IMPLEMENTAÇÃO DO FILTRO DE PARTÍCULAS PARA A LOCALIZAÇÃO EM ROBÓTICA MÓVEL

Aurélio T. Salton*, Rafael S. Castro*, Guilherme A. Pimentel*, Jeferson V. Flores†

*PUCRS - Grupo de Automação e Controle de Sistemas
Av. Ipiranga, 6681, 90619-900
Porto Alegre (RS), Brasil

†Departamento de Engenharia Elétrica, Universidade Federal do Rio Grande do Sul, Brazil

Emails: aurelio.salton@pucrs.br, rafael.castro@pucrs.br, guilherme.pimentel@pucrs.br, jeferson.flores@ufrgs.br

Abstract— This paper presents the use of Particle Filters for the problem of localization of planar vehicles. The crucial algorithms necessary for the implementation of the particle filter, model and sensor simulation, and resampling are presented in a pseudo-code form, so as to help the reader. The paper also serves the pourpose of showing a comprehensive implementation of particle filters to the estimation of nonlinear systems. Experimental results are presented supporting the efficacy of the presented algorithms.

Keywords— Particle Filters, Localization, Robotics

Resumo— Este artigo apresenta um tutorial sobre o uso de Filtros de Partícula (FP) aplicados ao problema de localização de veículos planares. Os algoritmos mais importantes necessários para a implementação do FP, simulação do modelo e sensores, e uma forma eficiente de *resampling* são apresentados na forma de pseudo-códigos, para facilitar a compreensão do leitor. O artigo também tem por propósito apresentar de forma compreensiva a implementação de FP na estimação de sistemas não-lineares. Resultados experimentais são apresentados para demonstrar a eficácia dos algoritmos apresentados.

Palavras-chave— Filtro de Partículas, Localização, Robótica Móvel

1 Introdução

Certamente a técnica de estimação de estados de sistemas dinâmicos mais conhecida é o chamado Filtro de Kalman (FK) (Kalman, 1960). Como todo o filtro Bayesiano, o FK separa o problema de estimação de estados em duas partes, a predição – onde o modelo matemático do sistema é utilizado – e a correção – onde informações sobre os sensores são exploradas. Esta técnica de estimação ainda pertence ao chamado conjunto de filtros Bayesianos paramétricos, por parametrizar as incertezas do sistema e dos sensores na forma de distribuições Gaussianas, descritas por suas médias e covariâncias. Ainda hoje este filtro representa o estado-da-arte na estimação de estados de sistemas lineares Gaussianos. Porém, para casos não-lineares e não-Gaussianos, adaptações do FK, ou abordagens inteiramente diferentes, devem ser utilizadas.

Dentre as alternativas mais populares para estimação de estados de sistemas não-lineares destaca-se o Filtro de Partículas (FP). Por ser de natureza não paramétrica, este filtro fornece uma solução adequada para o processo de estimação de estados de sistemas não-lineares e não-Gaussianos (Sanjeev Arulampalam et al., 2002). Particularmente, este método de estimação ganhou notoriedade no meio da localização de robótica móvel com o sucesso do projeto de Stanford no *DARPA Urban Challenge*. Esta e outras aplicações mostraram que, apesar de serem computacionalmente exigentes, hoje, filtros de partícula são uma ferramenta de cunho prático na estimação de estados de sistemas não-lineares. Além disto, espera-se demonstrar que este método de estimação é de simples implementação e robusto à incertezas de modelo e ruído de sensores.

O presente artigo descreve de forma completa a implementação de FP na localização de veículos móveis, e serve como um estudo de caso de cunho aplicado para acadêmicos interessados no uso de filtros de partícula para a estimação de sistemas não-lineares. Pseudo-códigos dos principais algoritmos envolvidos no processo de localização serão apresentados a fim de auxiliar o leitor. O artigo está dividido da seguinte forma: além de apresentar o problema em questão, a Seção II fornecerá uma breve exposição dos filtros de partícula e uma lista de possíveis sistemas que podem ser contemplados com os algoritmos apresentados no restante do artigo; a Seção III irá descrever em detalhes o passo da predição do filtro através do chamado modelo Rotação-Translação-Rotação (RTR); a Seção IV apresenta o passo de correção, focando no algoritmo de simulação de sensores e no passo de *Resampling* do filtro; a Seção V apresenta resultados simulados e experimentais e a Seção VI finaliza o artigo.

2 Preliminares

Esta seção será utilizada para fornecer uma breve explicação sobre filtros de partículas, definir o problema e exemplificar alguns dos modelos de veículos autônomos que podem ser contemplados pelo algoritmo de localização tratado no restante do artigo.

2.1 Definição do problema

Veículos planares podem ser descritos pelas suas coordenadas cartesianas (x, y) e pela direção θ a qual estão apontando, normalmente representada pelo ângulo entre a direção do veículo e o eixo x. Estas três grandezas determinam a chamada *pose* de um veículo planar. No instante de tempo k, esta pose é descrita por,

$$p_k = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}.$$

Portanto a tarefa de localizar um veículo em um plano é a tarefa de encontrar uma estimativa dos valores x, y e θ, que descrevem sua pose.

Para realizar esta tarefa supõe-se que duas formas de sensores são disponíveis: um sensor que descreva o deslocamento do sistema entre dois instantes de tempo, e um sensor que forneça alguma informação sobre o ambiente em que o veículo se encontra. Como será exposto a seguir, filtros de partícula apresentam uma maneira sistemática de fundir estes dois tipos de sensores a fim de alcançar uma boa estimativa da pose real do sistema.

2.2 Filtro de Partículas

O filtro de partícula é uma forma de filtro de Bayes por executar a estimação em dois passos: predição e correção. Este artigo supõe que a predição é feita através de *encoders* colocados nas rodas traseiras do veículo, e a correção é feita através de sonares alocados na parte da frente e nas laterias do mesmo. Neste cenário, a estratégia do filtro de partículas pode ser descrita nos quatro passos apresentados a seguir.

1. O filtro é inicializado com N_p "partículas" que representam possíveis condições iniciais do sistema. Cada uma destas partículas possui uma pose própria $\tilde{p}_i$ com valores estimados para x_i, y_i e θ_i, para $i = 1, \ldots N_p$.

2. Para cada partícula, o filtro utiliza os dados dos *encoders* e a pose passada para fazer uma predição da nova pose.

3. À toda partícula é associado um valor que representa a probabilidade de que ela se encontre na posição certa. Para isto o filtro usa os sonares e o mapa: ele calcula a probabilidade de que a partícula receba as medições que os sonares receberam, dado que ela se encontra em sua pose.

4. No último passo o filtro elimina as partículas com baixa probabilidade, substituindo-as por variações das partículas de maior probabilidade, dando origem a novos valores $\tilde{p}_i$.

Note que o passo 2 representa a predição, e os passos 3 e 4 representam a correção do filtro. Portanto, os sensores utilizados podem ser de qualquer tipo desde que permitam a iteração entre estes passos. A estimativa da pose do sistema $\hat{p}_k$ é dada pela média ponderada das poses das partículas, i.e., em um dado instante de tempo k,

$$\hat{p}_k = \sum_{i=1}^{N_p} w_i \tilde{p}_i, \quad \sum_{i=1}^{N_p} w_i = 1. \tag{1}$$

2.3 Modelo do Sistema

Aqui será considerado o veículo diferencial, onde as rodas direita e esquerda são controladas de forma independente. Este modelo determina a dinâmica do sistema a partir da velocidade de cada roda (v_d e v_e):

$$\begin{aligned} \dot{x} &= R\frac{(v_d + v_e)}{2}\cos(\theta), \\ \dot{y} &= R\frac{(v_d + v_e)}{2}\text{sen}(\theta), \\ \dot{\theta} &= R\frac{(v_d - v_e)}{l}, \end{aligned} \tag{2}$$

onde R representa o raio das rodas e l a distância entre cada roda. Além de serem de fácil construção, veículos diferencias podem rotacionar em cima do próprio eixo, facilitando sua locomoção. Por estas razões, este foi o modelo construído em laboratório e será considerado neste artigo.

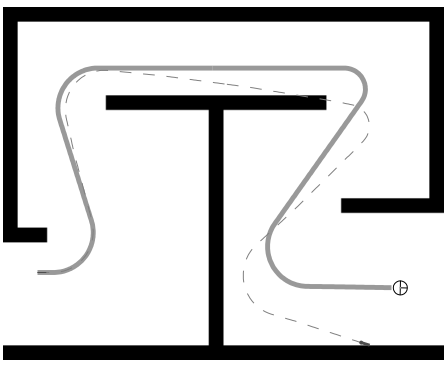


Figura 1: Trajetória do veículo (linha cinza) comparada com as medições errôneas da odometria (linha tracejada).

3 Predição

3.1 Odometria

A odometria consiste em uma estimativa do deslocamento do veículo a partir da informação proveniente de seus *encoders*. Considera-se que a leitura dos pulsos seja feita com um período de amostragem de T segundos. Dentro destes intervalos de tempo pode-se facilmente calcular a distância percorrida por cada roda. Para *encoders* de N pulsos por rotação, as distâncias percorridas pelas rodas direita e esquerda do veículo são calculadas por,

$$d_d = 2\pi R\frac{\Pi_d}{N}, \quad d_e = 2\pi R\frac{\Pi_e}{N}, \tag{3}$$

sendo Π_d o número de pulsos recebidos pelo *encoder* da roda direita e Π_e da roda esquerda.

A distância total percorrida pelo veículo no intervalo de T segundos pode ser aproximada por,

$$d_T = \frac{d_d + d_e}{2}. \tag{4}$$

O que leva à uma estimativa da pose do sistema dada por,

$$\begin{aligned} \bar{\theta} &= \theta + \frac{d_d + d_e}{L}, \\ \bar{x} &= x + d_T \cos\left(\frac{\bar{\theta}+\theta}{2}\right), \\ \bar{y} &= y + d_T \,\mathrm{sen}\left(\frac{\bar{\theta}+\theta}{2}\right). \end{aligned} \tag{5}$$

Sendo que $[x\ y\ \theta]^\intercal$ compõe a pose do sistema no instante anterior. Uma vez que esta é uma aproximação discreta, é natural que a pose real do sistema seja tal que, $p_k \neq [\bar{x}\ \bar{y}\ \bar{\theta}]^\intercal$, e que o erro de estimação esteja associado ao período de amostragem T. Além disto, *encoders* estão sujeitos à perdas de pulsos por deslizamento das rodas e problemas computacionais, levando a graves erros de estimação. Um exemplo das consequências do uso exclusivo de *encoders* na localização está demonstrado na Fig. 1. Para gerar esta figura, pulsos do *encoder* da roda direita foram perdidos ao longo da trajetória. Note que esta localização é "em malha aberta," uma vez que ela não corrige qualquer tipo de erro que aconteça durante a trajetória do veículo.

3.2 Modelo RTR

Como visto na Seção 2.2, o filtro de partículas requer que pequenas variações sejam adicionadas na dinâmica de cada partícula, é justamente esta a função do modelo RTR. Este modelo calcula a nova pose de uma partícula través de uma Rotação inicial δ_{r1}, seguida de uma Translação δ_{tr} e uma Rotação final δ_{r2}, por esta razão, aqui ele é referido como modelo RTR.

Seu algoritmo é descrito na Tabela 1, ele recebe como entrada a pose anterior $\tilde{p}_i = [x\ \ y\ \ \theta]^\intercal$

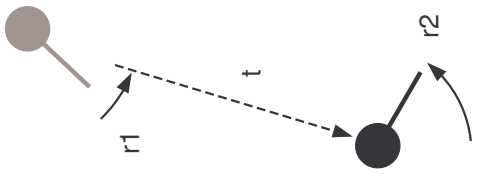


Figura 2: Ilustração do modelo RTR: Rotação inicial δ_{r1}, Translação δ_{tr} e Rotação final δ_{r2}.

$\tilde{p}_i = \text{modelo\_RTR}(\tilde{p}_i, [\bar{x}\ \ \bar{y}\ \ \bar{\theta}])$

1 $\delta_{r1} = \text{atan2}(\bar{y} - y, \bar{x} - x) - \bar{\theta}$
2 $\delta_{tr} = \sqrt{(\bar{x} - x)^2 + (\bar{y} - y)^2}$
3 $\delta_{r2} = \bar{\theta} - \theta - \delta_{r1}$

4 $\hat{\delta}_{r1} = \text{rand}\left(\mathcal{N}(\delta_{r1},\ \alpha_1\delta_{r1}^2 + \alpha_2\delta_{tr}^2)\right)$
5 $\hat{\delta}_{tr} = \text{rand}\left(\mathcal{N}(\delta_{tr},\ \alpha_3\delta_{tr}^2 + \alpha_4\delta_{r1}^2 + \alpha_4\delta_{r2}^2)\right)$
6 $\hat{\delta}_{r2} = \text{rand}\left(\mathcal{N}(\delta_{r2},\ \alpha_1\delta_{r2}^2 + \alpha_2\delta_{tr}^2)\right)$

7 $\bar{x} = x + \hat{\delta}_{tr}\cos(\theta + \hat{\delta}_{r1})$
8 $\bar{y} = y + \hat{\delta}_{tr}\text{sen}(\theta + \hat{\delta}_{r1})$
9 $\bar{\theta} = \hat{\delta}_{r1} + \hat{\delta}_{r2}$

10 return $\tilde{p}_i = [\bar{x}\ \ \bar{y}\ \ \bar{\theta}]^\intercal$

Tabela 1: Algoritmo do modelo RTR.

e a estimativa da pose atual após a predição dos *encoders* $\bar{p}_i = [\bar{x}\ \ \bar{y}\ \ \bar{\theta}]^\intercal$ (está estimativa deve ser feita para cada partícula), e retorna uma nova pose "perturbada." As três primeiras linhas do algoritmo são responsáveis pela estimativa da rotação inicial, translação e rotação final que o veículo teria realizado (Fig. 2). As linhas 4 à 6 geram variações destes movimentos de acordo com os parâmetros $\alpha_1 \ldots \alpha_4$. Neste passo, uma rotina de criação de números pseudo-aleatórios é utilizada para amostrar a curva normal de média μ e variância σ^2, representa por $\mathcal{N}(\mu,\ \sigma^2)$. As últimas linhas do algoritmo calculam a nova pose da partícula a partir destes parâmetros perturbados.

Uma simulação do modelo RTR com 50 partículas repetindo a trajetória da Fig. 1 pode ser visualizada na Fig. 3. A simulação supõe que a condição inicial do veículo era conhecida, mas permitiu que as partículas se dispersassem ao longo da trajetória. Esta dispersão é controlada pelos parâmetros $\alpha_1 \ldots \alpha_4$ (linhas 4 à 6).

4 Correção

O passo de correção pode ser divido em duas partes. Primeiro, à cada partícula é associada a probabilidade de que ela se encontre no lugar correto. Segundo, as partículas de baixa probabilidade são eliminadas e substituídas por variações das partículas de alta probabilidade. A realização do pri-

meiro passo exige que as medições reais dos sonares sejam comparadas com as medições que cada partícula deveria obter. Ou seja, este passo requer a simulação das medições vindas dos sonares.

4.1 Simulando os sonares

O método discutido aqui, denominado de *Ray Casting*, é comumente utilizado na computação gráfica para a solução de intersecções entre retas e sólidos (Roth, 1982). Ele consiste em projetar um raio na direção em que o sensor está apontando, a partir do veículo até um obstáculo. Obviamente, para que este algoritmo possa ser realizado, é necessário que um mapa do ambiente seja disponível. O mapa consiste em uma matriz binária onde cada obstáculo é representado pelo bit 1, e cada espaço livre pelo bit 0. Por exemplo, o mapa das figuras 1 e 3 pode ser dado pela matriz,

$$m = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}. \qquad (6)$$

No processo de simulação dos sonares, para toda a partícula é calculado o número de *pixels* que separam o veículo do obstáculo mais próximo na direção em que o sonar está apontando. Normalmente o mapa utilizado deve ser muito maior do que (6), mas depende de cada aplicação. Cada *pixel* pode representar poucos centímetros quadrados, por exemplo.

O algoritmo `ray_casting` da Tabela 2 é um modelo de sensores de distância do tipo *lasers*, mas pode ser utilizado para simular sonares[1]. O algoritmo recebe como entrada a pose da partícula $\tilde{p}_i$ e um vetor z contendo as medições de cada sonar. Também são utilizadas variáveis representando o mapa m em que o veículo está inserido, o vetor Γ_z com os ângulos em que cada sonar está apontando, a covariância σ_z das medições e o número de sonares N_z. Na primeira linha do código a posição da partícula é comparada com o mapa, se ela se encontra fora do mesmo ou sobre um obstáculo, recebe probabilidade zero ($w = 0$). Se este não é o caso, o algoritmo simula os sonares entre as linhas 5 e 16. Suas condições de parada para cada sonar se encontram na linha 15, se o sonar chegou à algum obstáculo, ou na linha 11, se foi alcançada distância máxima que o sensor pode medir. Na linha 18 uma comparação entre a distância calculada d e o valor real medido pelo sonar $z(i)$ é realizada através da gaussiana de média d e covariância σ_z. Assim, o algoritmo da Tabela 2 retorna uma medida de confiança (importância) w associada a cada partícula.

[1]Vale lembrar que sonares fazem medições cônicas, portanto, para um modelo preciso de sonares, inúmeros raios com ângulos ligeiramente diferentes deveriam ser lançados, o que gera um custo computacional elevado.

	w = `ray_casting`$(\tilde{p}, z)$
1	`if` $m(\tilde{p}(1), \tilde{p}(2)) = 1$ `||` $\tilde{p} \notin m$
2	$w = 0$
3	`else`
4	$w = 1$
5	`for` $i = 1 : N_z$
6	$dx = \cos(\theta + \Gamma_z(i))$
7	$dy = \sin(\theta + \Gamma_z(i))$
8	$x = \tilde{p}(1)$
9	$y = \tilde{p}(2)$
10	$d = d_{max}$
11	`while` $d \leq d_{max}$
12	$x = x + dx$
13	$y = y + dy$
14	$d = \sqrt{(x - \tilde{p}(1))^2 + (y - \tilde{p}(2))^2}$
15	`if` $m(x, y) = 1$
16	`break`
17	`end (while)`
18	$w = w \cdot \mathcal{N}(z(i) - d,\ \sigma_z^2)$
19	`end (for)`
20	`return` w

Tabela 2: Algoritmo *Ray Casting*.

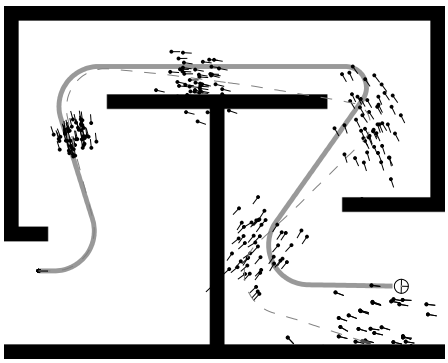
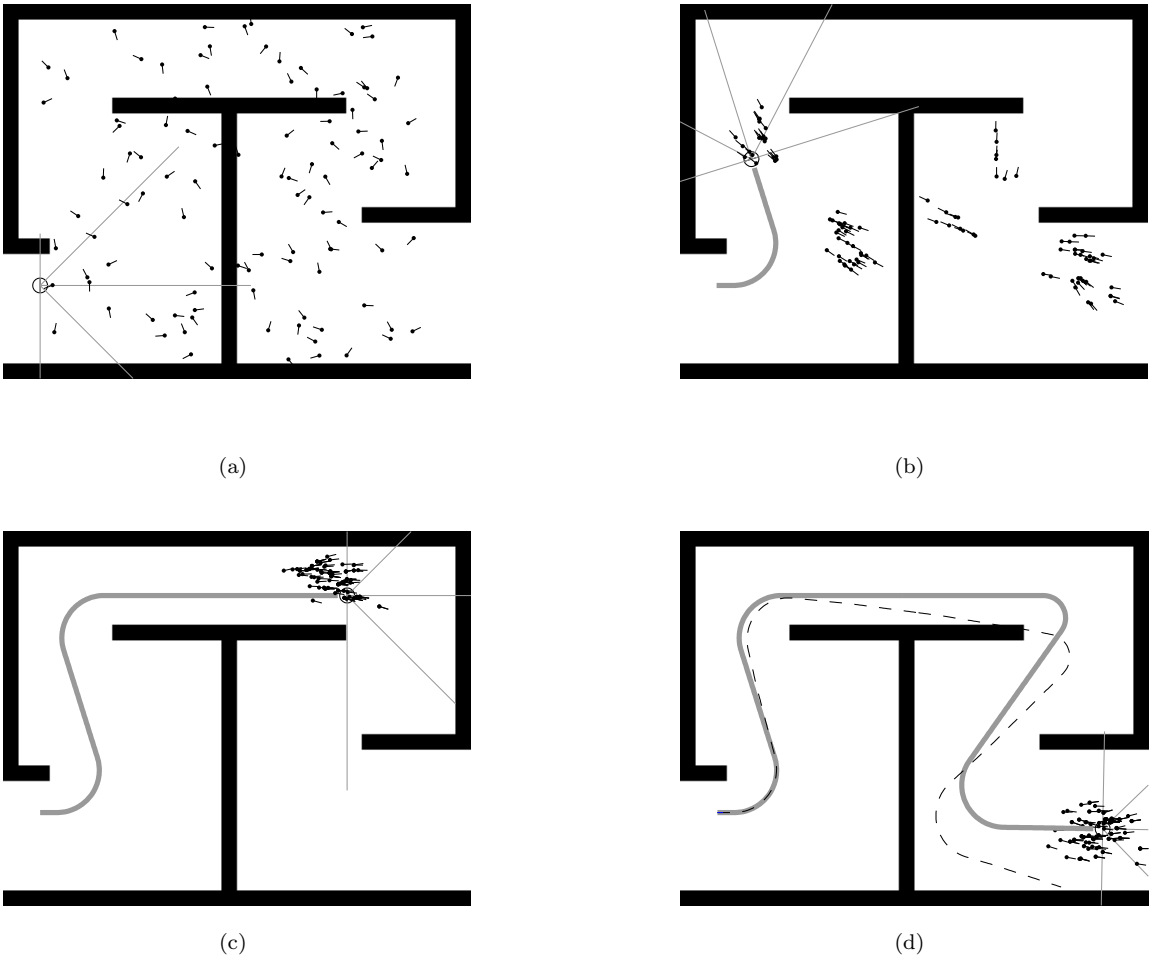


Figura 3: Dispersão das partículas devido ao modelo RTR. Neste caso $(\alpha_1,\ \alpha_2,\ \alpha_3,\ \alpha_4) = (2,\ 50,\ 50,\ 2)$.

4.2 Selecionado as melhores partículas

O método de seleção das melhores partículas é formalmente conhecido por *Importance Sampling* uma vez que amostra cada partícula de acordo com sua importância (Hastings, 1970). Este algoritmo faz com que partículas com alta importância w não apenas sobrevivam mas também



(a) (b) (c) (d)

Figura 4: Evolução do filtro a partir de uma condição inicial desconhecida. Neste caso foram utilizadas 100 partículas e um período de amostragem $T = 0.5$ s.

se "reproduzam" mais vezes que as de baixa importância. Em termos práticos, devemos criar um vetor $\mathcal{X}$ com as partículas que serão utilizadas no próximo passo do filtro. Uma forma simples de criar este vetor é conhecido por *Sampling Wheel* (Thrun et al., 2005) e está descrito na Tabela 3. A primeira linha deste algorítimo inicializa uma variável de controle β, que servirá como um limiar variante: partículas com probabilidade $w_j > \beta,\ j = 1 \dots N_p$, serão selecionadas para compor o próximo conjunto de partículas, ou seja, serão inseridas em $\mathcal{X}$. Para percorrer o vetor $W = [w_1 \dots w_{N_p}]$ a variável j é criada de forma aleatória na segunda linha do algoritmo. A terceira linha inicia um laço de repetição que irá selecionar N_p partículas e inseri-las no vetor $\mathcal{X}$.

Para fazer isto, o limiar β também é inicializado com um valor aleatório e comparado aos valores w_j das partículas. Quando um w_j for grande o suficiente, i.e., $w_j > \beta$, este é selecionado para compor o vetor $\mathcal{X}$. Do contrário, o limiar é reduzido e a comparação é feita com a próxima partícula do vetor W. Note que desta forma partículas com

	$\mathcal{X}$ = seleção$(W = [w_1 \ \ w_2 \ \ \dots \ w_{N_p}])$
1	β=0
2	$j =$ int (rand ($[1\ N_p]$))
3	for $i = 1 : N_p$
4	$\beta = \beta +$ rand($[0 \ \ \max(W)]$)
5	while $\beta > w_j$
7	$\beta = \beta - w_j$;
6	$j = j + 1$
8	end (while)
9	$\mathcal{X}_i = w_j$
10	end (for)
11	return $\mathcal{X}$

Tabela 3: Algoritmo de seleção e reprodução das melhores partículas.

baixo valor de w podem ser selecionadas, mas há uma probabilidade proporcionalmente pequena de que isto aconteça. Esta é uma maneira eficaz de criação de novas amostras das partículas com base

na probabilidade que cada partícula esteja na posição correta.

5 Implementação

5.1 Resultados Simulados

O filtro descrito até agora foi simulado no ambiente apresentado nas figuras 1 e 3. Nestas simulações o mapa consistiu em uma matriz 300×240 posições, representando um espaço físico de 2.5×2 metros. Além de *encoders* com medições ruidosas, estas simulações utilizaram cinco sensores de distância direcionados à -90°, -45°, 0°, 45° e 90° com relação à frente do veículo, com covariância $\sigma_z = 0.5$ m e alcance máximo $d_{max} = 1.0$ m. As simulações utilizaram um total de 100 partículas que foram inicializadas de forma aleatória no mapa, supondo que a posição inicial do veículo era completamente desconhecida. Quatro *snapshots* desta simulação estão apresentados nos gráficos da Fig. 4. O período de amostragem utilizado foi de 0.5 s e o passo de correção do filtro (*importance sampling*) ocorreu a cada dez amostras. Além disto, os coeficientes que determinam a variabilidade das partículas foram definidos como $(\alpha_1,\ \alpha_2,\ \alpha_3,\ \alpha_4) = (2,\ 50,\ 50,\ 2)$.

5.2 Resultados Experimentais

Este mesmo filtro foi aplicado à um veículo real que possui características similares ao simulado: mesmo número e disposição dos sonares, dois *encoders* nas rodas traseiras, dimensões físicas similares, etc. Nos resultados experimentais o veículo foi exposto ao labirinto da Fig. 5. Além da trajetória real aproximada do veículo (linha tracejada), estão demostradas na figura a trajetória estimada pela integração dos *encoders* (linha cinza) e a trajetória corrigida pelo filtro de partículas (linha escura). Neste experimento o filtro utilizou 250 partículas em um mapa dado por uma matriz com dimensões 120×166, representando um ambiente de 1.20×1.66 m. No experimento a dispersão das partículas foi controlada por $(\alpha_1,\ \alpha_2,\ \alpha_3,\ \alpha_4) = (0.001,\ 200,\ 50,\ 0)$ (o ajuste destes parâmetros requer um bom entendimento do modelo RTR). Um Arduino Mega foi utilizado para a comunicação do veículo com um PC externo, onde o processamento do filtro foi executado. Para atender o número de partículas utilizadas o período de amostragem de $T = 0.5$ s se mostrou suficiente.

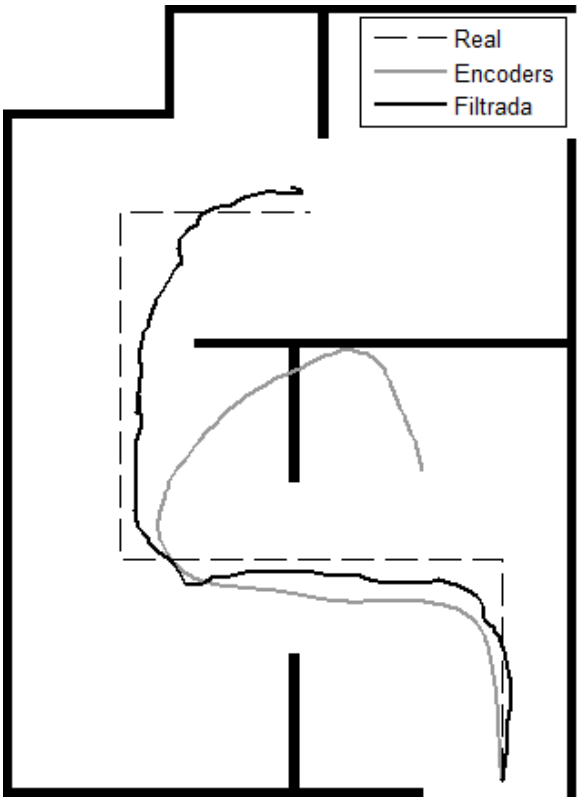


Figura 5: Resultados Experimentais.

6 Conclusão

Este artigo propôs uma exposição sobre o uso de Filtros de Partículas no problema de localização de veículos móveis. Ao mesmo tempo em que os principais algoritmos para a solução do problema foram apresentados, o mesmo serviu como uma motivação e introdução ao uso de FP para a estimação de estados de sistemas não-lineares. Resultados simulados e experimentais demonstraram que esta classe de algoritmos pode ser facilmente implementada, não estando mais limitada ao seu custo computacional.

Referências

Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications, *Biometrika* **57**(1): 97–109.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems, *Journal of basic Engineering* **82**(1): 35–45.

Roth, S. D. (1982). Ray casting for modeling solids, *Computer Graphics and Image Processing* **18**(2): 109–144.

Sanjeev Arulampalam, M., Maskell, S., Gordon, N. and Clapp, T. (2002). A tutorial on particle filters for online nonlinear/non-gaussian bayesian tracking, *Signal Processing, IEEE Transactions on* **50**(2): 174–188.

Thrun, S., Burgard, W. and Fox, D. (2005). *Probabilistic robotics*, MIT press.