Projeto 5

Gustavo Cayres, Gustavo Covas e Pedro Marcondes

MAC0318 - Introdução à Programação de Robôs Móveis

1. Parte A - Navegação por Segmentos de Retas

Ao utilizarmos a navegação por segmento de retas do robô, percebemos que trajetórias mais curtas estão menos propensas a erros. A última trajetória, constituída de grandes caminhos retos, foi a que geral maior erro, provavelmente devido aos pequenos erros que se acumularam a medida que o robô andava sem verificar seu tacômetro. Apesar disso, os resultados foram bastante bons, com os erros não ultrapassando 11 centímetros, comprimento aproximado do eixo do robô.

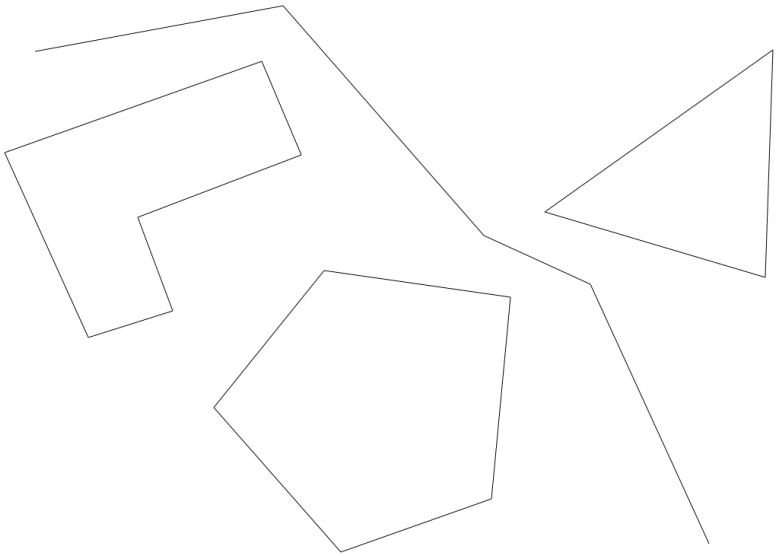

Figura 1: Trajetória P1-P2-P6-P5-P10. O erro em relação a meta foi de 4 cm.

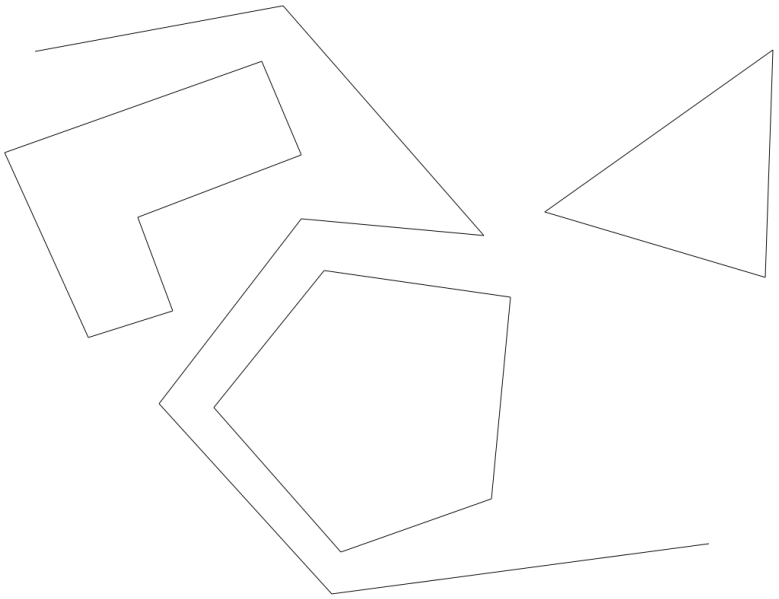

Figura 2: Trajetória P1-P2-P6-P7-P8-P11-P10. O erro em relação a meta foi de 6 cm.

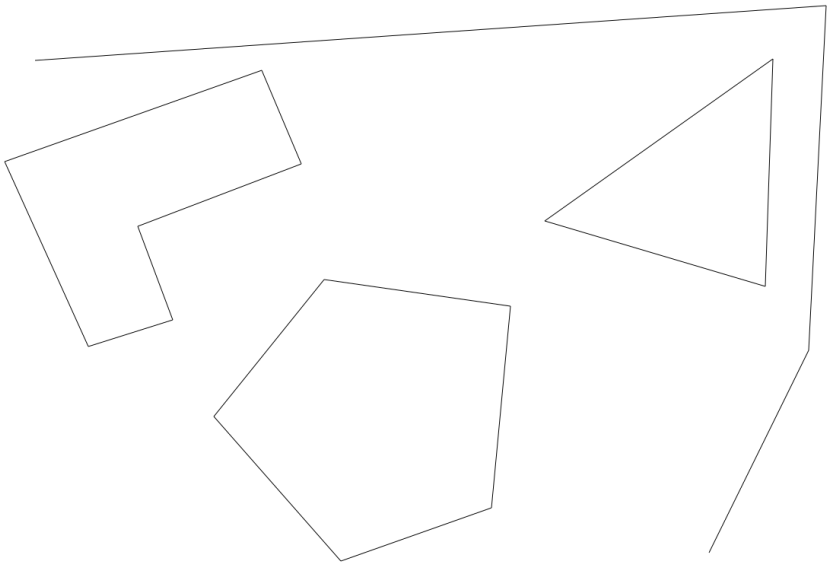

Figura 3: Trajetória P1-P3-P4-P10. O erro em relação a meta foi de 7 cm.

2. Parte B - Busca em custo uniforme

A fim de realizarmos a busca de custo uniforme no mapa topológico, implementamos uma classe de grafos, o que nos permitiu usar o algoritmo de Dijkstra clássico para encontrar o menor caminho entre dois pontos num grafo não-dirigido com pesos. O grafo foi formado obtido de forma intuitiva, com os *waypoints* do mapa tornando-se vértices, as ligações entre os *waypoints* tornando-se arestas e o peso de cada aresta sendo igual a seu comprimento.

2.1. Algoritmo

Para cada vértice do grafo, são mantidas as seguintes informações:

- A distância deste vértice até a origem do percurso (no início da busca, vale ∞ para todos os vértices diferentes da origem, e para esta vale 0).
- O vértice "pai", a partir do qual o algoritmo chegou neste vértice.
- Se este vértice já foi visitado nesta busca ou não.

Em cada iteração, o algoritmo:

1. Procura o vértice V que ainda não foi visitado e que está a menor distância da origem (incluindo a própria origem).
2. Marca V como visitado.
3. Para cada vizinho W de V, se W não foi visitado, atualiza a distância de W para o mínimo entre *distância de W* e *distância de V + peso da aresta V-W*.
4. Se a distância de W foi alterada, o vértice V será marcado como "pai" de W.

No final da busca, o menor caminho entre a origem e o destino pode ser obtido olhando-se os pais de cada vértice (o penúltimo vértice do caminho é o pai do destino, o antepenúltimo é o pai desse, e assim até atingirmos a origem).

2.2. Resultados

2.2.1. Caminho P1 - P10

- Trajetória escolhida: P1 - P2 - P6 - P5 - P10 (Vídeo da trajetória P1 - P10).
- Distância final da meta: 4,4 cm.

2.2.2. Caminho P11 - P1

- Trajetória escolhida: P11 - P8 - P7 - P6 - P2 - P1 (Vídeo da trajetória P11 - P1).
- Distância final da meta: 3,2 cm.

3. Parte C - Mapa de ocupação

3.1. Algoritmo "Frente de Onda"

Para utilizar mapa de ocupação, o mapa foi dividido em células de tamanho fixo. Uma célula que não contém um obstáculo é considerada como livre, enquanto uma célula que contém um obstáculo é considerada ocupada. A fim de decidirmos se uma célula está ou não ocupada, cada célula foi tratado como um quadrado e, caso o lado de algum obstáculo cruze algum lado desse quadrado, consideramos que a célula contém o obstáculo. Essa intersecção foi calculada através da função *intersectsAt* da classe *lejos.geom.Line*.

A busca a partir da meta foi feita com conectividade 4, enquanto a geração do caminho, a partir da origem, foi realizada com conectividade 8. A imagem 4 mostra a matriz das distâncias após a busca.

22	21	20	21	22	23	24	25	26	27	28	29	30	31	32	inf	inf	33	inf
21	20	19	20	21	22	23	24	25	26	27	-1	-1	-1	inf	inf	33	32	33
20	19	18	19	20	21	22	23	24	-1	-1	-1	inf	-1	-1	inf	32	31	32
19	18	17	18	19	20	21	22	-1	-1	inf	inf	inf	inf	-1	32	31	30	31
18	17	16	17	18	19	20	21	-1	-1	inf	-1	-1	inf	-1	31	30	29	30
17	16	15	16	17	18	19	20	21	-1	-1	-1	-1	inf	-1	-1	29	28	29
16	15	14	15	16	17	-1	-1	22	23	22	21	-1	inf	inf	-1	28	27	28
15	14	13	14	15	-1	-1	-1	-1	22	21	20	-1	-1	inf	-1	-1	26	27
14	13	12	13	-1	-1	inf	inf	-1	-1	20	19	20	-1	-1	-1	24	25	26
13	12	11	-1	-1	inf	inf	inf	inf	-1	-1	18	19	-1	21	22	23	24	25
12	11	10	-1	inf	inf	inf	inf	inf	inf	-1	17	18	19	20	21	22	23	24
11	10	9	-1	inf	inf	inf	inf	inf	inf	-1	16	17	18	19	20	21	22	23
10	9	8	-1	-1	inf	inf	inf	inf	inf	-1	15	16	17	18	19	20	21	22
9	8	7	6	-1	inf	inf	inf	inf	-1	-1	14	15	16	17	18	19	20	21
8	7	6	5	-1	-1	-1	-1	-1	-1	12	13	14	15	16	17	18	19	20
7	6	5	4	5	6	7	8	9	10	11	-1	-1	16	17	18	19	20	21
6	5	4	3	4	5	6	7	8	9	10	-1	-1	-1	18	19	20	21	22
5	4	3	2	3	4	5	6	7	8	9	-1	inf	-1	19	20	21	22	23
4	3	2	1	2	3	4	5	6	7	-1	-1	inf	-1	-1	21	22	21	22
3	2	1	0	1	2	3	4	5	6	-1	inf	inf	inf	-1	-1	21	20	21
4	3	2	1	2	3	4	5	6	7	-1	inf	inf	inf	inf	-1	-1	19	20
5	4	3	2	3	4	5	6	7	8	-1	-1	-1	-1	-1	-1	-1	18	19
6	5	4	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
7	6	5	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

Figura 4: Matriz das distâncias para o caminho P1 a P10. O valor *0* marca a meta, *inf* são os pontos que não foram atingidos na busca, ou porque estavam envoltos por obstáculos (representados com *-1*), ou porque a busca atingiu a origem e terminou antes de atingi-los.

3.2. Algoritmo para dilatação de imagens binárias

A dilatação foi feita expandindo-se cada célula ocupada C para toda a célula não ocupada C' tal que C' esteja à distância 1 de C, utilizando conectividade 8 (em outras palavras, só as células 8-vizinhas são preenchidas). Portanto o tamanho da dilatação é análogo ao tamanho da célula utilizada.

3.3. Algoritmo de linearização

O algoritmo de linearização faz uso da função *intersectsAt* do LeJOS, que verifica se há um ponto de intersecção entre duas linhas. Sejam O a origem e D o destino, as iterações da linearização tem essa forma:

1. Encontrar o ponto P de maior índice no caminho (mais distante da origem) de forma que haja um caminho reto não obstruído entre a origem e P.
2. Se P for a própria origem, não são feitas modificações no caminho e o algoritmo termina.
3. Se P for diferente da origem, então todos os pontos entre a origem e P são eliminados do caminho, e o algoritmo de linearização é aplicado entre P e o destino.

A fim de aumentarmos a eficiência da linearização, preferimos fazer a implementação iterativa do algoritmo, embora sem modificações na lógica.

3.4. Resultados

3.4.1. Caminho P1 - P10

Linearização:

- *Caminho não-linearizado (em* mm): (150, 850) → (200, 850) → (250, 850) → (300, 850) → (350, 850) → (350, 850) → (400, 800) → (450, 750) → (500, 700) → (550, 650) → (600, 600) → (650, 550) → (700, 500) → (750, 450) → (800, 400) → (850, 350) → (900, 300) → (950, 250) → (950, 200) → (950, 150)

- *Caminho linearizado (em* mm): (150, 850) → (400, 800) → (700, 500) → (750, 450) → (950, 150)

Dilatação:

- Sem dilatação:
  - Distância da meta: 10 cm.
  - Comentários: Robô encosta as rodas nos obstáculos durante o percurso.

- Com dilatação (células 5x5 cm^2):
  - Distância da meta: 6 cm.
  - Comentários: O aumento da dilatação leva o robô a manter uma certa distância dos obstáculos, mas uma dilatação muito grande impediu que o robô encontrasse uma rota para a meta. Com células de lado 3 cm não houve diferença perceptível na rota. A melhor rota foi obtida com células de 5 cm (aproximadamente metade do comprimento do eixo), cujo erro foi reportado acima.

3.4.2. Caminho P11 - P1

Linearização:

- *Caminho não-linearizado (em* mm): (400, 150) → (350, 200) → (300, 250) → (250, 300) → (200, 350) → (150, 350) → (100, 400) → (050, 450) → (000, 500) → (000, 550) → (000, 600) → (000, 650) → (050, 700) → (100, 750) → (100, 800)
- *Caminho linearizado (em* mm): (400, 150) → (0, 600) → (100, 800)

Dilatação:

- Sem dilatação:
  - Distância da meta: 1 cm.
  - Comentários: O robô atravessou o bico de um dos obstáculos.
- Com dilatação (células 5x5 cm^2):
  - Distância da meta: 17 cm.
  - Comentários: A dilatação levou o robô a realizar um caminho longo e a efetuar muitas curvas. Isso levou a uma grande que na precisão, levando o robô a cruzar alguns obstáculos e a parar longe da meta.

4. Parte D - Mapa de visibilidade

4.1. Algoritmo de dilatação

A dilatação das linhas é feita substituindo cada linha por um polígono que circunde a linha. Em nosso algoritmo, decidimos por substituir as linhas por retângulos com 2 lados paralelos à linha e 2 lados perpendiculares àquela (figura 6). Em detalhes, aumentamos a linha em um determinado tamanho que desejávamos dilatar o mapa, depois calculamos a reta perpendicular a

esse segmento expandido na duas bordas dele e com isso geramos os 4 pontos que formam esses retângulos em volta dos segmentos dos polígonos originais. Foi adotada uma dilatação de 5cm ao redor dos segmentos de reta.

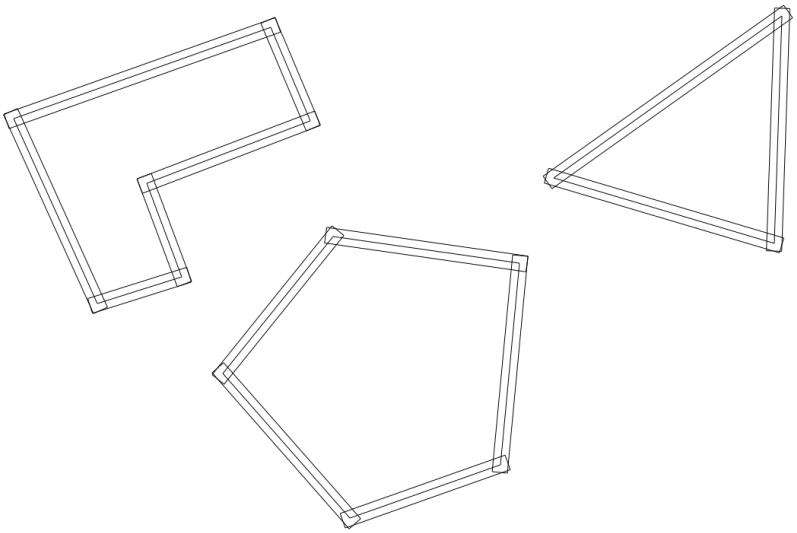

Figura 5: Efeito da dilatação sobre o mapa do projeto.

4.2. Obtenção do mapa de visibilidade

O mapa de visibilidade foi obtido ao marcarmos os vértices dos polígonos resultantes da dilatação como *waypoints*. Foi necessário, então, estipularmos a mínima distância entre 2 vértices para que esses sejam considerados como *waypoints* distintos, já que o algoritmo de dilatação inevitavelmente cria aglomerados de vértices nas regiões das extremidades das linhas. A partir desses *waypoints*, as arestas foram criadas entre quaisquer pares de vértices entre os quais não haja obstáculos (verificação realizada pelo *intersectsAt*).

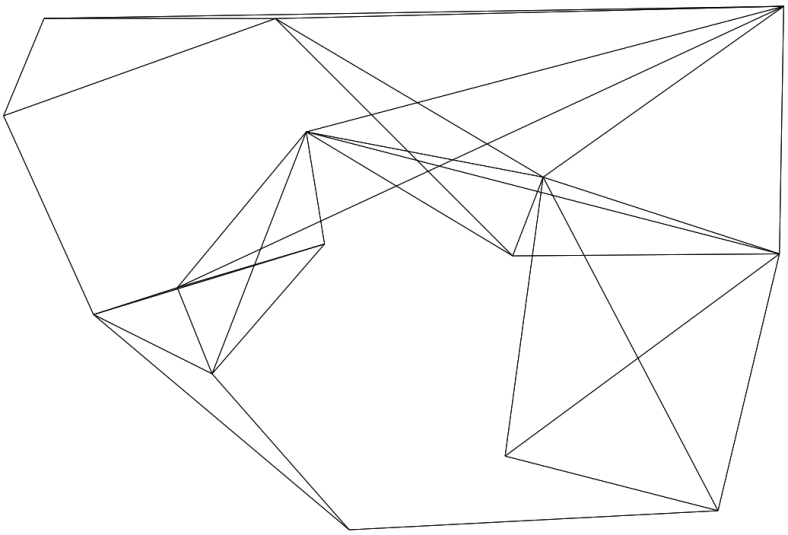

Figura 6: Mapa de visibilidade obtido.

4.3. Results

4.3.1. Caminho P1 - P10

- Distância da meta: 5 cm.
- Comentários: O robô acabou tocando os obstáculos durante alguns momentos do percurso, entretanto não houve cruzamento.

4.3.2. Caminho P11 - P1

- Distância da meta: 3 cm.
- Comentários: Robô percorreu com sucesso os polígonos pelo lado externo (caso ideal).

5. Comparação

A partir dos nossos experimentos, concluímos que os *waypoints* iniciais, presentes no mapa topológico, já foram suficientes para gerar bons resultados para o mapa estudado. Entretanto, o aumento do número de *waypoints* no mapa de visibilidade levou o robô a ter um comportamento mais preciso, o que, somado à dilatação de linhas, permitiu que ele realizasse o percurso

mantendo uma distância saudável dos obstáculos. Portanto esse método pode ser mais apropriado para problemas reais. Os piores resultados foram obtidos no mapa de ocupação, tanto em relação ao tempo de execução quanto à distância final da meta.