Universidade Federal do Rio Grande do Sul
Instituto de Informática
INF01034 - Robótica II
Prof. Renan Maffei

Trabalho Prático 4 - Localização

Prazo: 17/11/2020

Considerações gerais

O trabalho consiste em localizar o robô em um ambiente conhecido, sem conhecer a pose inicial do robô. A localização deverá ser feita usando um filtro de partículas.

O trabalho deverá ser feito preferencialmente em DUPLA (senão INDIVIDUALMENTE). A avaliação do trabalho será feita através da análise do funcionamento das implementações no simulador MobileSim. Entregue via Moodle um arquivo compactado com os códigos desenvolvidos, e posteriormente o trabalho deverá ser apresentado ao professor em horário a combinar.

1. Instruções sobre implementação da localização de Monte Carlo no framework

Foi adicionada ao framework utilizado nos trabalhos da disciplina uma classe chamada `MCL` descrita no arquivo `MCL.cpp`. Esta classe é responsável por carregar um mapa do ambiente onde o robô se encontra e executar um filtro de partículas sobre este mapa para resolver o problema de localização global. Duas estratégias de pesagem de partículas serão desenvolvidas: uma com um modelo de pesagem tradicional de partículas (usando *ray-casting*) e outra com um modelo mais simples.

Na classe `MCL`, já estão implementadas a inicialização do mapa (método `readMap`), inicialização das partículas (método `initParticles`), desenho do mapa com as partículas (método `draw`) e cálculo da média e covariância das partículas (método `updateMeanAndCovariance`).

- O acesso às variáveis utilizadas em cada uma das duas estratégias é feito usando o indice da estratégia em cada variável. Todas as funções estão adaptadas para acessar somente as variáveis referentes a uma estratégia (informada pelo parâmetro `set`).
- Por padrão o filtro possui 10000 partículas. É possível alterar este número no construtor.
- A visualização das partículas pode (ou não) considerar transparência, o que auxilia na percepção quando muitas partículas ocupam o mesmo local. Para ligar/desligar a transparência, pressione `t`.

Também está implementado o método `run`, chamado a cada instante t pela classe `Robot`, que é composto pelas etapas de **sampling**, **weighting** e **resampling**. Por enquanto, essas três funções estão vazias, logo o objetivo deste trabalho é implementá-las corretamente.

Para rodar o programa é preciso passar como parâmetro o nome do mapa (que deverá estar dentro da pasta DiscreteMaps).
Ex: `-m SparseMap.txt`

Exemplo de funcionamento via **terminal**:

```
../build-make/program sim -m SparseMap.txt
```

Exemplo de funcionamento via **QtCreator**:

- vá em **Projects->Build & Run->Run**
- cole em '**Command line arguments**':
  `sim -m DenseMap.txt`

Note que o nome do mapa do arquivo .txt (da pasta DiscreteMaps) passado como parâmetro para o programa deve ser coerente com o arquivo .map (da pasta Maps) aberto no simulador MobileSim.

Três ambientes estão disponíveis para realizar testes, conforme mostrados na Figura 1. O primeiro ("DenseMap") é um ambiente pequeno, com pouca ambiguidade, onde a localização deverá ocorrer de forma rápida. Já o segundo ("AmbiguousMap") é um ambiente totalmente ambíguo. Neste ambiente, as partículas do filtro devem convergir para dois grupos simetricamente opostos. Por fim, o terceiro ("SparseMap") é um ambiente esparso grande, onde será possível ver que diferentes trajetos feitos pelo robô implicarão em tempos de convergencia bem diferente. São fornecidas duas configurações iniciais diferentes que podem ser carregadas no MobileSim ("SparseMap A" e "SparseMap B"), ambas devem ser usadas com o mesmo mapa no framework ("SparseMap.txt") - na primeira configuração o robô começa dentro da sala da esquerda e na segunda começa no meio do grande espaço vazio (esta é a que mais permite ver as diferenças de performance devido as escolhas na trajetoria do robô).

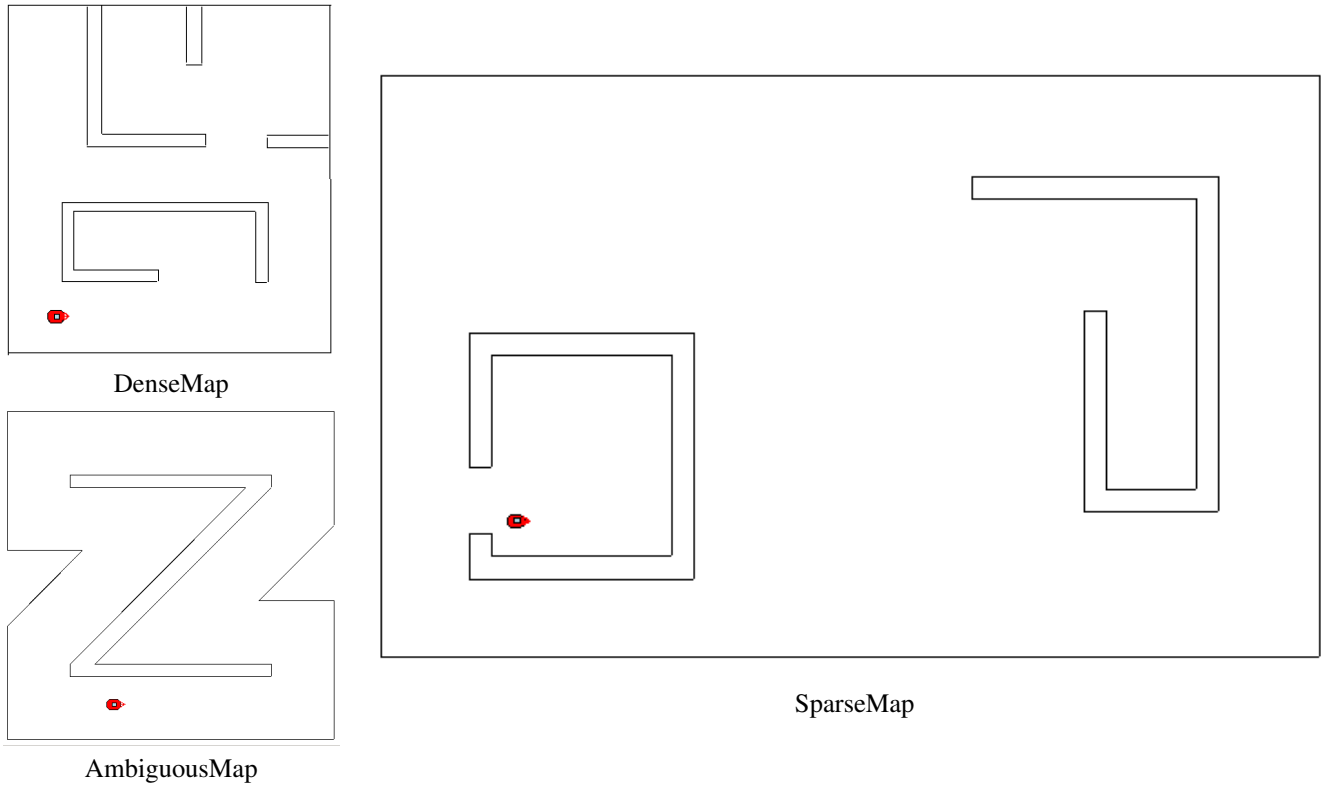

Figura 1. Exemplo de mapas

A versão atual do framework contém 8 modos de visualização do mapa, que podem ser alternados pressionando a tecla 'v'.

0. Mapa do MCL mostrando as partículas, com estratégia complexa de pesagem *(default)*
1. Mapa do MCL mostrando as partículas, com estratégia simples de pesagem
2. Mapa com LOG-ODDS usando LASER
3. Mapa com HIMM usando LASER
4. Mapa com classificação das células
5. Mapa com Campo Potencial Harmônico
6. Mapa com Campo Potencial com Preferências
7. Mapa com Campo Potencial com Objetivos Dinâmicos próximos à paredes

OBS: por enquanto todos os mapas estão em branco (com exceção dos mapas do MCL que são conhecidos), pois vocês precisam colocar a função de mapeamento que desejam usar.

1.1. Implementação da etapa de SAMPLING

Complete a função `sampling` da classe `MCL`. Esta função é exatamente igual para as duas estratégias de MCL a serem implementadas (portanto o código a ser implementado independe do valor do parâmetro `set`).

O objetivo é propagar cada uma das partículas de acordo com o **modelo de movimento baseado em odometria** visto em aula. Para isso deve-se usar a odometria informada pelo robô (descrita pela `Action u`) composta por três componentes: `rot1, trans, rot2`.
OBS: a geração dessas três componentes já é feita pela classe Robot.

Cada partícula deve gerar uma movimentação diferente (aleatória), mas próxima aos valores definidos por `u`. Para isso, primeiro é preciso gerar 3 distribuições normais de média zero e variâncias dadas em função de `rot1, trans, rot2` conforme descritas no algoritmo visto em aula. Então para cada partícula, deve-se gerar uma amostra de cada uma das 3 distribuições. Por fim, encontra-se a nova posição da partícula usando os valores amostrados de `rot1, trans, rot2`.

DICA: como no início o programa começa com milhares de partículas em todas as posições e direções do mapa, fica difícil de depurar se algo está errado. Portanto, para testar se essa etapa do algoritmo está OK, diminua a quantidade de partículas no construtor (por exemplo, para 10 partículas). Altere a inicialização das partículas em `initParticles`, fixando em ZERO o ângulo de cada partícula. Aí tente analisar se o movimento delas faz sentido em comparação com a trajetória do robô. Se estiver OK, o *sampling* deve estar correto.

1.2. Implementação da etapa de IMPORTANCE WEIGHTING

Complete a função `weighting` da classe `MCL`.

Nesta etapa, o objetivo é avaliar cada uma das partículas, ou seja, determinar a probabilidade de adequação de cada partícula à distribuição esperada. Duas estratégias serão implementadas, de acordo com o valor informado no parâmetro `set`.

Estratégia simples (`set` = 1):

A estratégia simplificada de pesagem apenas deverá colocar peso **zero** para todas as partículas que, após a movimentação, ficarem sobre alguma célula não-livre. As demais partículas receberão peso um. Observe como a checagem é feita na inicialização das partículas (método `initParticles`).

Na sequência é preciso normalizar os pesos das partículas, ou seja, calcular a soma de todos os pesos e dividí-los por essa soma. Note que se a soma der zero, a divisão não pode ser feita, nesse caso deixe todas as partículas com o mesmo peso ($1/N$). Porém isso é um sinal de que a variância no cálculo do peso está muito pequena.

Todas as partículas que ficarem com peso zero serão posteriormente eliminadas na etapa de *resampling*.

Estratégia complexa (`set` = 0):

A estratégia mais complexa de pesagem também realizará o teste simplificado acima, mas ele servirá apenas como um teste preliminar. Após o teste simplificado, o método deverá avaliar quão boa é cada partícula considerando as observações feitas pelo robô, ou seja, deverá computar probabilidades de acordo com o modelo de observação $p(\boldsymbol{z}_t \mid \boldsymbol{x}_t^{[p]}, \boldsymbol{m})$. Na prática, devemos comparar as observações do robô (dadas pelas leituras do laser `z`) e as observações esperadas da partícula na posição $\boldsymbol{x}_t^{[p]}$.

Sabemos que a cada instante o robô faz múltiplas leituras com o laser:

$$\boldsymbol{z}_t = (\boldsymbol{z}_t^1, \boldsymbol{z}_t^2 \cdots, \boldsymbol{z}_t^K)^T$$

Como as leituras são independentes, a probabilidade final associada à particula p pode ser aproximada pelo produto das probabilidades individuais:

$$p(\boldsymbol{z}_t \mid \boldsymbol{x}_t^{[p]}, \boldsymbol{m}) = \prod_{k=1}^{K} p(\boldsymbol{z}_t^k \mid \boldsymbol{x}_t^{[p]}, \boldsymbol{m})$$

E a probabilidade de uma medição individual pode ser definida de acordo com o modelo visto em aula (Basic Beam Model). Aqui vamos simplificar o modelo e dizer que para uma dada observação medida pelo robô $\boldsymbol{z}_t^k$ e a observação esperada $\boldsymbol{z}_t^{k*}$ pela partícula, a probabilidade de semelhança entre as observações é computada por:

$$\mathcal{N}(\boldsymbol{z}_t^k, \boldsymbol{z}_t^{k*}, var) = \frac{1}{\sqrt{2\pi var}} e^{-\frac{1}{2}\frac{(\boldsymbol{z}_t^k - \boldsymbol{z}_t^{k*})^2}{var}}$$

onde var é a variância (dada em m^2) da distribuição associada às medições.

OBS: Você deve definir essa variância. Valores muito altos podem fazer com que o filtro demore a convergir, pois medições diferentes (boas ou ruins) terão pesos parecidos. Também podem fazer com que mesmo após convergência as partículas fiquem muito espalhadas. No entanto, valores muito pequenos farão com que quase todas as medições tenham peso zero, e eliminarão muitas hipóteses boas no início do processo. Esse caso é muito pior, pois o filtro não tem como se recuperar.

$\boldsymbol{z}_t^k$ é o k-ésimo valor do vetor de observações `z`, informado como entrada para a função. Já $\boldsymbol{z}_t^{k*}$, o valor esperado da k-ésima medição feita na posição da partícula p deve ser computado usando *ray-casting* sobre o mapa. Para poupar trabalho, o framework fornece a função `computeExpectedMeasurement(int index, Pose p)`, então basta informar o indíce da medição que se deseja estimar e a posição da partícula, que a função retorna a distância (em m) que se espera medir no mapa.

DICA: Essa etapa é a mais demorada do filtro. Para acelerar você não precisa computar a probabilidade das 181 medições do laser. Uma dica é pular de 10 em 10, ou de 20 em 20. Ou seja, a probabilidade final pode ser aproximada pela $prob(z_0) \cdot prob(z_{10}) \cdot prob(z_{20}) \cdots prob(z_{180})$

Por fim, lembre-se de normalizar os pesos das partículas, como implementado na estratégia simplificada.

1.3. Implementação da etapa de RESAMPLING

Complete a função `resampling` da classe `MCL`.

Nesta etapa você deve reamostrar o conjunto de partículas, dando maior prioridade para partículas de alto peso. Então gere um novo vetor de partículas selecionando aleatóriamente (com reposição) as melhores partículas do conjunto atual. No final substitua o conjunto atual pelo conjunto reamostrado de partículas.

Implemente o amostrador de baixa variância pois este é o mais eficiente dentre os métodos vistos em aula.

Análise dos resultados

Compare as duas implementações de pesagem e veja como a convergência obtida com a estratégia complexa é muito mais rápida do que com a estratégia simplificada. Verifique também o impacto das diferentes estratégias de exploração implementadas no trabalho 3 (campos potenciais com e sem preferências) na eficiência do processo de localização.