Trabalho Prático 3 – Exploração e Mapeamento

Robótica Móvel

11/07/2024

1. Introdução

A exploração e mapeamento de ambientes desconhecidos, são dois conceitos extremamente importantes na robótica. Tais habilidades permitem que robôs sejam utilizados para realizar reconhecimento e mapeamento de áreas onde não é possível, ou ao menos, não é desejado, que essa tarefa seja realizada por humanos. Visando fixar estes conceitos que são frequentemente utilizados em aplicações do mundo real, implementamos o algoritmo de de Campos Potenciais para movimentar um robô Kobuki, não holonômico, que deve navegar em um ambiente não conhecido de forma autônoma e em seguida gerar uma representação do mapa a partir das leituras do laser e utilizando o algoritmo de Occupancy Grid.

2. Execução

Para execução do trabalho, basta abrir a cena que foi enviada juntamente do código e em seguida rodar as células de forma sequencial na ordem em que estão no próprio notebook. Para seleção do caminho executado pelo robô, basta descomentar as variáveis que setam a posição e orientação iniciais do Kobuti, assim como o array de posições que representam o caminho a ser seguido e comentar aquelas que representam o caminho não escolhido.

3. Navegação

Para realizar a navegação do nosso robô Kobuti, utilizamos funções que já haviam sido estudadas e implementadas em atividades anteriores da disciplina. A movimentação, portanto, é feita a partir do algoritmo de campos potenciais, o que permite ao nosso robô, lidar com ambientes com obstáculos desconhecidos, sejam eles estáticos ou dinâmicos. Para os goals utilizados durante a execução do algoritmo, foram escolhidas posições pseudo aleatórias no mapa, de forma empírica, que possibilitassem com que o robô percorresse todo o ambiente com o laser.

4. Implementação

Visando implementar o algoritmo de Occupancy Grid, iniciamos criando uma função para construir o grid. Essa função recebe o tamanho real (ou uma estimativa) e o tamanho de cada célula, ambos em metros, e retorna uma matriz com todos os valores inicializados em 0.5. Esse valor inicial indica incerteza e facilita a implementação da função de mapeamento do Occupancy Grid, já que seu log-odds é zero.

Também criamos uma função que retorna o par de células onde o robô se encontra, a partir da posição real dele no mapa. Sabendo que há posições negativas no mapa, sempre somamos metade do número de células para mover a origem global para o centro

da matriz. Utilizando a mesma lógica, implementamos fórmulas vistas em sala para mapear em quais células os pontos finais de cada laser incidem.

Além disso, desenvolvemos duas funções adicionais: uma para transformar probabilidades em log-odds e outra para fazer o processo inverso.

Com essas funções implementadas, iniciamos a construção do Occupancy Grid, seguindo o pseudocódigo apresentado em sala. O uso do l_0 não foi necessário, já que ele era zero (log-odds de 0.5).

Finalmente, implementamos o Inverse Range Sensor Model. Para isso, consideramos as células por onde um laser passa até atingir seu ponto final. Se o laser detecta algo e essa célula é o ponto final, retornamos l_{occ}; caso contrário, retornamos l_{free}. Definimos esses valores como l_{occ} = log-odds(0.9) e l_{free} = log-odds(0.1).

1: **Algorithm occupancy_grid_mapping**($\{l_{t-1,i}\}, x_t, z_t$):
2: for all cells $\mathbf{m}_i$ do
3: if $\mathbf{m}_i$ in perceptual field of z_t then
4: $l_{t,i} = l_{t-1,i} + \textbf{inverse\_sensor\_model}(\mathbf{m}_i, x_t, z_t) - l_0$
5: else
6: $l_{t,i} = l_{t-1,i}$
7: endif
8: endfor
9: return $\{l_{t,i}\}$

(a) Occupancy Grid Mapping

```python
def inverse_range_sensor_model(x:int, y:int, x_laser:int, y_laser:int, hit:bool = True) -> float:
    """…
    if (x != x_laser or y != y_laser) or not hit:
        return L_FREE

    if x == x_laser and y == y_laser:
        return L_OCC
```

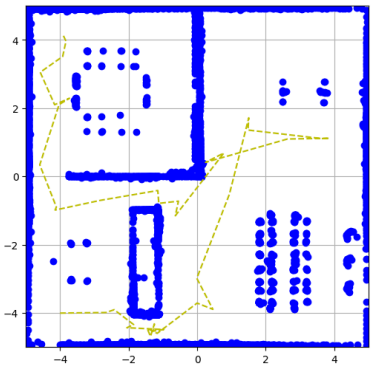
(b) Inverse Range Sensor Model

5. Testes

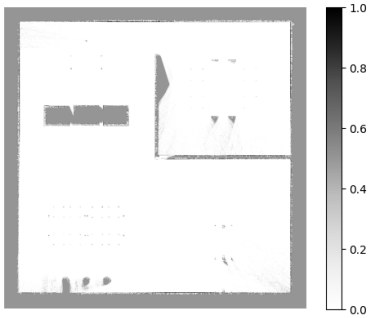
Com os algoritmos implementados, iniciamos nossos testes. Para torná-los mais realistas, adicionamos ruídos aleatórios ao laser, fazendo-o variar 10 centímetros para mais ou para menos. Em seguida, selecionamos três tamanhos de células: 0,01, 0,05 e 0,1 metros.

Para determinar qual desses tamanhos seria o mais adequado, realizamos testes em um cenário estático. O robô percorreu dois caminhos diferentes, gerando um mapa para cada um desses grids, como ilustrado abaixo.

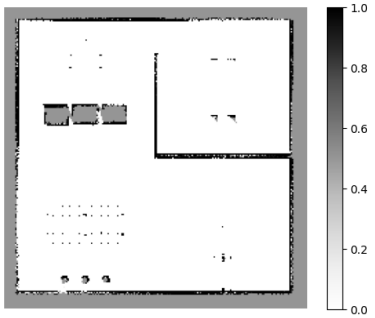
- Cena Estática
    - Path 1 (começando no canto inferior esquerdo)



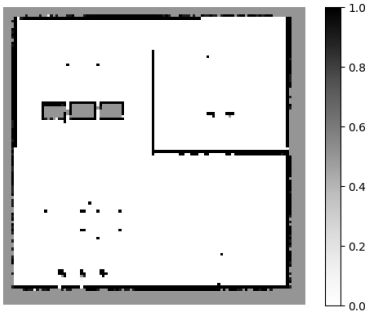
Plot Incremental



tamanho da célula = 0.01 m

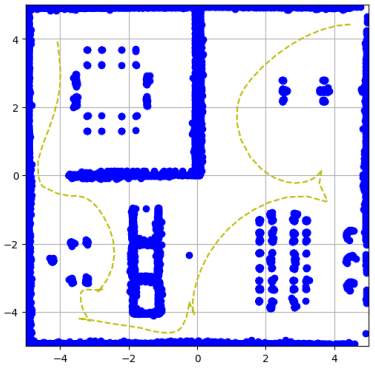


tamanho da célula = 0.05 m

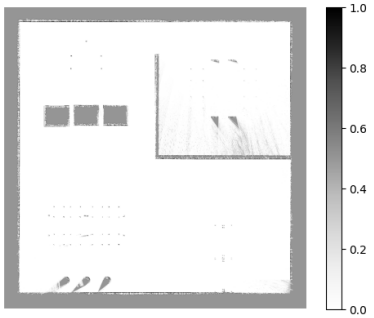


tamanho da célula = 0.1 m

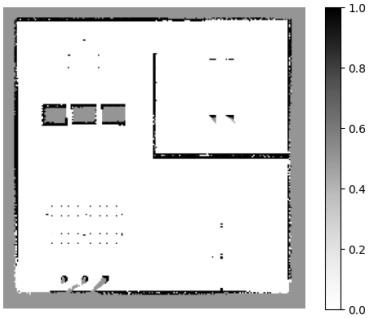
- Path 2 (começando no canto superior direito)



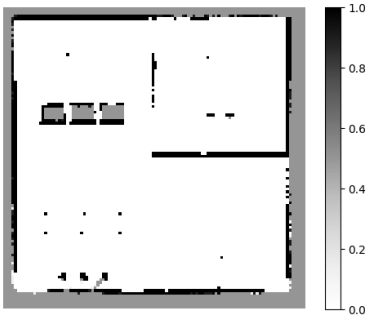
Plot incremental



tamanho da célula = 0.01 m



tamanho da célula = 0.05 m

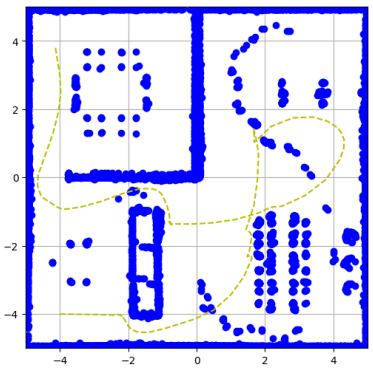


tamanho da célula = 0.1 m

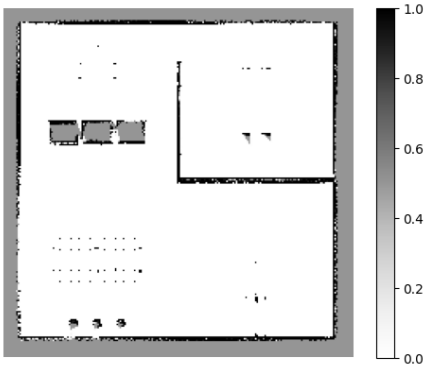
Como podemos observar, células maiores geram mapas com maior convicção, evidenciada pelo maior contraste entre zonas pretas e brancas. Em contraste, mapas com células menores apresentam contornos mais finos e detalhados.

Além disso, é importante ressaltar que um maior número de células torna o algoritmo mais custoso. Isso ocorre porque há mais valores a serem atualizados a cada iteração. Esse aumento no custo computacional é perceptível durante a movimentação do robô, que se torna menos responsiva devido ao maior tempo necessário para completar cada iteração. Tendo isso em mente, visando rodar o algoritmos na cena dinâmica, optamos pelas células de 5 centímetros uma vez que elas apresentaram uma boa mistura entre certeza, detalhes e eficiência.

- Cena Dinâmica
  - Path 1 (começando no canto inferior esquerdo)

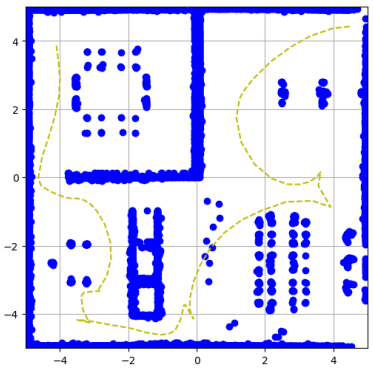


Plot incremental

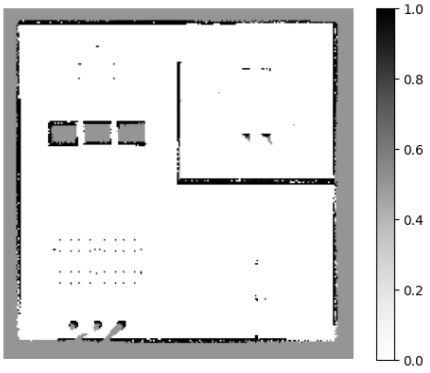


Occupancy Grid

  - Path 2 (começando no canto superior direito)



Plot incremental



Occupancy Grid

Por fim, no cenário em que há uma pessoa andando, podemos ver que o robô conseguiu mapear o ambiente de maneira correta, não adicionando o humano como uma parte do cenário, mesmo tendo visto ela inúmeras vezes, como no primeiro caminho. Observamos também que no segundo caminho menos

6. Conclusão

Após a conclusão deste trabalho prático podemos afirmar que ele cumpre seu papel, uma vez que este foi muito útil no seu papel de fixar a matéria de mapeamento. Nesse processo encontramos como principais dificuldades a construção do occupancy grid, que embora simples, gerou algumas dúvidas durante sua implementação. Portanto, esse trabalho foi de grande utilidade para fixar um conceito muito útil que será de grande utilidade para o Projeto Final.

7. Bibliografia

- Documentação da API CoppeliaSim, https://manual.coppeliarobotics.com/en/apiFunctions.htm
- GitHub da disciplina, https://github.com/verlab/dcc042-robotica-movel
- Slides de aula, https://homepages.dcc.ufmg.br/~doug/cursos/doku.php?id=cursos:roboticamovel:index
- Robótica Móvel • DCC-UFMG. Aula 18 - Mapeamento: Occupancy Grid (Grade de Ocupação). YouTube, 26/07/2021. https://www.youtube.com/watch?v=aROLZ8zB-2Y&t=3077s&ab_channel=Rob%C3%B3ticaM%C3%B3vel%E2%80%A2DCC-UFMG