sigma_{od} V_t V_t^T$$

- σ_{od} deve refletir a imprecisão da odometria do robô.
- $V_t V_t^T$ informa como esta imprecisão se reflete nos componentes x, y e θ da pose.

Implementação do Filtro de Kalman

Fase de Predição:

Pose no instante inicial (t = 0):

```
Pose = http_get("http:// ... /motion/pose");
Pose.th = mod(Pose.th*pi/180, 2*pi);
x = Pose.x;
y = Pose.y;
th = Pose.th;
```

$$\overline{\Sigma}_t = \Sigma_{t-1} + \sigma_{odom} V_t V_t^T$$

Implementação do Filtro de Kalman

Fase de Atualização:

$$z_t = \begin{bmatrix} \sqrt{\left(Z_{x_t} - x_t\right)^2 + \left(Z_{y_t} - y_t\right)^2} \\ \arctan 2\left(Z_{y_t} - y_t, Z_{x_t} - x_t\right) - \theta_t \end{bmatrix}$$

Z_x e Z_y fornecido pelo detector de features.

$$h(\overline{\mu}_t) = \begin{bmatrix} \sqrt{\left(L_x - x_t\right)^2 + \left(L_y - y_t\right)^2} \\ \arctan 2\left(L_y - y_t, L_x - x_t\right) - \theta_t \end{bmatrix}$$

Implementação do Filtro de Kalman

$$H_t = \begin{bmatrix} -\frac{L_x - x_t}{\sqrt{q}} & -\frac{L_y - y_t}{\sqrt{q}} & 0 \\ \frac{L_y - y_t}{q} & -\frac{L_x - x_t}{q} & -1 \\ .... & & \end{bmatrix}$$

$$Q_t = \begin{bmatrix} \sigma_{ld}^2 & 0 & 0 & 0 \\ 0 & \sigma_{l\theta}^2 & 0 & 0 \\ 0 & 0 & \sigma_{ld}^2 & 0 \\ 0 & 0 & 0 & \sigma_{l\theta}^2 \\ & & & ... \end{bmatrix}$$

$$q = (L_x - x_t)^2 + (L_y - y_t)^2$$

σ_{ld} e $\sigma_{l\theta}$ refletem a precisão do laser quanto à distância e a direção.

$$K_t = \overline{\Sigma}_t H_t^T (H_t \overline{\Sigma}_t H_t^T + Q_t)^{-1}$$

$$\mu_t = P_t + K_t (z_t - h(\overline{\mu}_t))$$

$$\Sigma_t = (I - K_t H_t) \overline{\Sigma}_t$$

Implementação do Filtro de Kalman

Após o cômputo de $\mu_t = [x\ y\ th]^T$ atualiza-se a pose do robô (esta atualização não causará a movimentação do robô):

```
NovaPose.x = x;
NovaPose.y = y;
NovaPose.th = th*180/pi;
http_put("http:// ... /motion/pose", NovaPose);
```

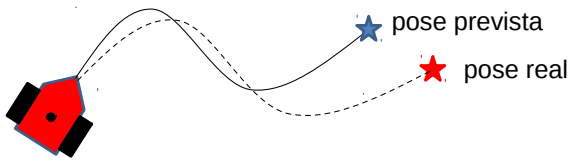
O processo se repete até um certo número de iterações ou até que não ocorra mais progresso na atualização da pose do robô (por exemplo, quando $\det(\Sigma_t) - \det(\Sigma_{t-1}) < \eta$).

OBS:
- Note que a pose do robô é suprida uma única vez pela odometria.
- Caso a pose inicial tenha um erro cumulativo este erro vai sendo cancelado pelo Filtro de Kalman.

Implementação do Filtro de Kalman

Consideremos agora o robô em movimento. Vamos supor uma aplicação realizando o controle do robô (movimentação) e outra realizando a localização. Estas duas aplicações não interagem entre si.

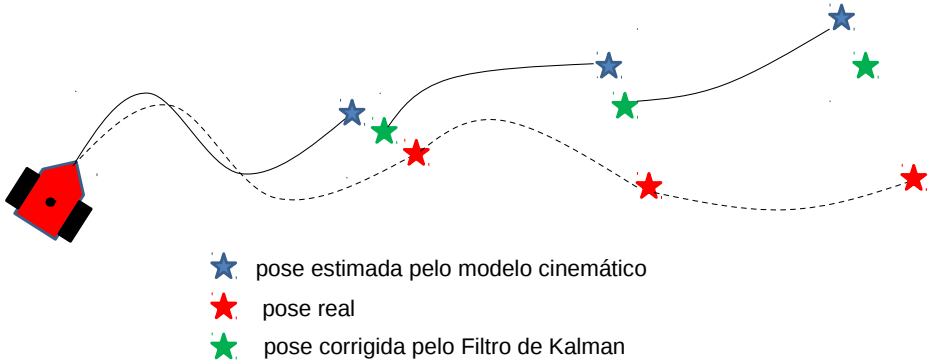
É possível substituir a odometria pelo modelo cinemático do robô e utilizar o Filtro de Kalman Estendido como vimos anteriormente. Entretanto, é muito provável o Filtro de Kalman divergir pelo descompasso entre o modelo cinemático (linearizado !) e a locomoção real do robô.



Implementação do Filtro de Kalman

A detecção de features se dá com base na posição prevista, mas as leituras do laser ocorrem na posição real.

Isto causa divergência do Filtro de Kalman.



Implementação do Filtro de Kalman

Outro ponto de divergência do filtro de Kalman é a temporização:
Δs e $\Delta\theta$ são calculados em função das velocidades das rodas e Δt.

$$\Delta s \approx \frac{V_d + V_e}{2}\Delta t \qquad \Delta\theta \approx \frac{Vd - Ve}{2b}\Delta t$$

Entretanto, o relógio do computador onde o algoritmo de localização executa não é o mesmo relógio do robô.

Dois ajustes são necessários, um na fase de predição e um na fase de atualização.

Implementação do Filtro de Kalman

tic()

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

$$\bar{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + V_t \Sigma_{\Delta_t} V_t^T + R_t$$

sleep(Δt - toc())

Tempo para computar a fase de atualização.

tic()

$$K_t = \bar{\Sigma}_t H_t^T (H_t \bar{\Sigma}_t H_t^T + Q_t)^{-1}$$

$$\mu_t = \bar{\mu}_t + K_t(z_t - h(\bar{\mu}_t))$$

$$\Sigma_t = (I - K_t H_t)\bar{\Sigma}_t$$

dt = toc()

Estima a nova pose após dt.

Implementação do Filtro de Kalman

Se o robô dispõe de odometria, podemos fundir a odometria com o laser com o robô em movimento.

Neste caso, ainda utilizamos a modelo cinemático para computar a propagação de erros na fase de predição.

Para prever a matriz de covariança da pose, necessitamos de Δs e $\Delta\theta$. Estes valores são obtidos a partir da velocidade individual das rodas (recurso */motion/vel2*):

$$\Delta s \approx \frac{V_d + V_e}{2}\Delta t \qquad \Delta\theta \approx \frac{Vd - Ve}{2b}\Delta t$$

Implementação do Filtro de Kalman

Fase de Predição:

```
Pose = http_get("http:// ... /motion/pose");
Pose.th = mod(Pose.th*pi/180, 2*pi);
```

$$\mu_t = \begin{vmatrix} Pose.x \\ Pose.y \\ Pose.th \end{vmatrix}$$

$$\overline{\Sigma}_t = G_t \Sigma_{t-1} G_t^T + V_t \Sigma_{\Delta_t} V_t^T + R_t$$

Implementação do Filtro de Kalman

Fase de Atualização:

$$K_t = \overline{\Sigma}_t H_t^T (H_t \overline{\Sigma}_t H_t^T + Q_t)^{-1}$$

$$dP_t = K_t (z_t - h(\overline{\mu}_t))$$

$$\mu_t = P_t + dP_t$$

$$\Sigma_t = (I - K_t H_t)\overline{\Sigma}_t$$

Para atualizar a pose do robô, fazemos:

P**.dx** = dP(1);
P**.dy** = dP(2)
P**.dth** = dP(3)*180/pi;
http_post("htttp:// ... /motion/pose", P);

Implementação do Filtro de Kalman

Algumas questões de ordem prática:

1. É recomendável adquirir as leituras do laser o mais próximo possível da leitura da pose do robô. Para tal podemos definir um grupo com estes dois recursos e ler (*http_get*) estes recursos conjuntamente.

2. É recomendável, caso seja possível, executar a fusão da odometria e laser (ou seja, aplicar o Filtro de Kalman) quando o robô se move em linha reta ($V_d = V_e$).

3. Se a aplicação permitir, é recomendável de tempos em tempos fazer uma ação de localização com o robô parado.

Mapeamento

Mapeamento é o processo pelo qual o robô, utilizando seus sensores proprioceptivos (odometria) e estereoceptivos (laser, sonar, câmeras), deriva o mapa do ambiente. Este mapa pode ser:
• contínuo: retas representando obstáculos intransponíveis;
• discreto: células livres e ocupada (occupancy grid);
• composto de landmarks;
• topológico: lugares interconectados.

Para mapear o ambiente o robô necessita se locomover através do ambiente, permitindo assim que seus sensores cubram toda a área de interesse.

O mapeamento depende da precisão da odometria do robô pois imprecisão na pose gera distorções no mapa.

Mapeamento

Um mapeamento contínuo pode ser obtido com o algoritmo Split-And-Merge após o robô executar um "passeio" pelo ambiente. A precisão do sensor e da odometria irá ditar a precisão das retas e dos landmarks.

Um mapeamento topológico pode ser obtido via detecção de elementos de interconexão de ambiente (passagens de ligação e portas). Como podemos detectar uma passagem ou porta ?

No mapeamento discreto o ambiente é segmentado em células de tamanho fixo (da ordem do tamanho do robô). Cada célula possui uma probabilidade (ou outro indicativo) de seu estado que pode ser ocupada (por obstáculos), vazia (área navegável) ou indeterminada.

Mapeamento

Occupancy Grids (A. Elfes, 1989)

Esta técnica de mapeamento supõe que a odometria do robô é precisa, mas seus sensores estereoceptivos estão sujeitos a erros. Supõe também que o ambiente mapeado se mantém estático durante o mapeamento.

O ambiente é discretizado em células. Cada célula possui duas probabilidades associadas: p(V) e p(O), respectivamente, a probabilidade da célula estar vazia (livre de obstáculos) e estar ocupada por obstáculos. Como os estados vazia e ocupada são os únicos estados possíveis para uma célula: p(V) + p(O) = 1.

Mapeamento

Comumente para espaço amostral binário (como livre/ocupada), define-se a relação que expressa a chance (odds) de um estado ocorrer em relação ao outro:

$$l(x) = \log \frac{p(x)}{p(\neg x)} = \log \frac{p(x)}{1 - p(x)}$$

Onde x = O (ocupada) e ¬x = V (vazia). Considerando que inicialmente não temos nenhuma informação sobre o mapa, p(O) = p(V) = 0.5, ou l(O) = l(V)= 0. Valores positivos de l indicam que a chance da célula estar ocupada é maior que a chance dela estar livre. Valores negativos indicam o oposto.

Supondo que efetuamos uma leitura do(s) sensor(es) z_t desejamos computar $l(x|z_t)$, ou seja, atualizar as chances das células estarem ocupadas.

Mapeamento

Para cada célula do mapa: $l(x \mid z_t) = \log \frac{p(x \mid z_t)}{1 - p(x \mid z_t)}$

Aplicando a Regra de Bayes (Filtro de Bayes Binário) obtem-se:

$$l_t(x) = l_{t-1}(x) + \log \frac{p(x \mid z_t)}{1 - p(x \mid z_t)} - l_0(x)$$

Onde $p(x|z_t)$ é a probabilidade da célula estar ocupada dado que o sensor leu z_t (modelo inverso do sensor).

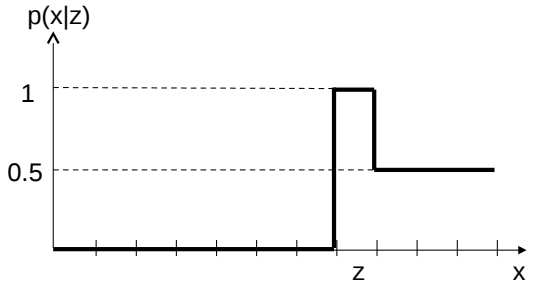
Computado $l(x)$ pode-se recuperar p(x): $p(x) = 1 - \frac{1}{1 + \exp(l(x))}$

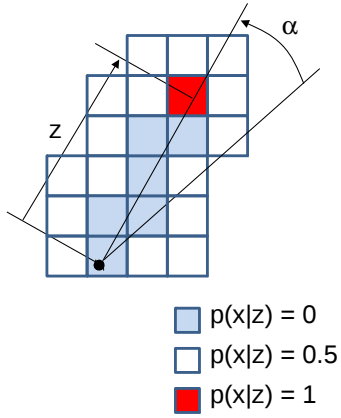
OBS: Se $l(x) \rightarrow \infty$, $p(x) \rightarrow 1$ e se $l(x) \rightarrow -\infty$, $p(x) \rightarrow 0$

Mapeamento

Como computar o modelo inverso do sensor?

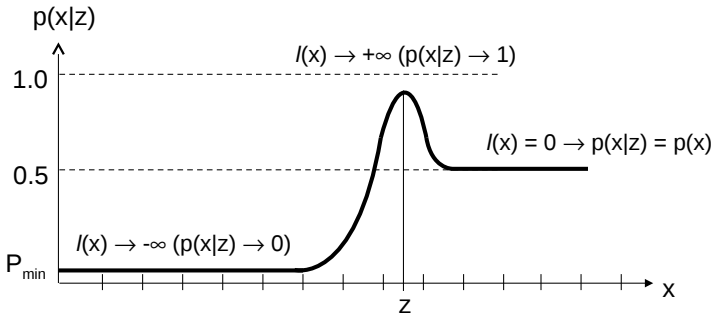
Para um sensor ideal (livre de erros) o modelo inverso é dado por:





Mapeamento

Modelo inverso de sensor com erro na determinação da distância:

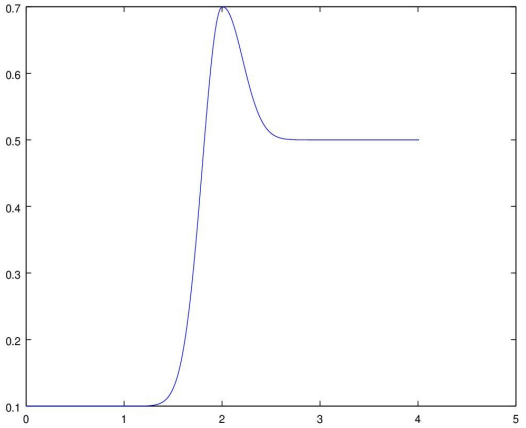


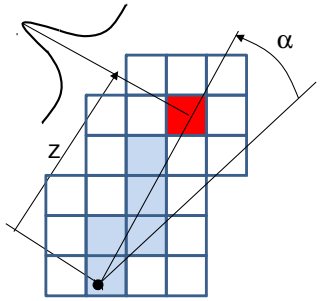
$$p(x \mid z) = P + \left(\frac{k}{\sigma\sqrt{2\pi}} + 0.5 - P\right)\exp\left(-\frac{1}{2}\left(\frac{x-z}{\sigma}\right)^2\right)$$

$$\begin{cases} P = P_{min}, \text{ se } 0 \le x \le z \\ 0.5 \text{ se } x \ge z \end{cases}$$

k é um fator de ponderação para o pico da função ficar entre 0.5 e 1.0.

Mapeamento

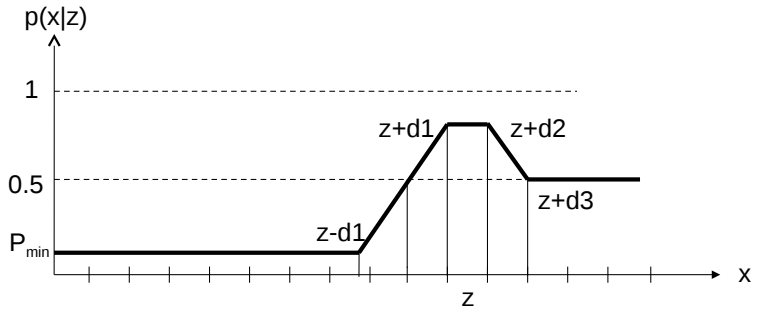




OBS: Esta função não é uma PDF !

Mapeamento

Alternativamente, podemos definir um modelo inverso de sensor de forma mais simples:



Mapeamento

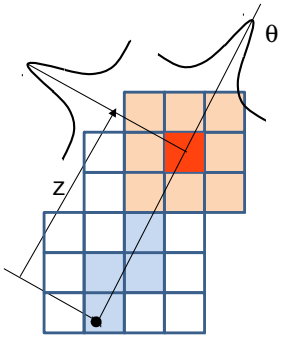
Se o sensor possuir erros de orientação (bearing) podemos introduzir esta incerteza no modelo inverso do sensor. Supondo que os erros de distância e orientação sejam não correlacionados:

$$p(x|z_t,\theta_t) = P + (\frac{k}{2\pi\sigma_d\sigma_\theta} + 0.5 - P)exp\big(-\frac{1}{2}(\mu_t - x)\Sigma(\mu_t - x)'\big)$$

$$\Sigma = \begin{bmatrix}\sigma_d^2 & 0\\ 0 & \sigma_\theta^2\end{bmatrix}$$

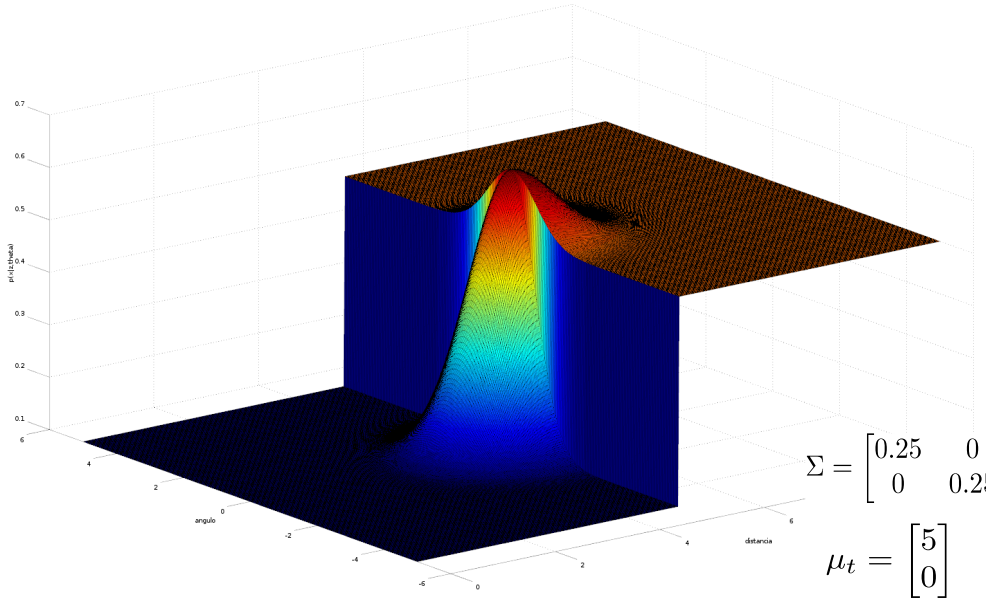
$$\mu_t = \begin{bmatrix}z_t\\ \theta_t\end{bmatrix}$$

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



P = P_{min}, se $0 \le x \le z$
0.5 se $x \ge z$

Mapeamento

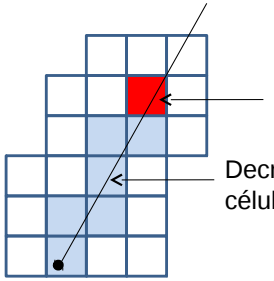


$$\Sigma = \begin{bmatrix} 0.25 & 0 \\ 0 & 0.25 \end{bmatrix}$$

$$\mu_t = \begin{bmatrix} 5 \\ 0 \end{bmatrix}$$

Mapeamento

Como a imprecisão do laser é muito menor que o tamanho da célula, pode-se utilizar o número de leituras que detectou obstáculo em uma célula como um indicativo da probabilidade da mesma estar ocupada por um obstáculo. Cada célula possui um contador iniciado como zero:



Incrementa-se 1 nas células onde foram detectados obstáculos

Decrementa-se 1 nas células ao longo do raio

A final da exploração do ambiente:
- Se o valor do contador for maior que N $\rightarrow$ P(x) =1
- Se o valor do contador for menor que -N $\rightarrow$ P(x) = 0
- P(x) = 0.5, caso contrário (área não mapeada).

Mapeamento

Algoritmo de mapeamento:

1. Associe a cada célula i $l(x) = 0$

2. Faça uma leitura do(s) sensor(es) - z,θ.

3. Para as células i no campo de cobertura do sensor compute
   $l_{t,i} = l_{t-1,i}$ + INV_SENSOR(i,z,θ) - l_0

4. Se o critério de parada for satisfeito, retorne p(x) para cada célula do mapa

4. Movimente o robô e vá para 2.

SLAM (Simultaneous Localization and Mapping)

Vimos que:
1. Dado um mapa do ambiente, o robô pode corrigir sua pose com base neste mapa (Kalman, Markov).
2. Dada uma odometria livre de erros, o robô pode derivar um mapa do ambiente (Occupancy Grid).

Nenhuma destas hipóteses é totalmente verdadeira na prática:
3. Tal mapa do ambiente raramente existe.
4. A odometria é sempre imprecisa.

SLAM é o processo pelo qual o robô levanta o mapa do ambiente e simultâneamente utiliza este mapa para se localizar.

Problema do ovo e da galinha:
- Para se localizar o robô necessita do mapa.
- Para obter o mapa o robô precisa se localizar.

SLAM

Localização X Mapeamento X SLAM

Localização: Calcular p(x | u, z, m)

Mapeamento: Calcular p(m | x, z)

SLAM: Calcular p(x, m | u , z)

Algoritmos de SLAM podem operar de duas formas:

Online: O mapa e a pose são atualizados com base apenas na pose anterior do robô.

Full: O mapa e a pose são atualizados com base no caminho percorrido pelo robô até o momento anterior (todas as poses anteriores).

SLAM

Existem basicamente 3 algoritmos (e seus derivados) de SLAM baseados em sensores de distância e mapas de landmarks:

1. EKF SLAM (online): utiliza o Filtro de Kalman Estendido. A pose e os landmarks compõem o vetor de estado.

2. GraphSLAM (full): utiliza equações lineares que relacionam a pose e os landmarks.

3. **FastSLAM** (online e full): combina Filtro de Partículas e EKF para relacionar a pose e os landmarks.