Instituto Tecnológico de Aeronáutica

CT-213

Laboratório 2 – Busca Informada

Aluno:
Pedro Elardenberg Sousa e Souza

Professor:
Marcos Ricardo Omena de Albuquerque Maximo

9 de abril de 2021

São José dos Campos, Brasil

Nas seções a seguir, serão comentadas as particularidades de implementação dos algoritmos para a linguagem *Python* e para o presente lab, em comparação com os pseudocódigos dados na aula.

Considerações iniciais:

Primeiramentem, o código recebe as posições (x, y) dos nós, não os nós em si. Para isso, foi utilizada a função *get_node(position[0], position[1])* do objeto *node_grid*. Para a execução de Monte Carlo, a fila de prioridades foi esvaziada antes de se inserir o primeiro nó da nova busca.

Além disso, foi necessário declarar, no método *__init__*, a fila de prioridades *pq*, e os nós *node*, *successor_node* e *goal_node* que representam, respectivamente, o primeiro nó na fila de prioridades, os nós adjacentes e o nó cuja posição se deseja chegar.

Para conseguir os sucessores no *for*, foi utilizada a função *get_successors* do objeto *node_grid*, passando como parâmetro a posição do nó *node*.

1 Algoritmo de Dijkstra

Depois de buscar e retirar o primeiro nó da fila de prioridades (isto é, aquele cuja distância *node.f* à posição inicial é a menor dentre os nós na fila), verifica-se se este é o nó alvo *(goal_node)*. Se não, busca em seus sucessores e, se há algum cujo custo *successor_node.f* é menor que o custo *node.f* + o custo da aresta entre esses dois nós, calculado com a função *get_edge_cost(node.get_position(), successor_node.get_position())* do objeto *node_grid*, o valor é atualizado e o nó sucessor é adicionado na fila de prioridades, tendo como *node* seu nó pai.

Quando o *goal_node* é encontrado, a função retorna o custo do nó e o caminho que ele percorreu, utilizando o método *construct_path*.

A imagem abaixo ilustra o caminho percorrido com o algoritmo de Dikjstra.

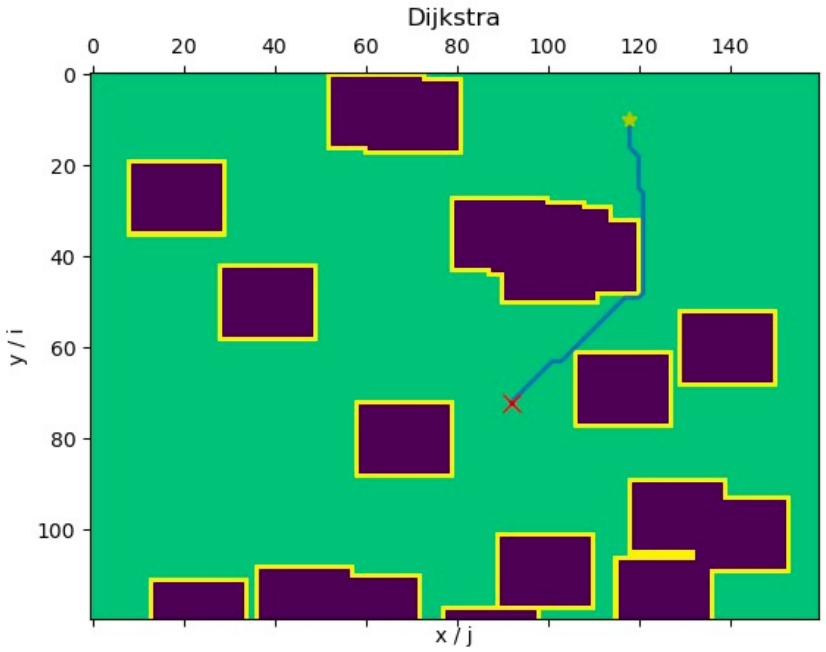

Figura 1: Caminho do robô com o Dikjstra

2 Busca Gulosa

Para a busca gulosa, primeiro define-se as variáveis g, que é o custo de um nó à posição inicial, e f, que é a distância euclidiana do nó ao objetivo.

Após um nó ser visitado, muda-se o estado da variável booleana *closed* para true. Ao buscar os nós sucessores no *for*, se *sucessor_node.closed* for False, atualiza-se de *sucessor_node* o nó pai e o valor f, e a variável g torna-se o g do nó pai + a distância entre o nó sucessor e o pai.

Com isso, o algoritmo não descobre necessariamente um caminho de custo mínimo, pois ele não analisa o custo dos nós sucessores, apenas se eles foram visitados ou não.

A imagem abaixo mostra o caminho encontrado com este algoritmo.

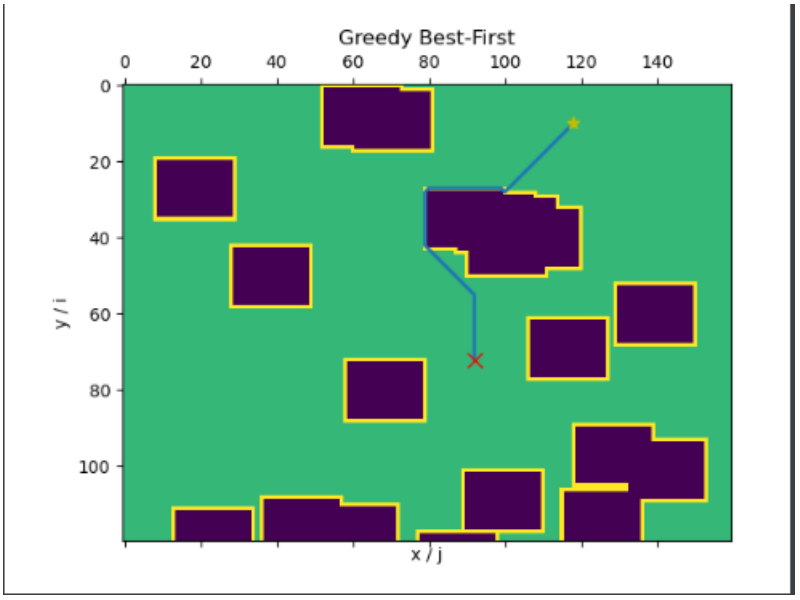

Figura 2: Caminho encontrado pelo robô com o algoritmo guloso

3 A*

Para este caso, foi utilizada uma função heurística, que busca alguma função que chegue ao objetivo. Aproveitando a ideia da busca gulosa, essa função heurística foi definida como a distância euclidiana para o nó objetivo.

Inicia-se o algorítimo definindo o valor de f e de g da posição inicial (com a função heurística e 0, respectivamente), e insere-se o primeiro nó na fila de prioridades.

Utilizando a mesma lógica da fila de prioridades dos algoritmos anteriores, enquanto a fila não é vazia, retira-se da fila o nó de menor custo, fecha-se o nó e, se ele não for o nó objetivo, busca-se seus sucessores.

Aqueles que não foram ainda visitados e cujo valor de f seja menor que a distância g do nó pai ao início do caminho + o custo da aresta entre esses nós + a distância do nó sucessor para o objetivo, atualiza-se para este o valor de f do nó sucessor, define-se *node* como o nó pai e insere-se o sucessor na fila de prioridades.

O processo para quando o *node* chega no nó objetivo *goal_node*.

A imagem abaixo mostra o caminho percorrido com o algoritmo A*.

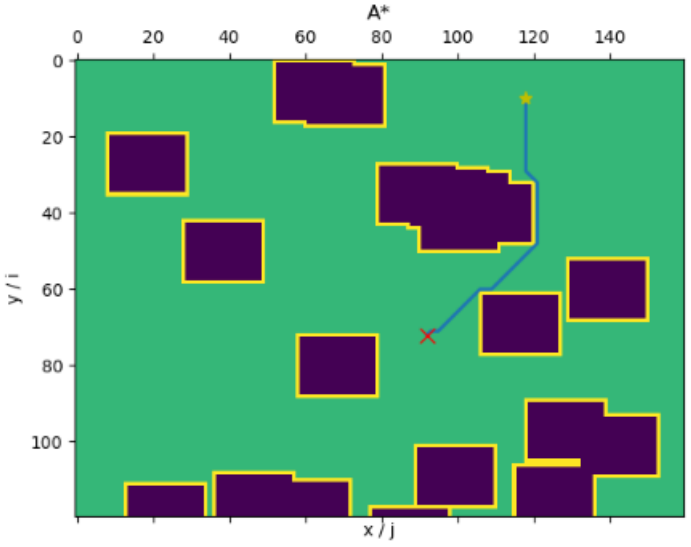

Figura 3: Caminho encontrado pelo robô com o algoritmo A*

4 Simulação de Monte Carlo

Repetindo o algoritmo 100 vezes e calculando a média e o desvio padrão da distância percorrida e o tempo em cada iteração, foi encontrado os seguintes dados.

Tabela 1: Resultados de cada algoritmo advindos da simulação de Monte Carlo

Algoritmo	**Tempo computacional (s)**		**Custo do caminho**	
	Média	**Desvio Padrão**	**Média**	**Desvio Padrão**
Dijkstra	0.229351	0.127739	79.829197	38.570962
Greedy Search	0.005458	0.000924	82.563126	41.786792
A*	0.226415	0.126580	79.829197	38.570962