Instituto Tecnológico de Aeronáutica - ITA
Inteligência Artificial para Robótica Móvel - CT213
Aluno:

Relatório do Laboratório 2 - Busca Informada

1 Breve Explicação em Alto Nível da Implementação

A implementação da fila de prioridade foi feita como recomendado. Além disso, nos três algoritmos foi utilizado o seguinte trecho de código para garantir que os nós só fossem fixados (removidos da fila) uma vez:

```
g, node_temp = heapq.heappop(pq) # ou f
if not self.node_grid.get_node(node_temp.i, node_temp.j).closed:
        node = node_temp
        node.closed = True
else:
        continue
if node.i == goal_position[0] and node.j == goal_position[1]:
        break
```

Desse modo, o nó só é acrescido à análise uma vez, evitando de que consultas antigas a um mesmo nó não sejam implementadas após que este já esteja fixado no caminho.

1.1 Algoritmo Dijkstra

Como para o Dijkstra não considera-se o objetivo, foi implementado a busca com o custo sendo dado por g. O custo de g, por sua vez, foi dado pelo custo incremental entre células:

$$C(c_i, c_j) = f\frac{C(c_i) + C(c_j)}{2}, \tag{1}$$

com os fatores sendo adotados como sugerido no roteiro. Utilizou-se o método "*get_ cell_ cost*"do *cost map* da classe *PathPlanner* para obter os valores individuais de custo das célula.

Para garantir que o problema não fique preso num loop, seguiu-se a proposição geral para acréscimo de elementos e, ainda, para a inserção dos sucessores foi feito o condicional:

```
if successor.g > node.g + cost_i_to_j and (not successor.closed):
        successor.g = node.g + cost_i_to_j
        successor.parent = node
        heapq.heappush(pq, (successor.g, successor))
```

O condicional garante que não está sendo adicionado à fila nós que já estão fixados. Esta lógica se repete nos demais algoritmos.

1.2 Algoritmo *Greedy Search*

Para o *Greedy Search*, foi utilizado apenas o $h(successor, goal)$ da heurística, com auxílio do método "*distance_to*"do atributo *node_grid* da classe *PathPlanner*. O valor de g ainda foi calculado, para comparação do custo com os demais algoritmos. Por fim, teve-se o condicional para a análise do sucessor:

```
if not successor.closed:
        successor.g = node.g + cost_i_to_j
        successor.f = h
        successor.parent = node
        heapq.heappush(pq, (successor.f, successor))
```

1.3 Algoritmo A*

Utilizou-se para o A* o custo f, utilizando a heurística e o custo entre células dos métodos anteriores. O condicional para o sucessor foi implementado como a seguir, análogo aos demais métodos:

```
if successor.f > node.g + cost_i_to_j + h and (not successor.closed):
        successor.g = node.g + cost_i_to_j
        successor.f = successor.g + h
        successor.parent = node
        heapq.heappush(pq, (successor.f, successor))
```

2 Figuras Comprovando Funcionamento do Código

Inseridas as figuras da primeira e da décima iterações.

2.1 Algoritmo Dijkstra

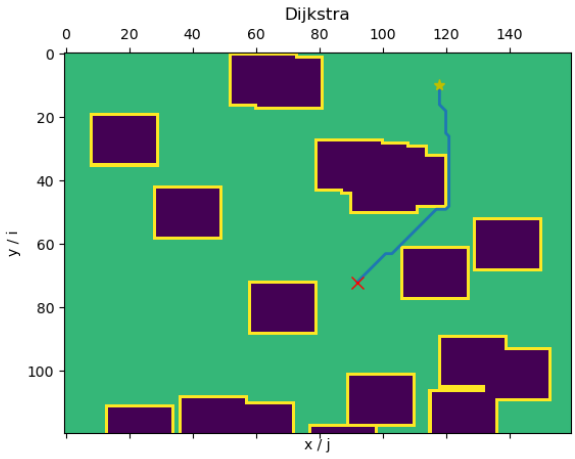

Figura 1: Primeira iteração do algoritmo dijkstra.

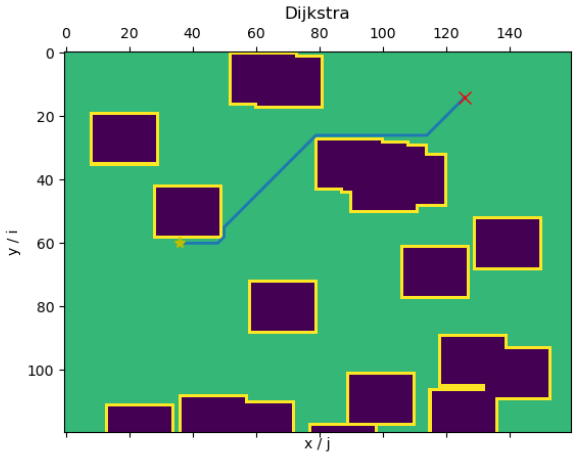

Figura 2: Décima iteração do algoritmo dijkstra.

2.2 Algoritmo *Greedy Search*

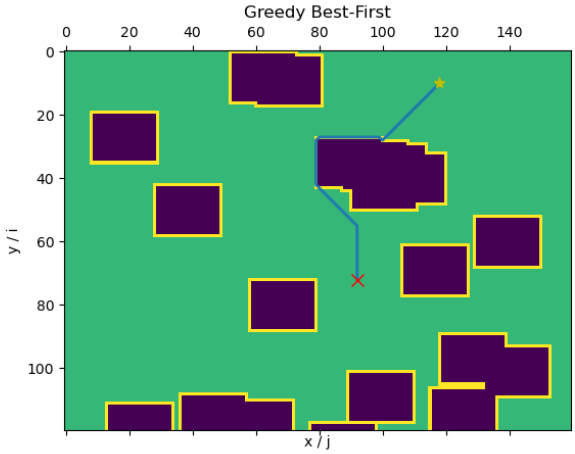

Figura 3: Primeira iteração do algoritmo greedy.

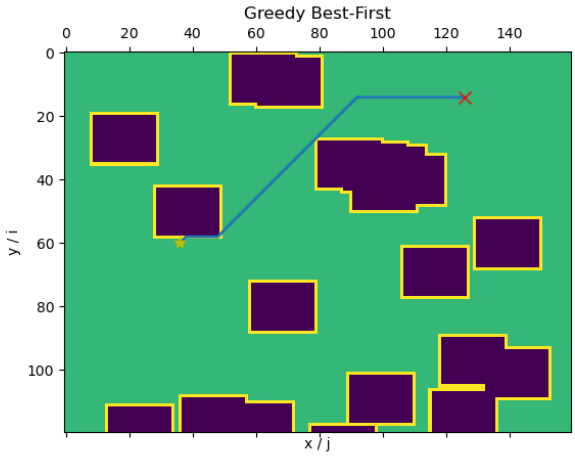

Figura 4: Décima iteração do algoritmo greedy.

2.3 Algoritmo A*

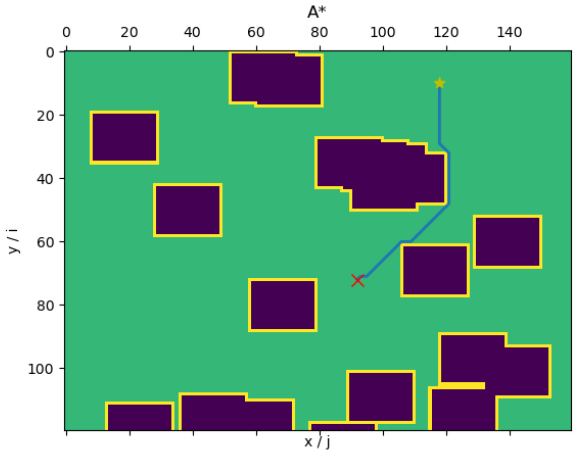

Figura 5: Primeira iteração do algoritmo A*.

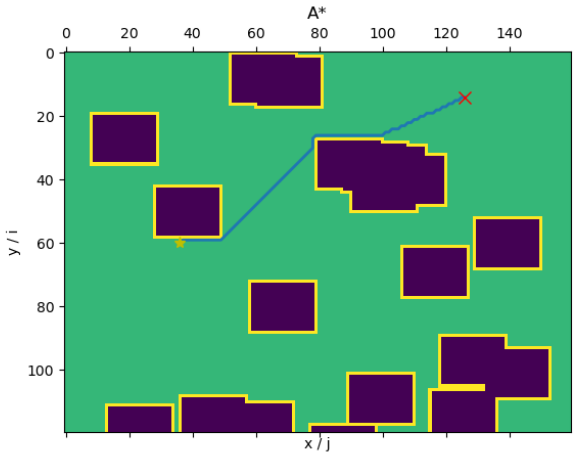

Figura 6: Décima iteração do algoritmo A*.

3 Comparação entre os algoritmos

Tabela 1 com a comparação do tempo computacional, em segundos, e do custo do caminho entre os algoritmos usando um Monte Carlo com 100 iterações.

Tabela 1: tabela de comparação entre os algoritmos de planejamento de caminho.

Algoritmo	**Tempo computacional (s)**		**Custo do caminho**	
	Média	**Desvio padrão**	**Média**	**Desvio padrão**
Dijkstra	0.1282	0.0709	79.8292	38.5710
Greedy Search	0.0057	0.0013	105.0559	63.6133
A*	0.0315	0.0269	79.8292	38.5710