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

Trabalho Prático 2 - Mapeamento

Prazo: 06/05/2019

Considerações gerais

O trabalho deverá ser feito preferencialmente em DUPLA (senão INDIVIDUALMENTE).

A avaliação do trabalho será feita através da análise do funcionamento das implementações: primeiramente no simulador MobileSim, e posteriormente com o robô real. O teste no robô real só poderá ser feito quando a implementação estiver funcionando bem no simulador.

Além de apresentar o funcionamento da implementação daquilo que for pedido, entregue via Moodle um arquivo compactado com os códigos desenvolvidos.

Instruções sobre implementação de occupancy grids no framework

O framework disponibilizado neste trabalho contem uma classe chamada `Grid` descrita no arquivo `Grid.cpp`, que dentre outras coisas define funções para desenhar o mapa na tela. **Não é necessário modificar essa classe.**
A versão atual do framework contém 4 modos de visualização do mapa, que podem ser alternados pressionando a tecla 'v'.

0. Mapa com LOG-ODDS usando LASER *(default)*
1. Mapa usando SONAR
2. Mapa com HIMM usando LASER
3. Sem mapa

OBS: por enquanto todos os mapas estão em branco, pois falta implementar as funções de mapeamento.

O grid é composto por células (classe `Cell`) contendo atributos de ocupação. A escala do grid (que indica quantas células correspondem a um metro) pode ser obtida através da função:
`int scale = grid->getMapScale();`
Por padrão, a escala é 10, logo se, por exemplo, o robô fizer uma leitura de 5 metros com o laser, o método de mapeamento deverá atualizar uma distância de 50 células.

Para atualizar o mapa (independente do método implementado), é preciso acessar as células dentro do campo-de-visão do sensor que está sendo utilizado, cujo alcance máximo é dado por `base.getMaxLaserRange()` ou `base.getMaxSonarRange()`. Tal valor é dado em metros, logo lembre-se de multiplicar por `scale` para obter o valor correspondente em número de células. Não é preciso checar células mais distantes do robô do que o alcance máximo do sensor, conforme mostra a Figura 1.

Uma posição (x,y) em metros é mapeada para uma célula do grid (i,j), multiplicando-se (x,y) pela escala do mapa. Por exemplo, considerando `scale=10`:

- posição (0.0, 0.0) corresponde à célula (0,0)
- posição (1.0, -2.0) corresponde à célula (10,-20)
- posições (0.30, 0.50) e (0.33, 0.57) correspondem à célula (3,5)

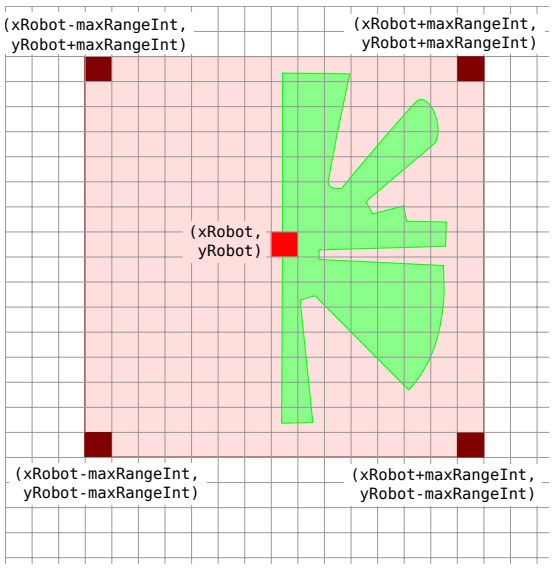


Figura 1: Células a serem atualizadas durante o mapeamento.

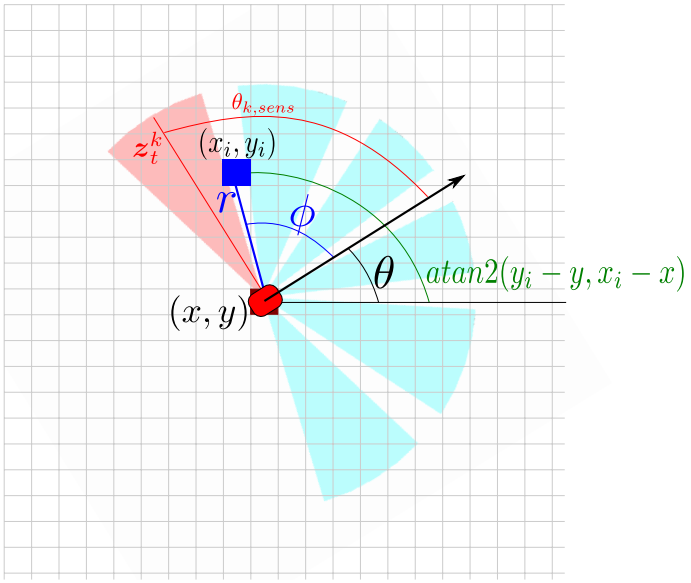


Figura 2: Atualização de uma célula

O acesso à célula situada na posição (i, j) é feito atraves da função:

```
Cell* c = grid->getCell(i,j);
```

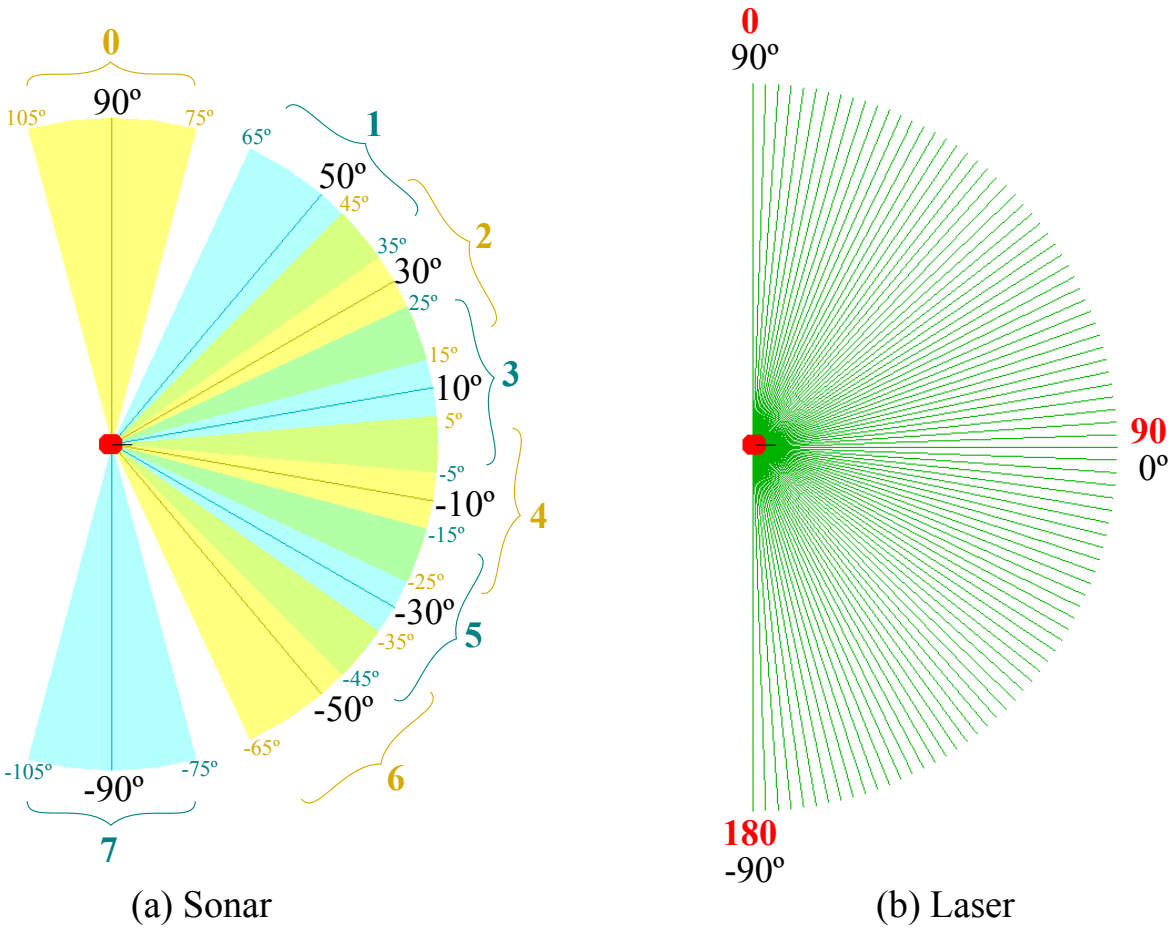
A atualização da ocupação das células deve ser feita seguindo o modelo inverso de sensor visto na "*Aula 08 - Occupancy Grids*", exemplificado na Figura 2. Para uma dada célula na posição (x_i, y_i) é preciso:

1. Computar a distância r até a célula onde está o robô. Como essa distância é dada em número de células, lembre-se de dividí-la por `scale` para convertê-la para metros e poder compará-la com as medidas dos sensores.

2. Computar a orientação ϕ da célula em relação ao robô em coordenadas locais (ou seja, já descontando a orientação θ do robô).
   OBS: Todos os ângulos computados devem estar devidamente normalizados (entre $-180°$ e $180°$). Para auxiliar use a função: `phi = normalizeAngleDEG(phi);` descrita em `Utils.cpp`

3. Encontrar a medida do sensor k mais próxima da orientação da célula em relação ao robô. Isso depende se está sendo utilizado laser ou sonar. A Figura 3 mostra a configuração tanto do sonar quanto do laser. Para facilitar o trabalho, são fornecidas as funções `base.getNearestSonarBeam(phi)` ou `base.getNearestLaserBeam(phi)`, que retornam o índice da medida mais próxima do ângulo `phi`. Se a orientação da célula for menor do que $-90°$ o sensor mais próximo será o último à direita. Se a orientação da célula for maior que $90°$ o sensor mais próximo será o último à esquerda.

4. Atualizar a ocupação da célula como ocupada ou livre dependendo da região do sensor em que se enquadrar. Para isso deve-se testar se a célula está dentro da abertura do campo-de-visão do sensor através da diferença entre a orientação ϕ da célula e a orientação da medida k, dada pelas funções `base.getAngleOfSonarBeam(k)` ou `base.getAngleOfLaserBeam(k)`. E também verificar se a distância r é próxima ou menor da médida do sensor k, dada pelas funções `base.getKthSonarReading(k)` ou `base.getKthLaserReading(k)`.



(a) Sonar (b) Laser

Figura 3: configuração dos sensores sonar e laser.

1. Implementação do método de mapeamento com LOG-ODDS usando LASER

Complete a função `mappingWithLogOddsUsingLaser` da classe `Robot` no arquivo `Robot.cpp`. Essa função deverá fazer a atualização dos valores de ocupação das células no campo de visão do laser do robô.

Utilize $\alpha = 0.1$ (isto é, $0.1m$) como a largura dos obstáculos e $\beta = 1.0$ (isto é, 1 grau) como abertura do sensor.

Defina valores fixos de ocupação para áreas ocupadas (p_{occ}) e áreas livres (p_{free})

$$0.0 < p_{free} < 0.5 < p_{occ} < 1.0$$

Os valores escolhidos impactarão na velocidade de atualização da ocupação das células. Na prática, serão utilizados os valores em *log-odds*, isto é:

$$l_{occ} = \log \frac{p_{occ}}{1 - p_{occ}} \qquad l_{free} = \log \frac{p_{free}}{1 - p_{free}}$$

Dada uma célula qualquer do grid (ex. `Cell* c = grid->getCell(i,j);`), podemos acessar os valores de *log-odds* e ocupação dela em `c->logodds` e `c->occupancy` respectivamente. Embora o método de atualização seja aplicado sobre `c->logodds`, como desejamos visualizar o passo-a-passo do

mapeamento também precisamos atualizar constantentemente `c->occupancy`, pois este valor (normalizado entre 0 e 1) que é ilustrado no mapa em tons de cinza, conforme implementado em `drawCell` em `Grid.cpp`. A conversão de *log-odds* para probabilidades pode ser feita através de:

```
c->occupancy = getOccupancyFromLogOdds(c->logodds);
```

2. Implementação do método de mapeamento HIMM usando LASER

De forma análoga à questão 1, complete a função `mappingWithHIMMUsingLaser` da classe `Robot` no arquivo `Robot.cpp`. Essa função deverá fazer a atualização dos valores de ocupação das células no campo de visão do laser do robô.

Utilize novamente $\alpha = 0.1$ (isto é, $0.1m$) como a largura dos obstáculos e $\beta = 1.0$ (isto é, 1 grau) como abertura do sensor.

A ocupação de cada célula é dada por um contador (valor inteiro) e acessada em `c->himm`. Sempre que a célula cair em uma região de obstáculo do sensor, incremente o contador em 3 unidades. Sempre que a célula cair em uma região livre decremente o contador em 1 unidade. Limite o valor máximo de ocupação em 15 e mínimo em 0.

3. Implementação do método de mapeamento usando SONAR

De forma análoga às questões anteriores, complete a função `mappingUsingSonar` da classe `Robot` no arquivo `Robot.cpp`. Essa função deverá fazer a atualização dos valores de ocupação das células no campo de visão do sonar do robô.

Agora utilize $\alpha = 0.1$ (isto é, $0.1m$) como a largura dos obstáculos e $\beta = 30.0$ (isto é, 30 graus) como abertura do sensor.

Implemente o modelo de sensor invertido proposto por Murphy, que dá maior peso as células mais próximas do robô (r pequeno) e mais próximas do eixo acústico do sonar (α pequeno).

$$OccUpdate = 0.5\times \left(\frac{\frac{R-r}{R} + \frac{\beta-\alpha}{\beta}}{2} \right) + 0.5 \qquad \text{(Região 1) valores de 0.5 a 1}$$

$$OccUpdate = 0.5\times \left(1.0 - \frac{\frac{R-r}{R} + \frac{\beta-\alpha}{\beta}}{2} \right) \qquad \text{(Região 2) valores de 0 a 0.5}$$

ATENÇÃO: neste método a atualização da ocupação de cada célula é feita através de um produto de probabilidades.

$$Occ = \frac{OccUpdate \times Occ}{(OccUpdate \times Occ) + ((1.0 - OccUpdate) \times (1.0 - Occ))}$$

É importante evitar que o valor de Occ chegue em 0 ou 1, pois nesse caso ele nunca mais pode ser alterado. Por exemplo, após a atualização da ocupação limite o valor para ficar entre 0.01 e 0.99.

Para não sobrescrever os valores de ocupação atualizados pelo método da questão 1, acesse os valores de ocupação de cada célula em `c->occupancySonar`.