SUMÁRIO

main.py

OBJETIVOS

- Deslocar o robô por 10 nós do mapa topológico
- Construir a grade de ocupação de cada nó visitado
- Com o mapa contruido ir de um ponto A ao ponto B
- Não colidir com nenhum objeto ao deslocar-se de A a B
- Exibir no terminal a posição do robô em realação ao mapa ocupacional e topológico

ABORDAGEM UTILIZADA

Mover do nó inicial para o nó B e criar o mapa ocupacional para cada nó

- Realizar uma leitura 360 do ambiente para o primeiro nó, verificando o que há atrás
- Calcular a rota para até o nó B
- Andar x distância em diração ao proximo nó, e realizar um escaneamento para o mapa
- Ao cruzar de um nó para o outro, salvar o mapa no arquivo e inicializar outro mapa para o nó atual
- Repetir até o ultimo nó
- Ao chegar no centro do nó final, realizar um untimo escaneamento e salvar o mapa

Mover do ponto inicial para o ponto final, exibindo a posição do robo no mapa no terminal

- Calcular a rota até o ponto final
- Ler do arquivo o mapa do nó atual
- Andar uma distância x em direção ao próximo nó, e exibir na tela o mapa com a posição do robo atual
- Repetir até o ultimo nó
- Ao chegar no centro do ultimo nó, mover o robo até o ponto entrado pelo usuario

cords_calc.py

Descrição

Responsável pelo cálculo de cordenadas, tais como, distância entre dois pontos, yaw entre dois pontos, entre outros.

Variáveis

Métodos

desired_yaw() => float

Descrição:
Calcula o yaw necessário para se locomover em linha reta da posição atual até o ponto final.
Ambiente:

Entradas:

Saidas:
Yaw necessário para se locomover em linha reta da posição atual até o ponto final.

waypoint_yaw() => float

Descrição:
Calcula o yaw necessário para se locomover em linha reta da posição atual até o ponto intermediário.
Ambiente:

Entradas:

Saidas:
Yaw necessário para se locomover em linha reta da posição atual até o ponto intermediário.

yaw_to_point(x,y) => float

Descrição:
Calcula o yaw necessário para se locomover em linha reta da posição atual até um ponto (x, y) qualquer.
Ambiente:

Entradas:
1 -> x(float):
Posição x
2 -> Y(float):
Posição y

Saidas:
Yaw necessário para se locomover em linha reta da posição atual até um ponto (x, y) qualquer.

in_range_of_collision(p_x, p_y, range_x=1, range_y=3) => boolean

Descrição:
Verifica se uma dada posição (p_x, p_y) está dentro de um dado range relativo ao robô, ou seja, se o ponto está dentro da área de colisão do robô.
Ambiente:

Entradas:
1-> p_x(float):
posição em x

2-> p_y(float):
posição em y
3-> range_x(float):
distância em x
4-> range_y(float):
distância em y

Saidas:

distance_between_points((p1_x, p1_y), (p2_x, p2_y)) => float

Descrição:
calcula a distância entre dois pontos
Ambiente:

Entradas:
1-> (p1_x, p1_y)(Tupla):
coordenadas x e y do ponto 1
2-> (p2_x, p2_y)(Tupla):
coordenadas x e y do ponto 2

Saidas:
distância em linha reta entre dois pontos

relative_positions() => hash

Descrição:
Calcula a posição relativa do objeto mais proximo, utilizando o sensor a laser e sonar
Ambiente:

Entradas:

Saidas:
Hash no formato:
rel_positions:
- sonar_front_x
- sonar_front_y
- laser_x
- laser_y

data.py

Descrição

Responsável por reter e atualizar todos os dados do robô, tais como sua posição, yaw, entre outros.

Variáveis

```
front_min (float)               # menor distância do sonar frente
front_min_ang (float)             # ângulo
back_min (float)                # menor distância do sonar back
back_min_ang (float)              # ângulo
hokuyo_min (float)              # menor distância do scan
hokuyo_min_ang (float)            # ângulo
hokuyo_ranges (Array[float])      # todos os ranges do scan
hokuyo_ang_inc (float)            # ângulo de incremento
abs_position_x (float)          # posição absoluta em x
abs_position_y (float)          # posição absoluta em y
abs_position_z (float)          # posição absoluta em z
roll (float)                  #
pitch (float)                   #
yaw (float)                     #

params(hash):                 # Hash para parametros que guiaram o robô
   type_sensor (string)           # tipo do sensor a ser utilizado para desvio de colisões: (laser ou sonar)
   cord_x (float)               # coordenada x do ponto final
   cord_y (float)               # coordenada y do ponto final
   cord_range (float)             # range de erro para alcance do ponto final
   waypoint_x (float)             # coordenada x do ponto intermediário
   waypoint_y  (float)            # coordenada y do ponto intermediário
```

Métodos

initialize() => nil

Descrição:

Inicializa os Subscribers para atualizar os dados do robô.

Ambiente:

Entradas:

Saidas:

is_data_initialized() => boolean

Descrição:

Verifica se todos os dados já foram inicializados.

Ambiente:

Entradas:

Saidas:

retorna verdadeiro se todos os dados já foram inicializados

print_all_data() => nil

Descrição:

Exibe no terminal todas as variáveis de data, e seus valores.

Ambiente:

Entradas:

Saidas:

set_params(key, value) => nil

Descrição:
Altera o valor do hash params.
Ambiente:

Entradas:
1-> key(string):
chave do hash params
2-> value:
valor a ser atribuido

Saidas:

callback_sonar_front(scan) => nil

Descrição:
Atualiza os dados da menor distância e ângulo encontrado pelo sonar frontal.
Ambiente:

Entradas:
1-> scan (sensor_msgs.msg.LaserScan):
dados do sonar
Saidas:

callback_sonar_back(scan) => nil

Descrição:
Atualiza os dados da menor distância e ângulo encontrado pelo sonar traseiro.
Ambiente:

Entradas:
1-> scan (sensor_msgs.msg.LaserScan):
dados do sonar
Saidas:

callback_hokuyo_scan(scan) => nil

Descrição:
Atualiza os dados da menor distância e ângulo encontrado pelo laser. Também salva todos os ranges obtidos e o ângulo de incremento.
Ambiente:

Entradas:
1-> scan (sensor_msgs.msg.LaserScan):
dados do scan
Saidas:

callback_pose(odometry) => nil

Descrição:
Alimenta os dados de posição absoluta em x, y e z, e os dados de roll, pitch e yaw.
Ambiente:

Entradas:
1-> scan (sensor_msgs.msg.Odometry):
dados da odometria
Saidas:

graph.py

Descrição

Responsável por ler o arquivo que define o mapa topológico e criar em memória o grafo do mapa.

Variáveis

```
arr_nodes[node]:                    # Array contendo todos os nós, onde cada nó é um hash:
  - center (Array[int])             # Ponto central de um nó (p_x, p_y)
  - connected_nodes (Array[int])        # Array de nós que estão conectado a este nó
  - edges (Array[Array[float]])       # Conjunto de pontos (p_x, p_y) que constituem a área do nó
  - node                          # Número do nó
```

Métodos

initialize() => nil

Descrição:
Realiza a leitura do arquivo e inicializa o arr_nodes com os dados de cada nó.
Ambiente:

Entradas:

Saidas:

is_grapho_initialized() => boolean

Descrição:
Verifica se o arr_nodes já foi inicializado
Ambiente:

Entradas:

Saidas:
retorna verdadeiro se arr_node já foi inicializado

my_node() => int

Descrição:
dado a posição atual do robô, calcula em qual nó do grafo ele esta.
Ambiente:

Entradas:

Saidas:
número correspondente ao nó

is_connected_node(n_grapho) => boolean

Descrição:
Verifica se um dado nó esta conectado ao nó atual do robô.
Ambiente:

Entradas:
1-> n_grapho(int):
número do nó no grafo.
Saidas:
verdadeiro se o dado nó está conectado ao nó atual.

is_inside_of_node((p_x, p_y), edges) => boolean

Descrição:
Verifica se um dado ponto esta dentro da área de um nó.
Ambiente:

Entradas:
1-> (p_x, p_y) (tupla):
cordenadas x e y do ponto.
2-> edges (Array[Array[float]]):
pontos que formam a área do nó.

Saidas:
verdadeiro se o ponto esta dentro da área.

calc_route(final_node) => Array[int]

Descrição:
Calcula a rota minima em quantidade de nós, para se deslocar do nó atual até um dado nó final.
Ambiente:

Entradas:
1-> final_node (int):
número correspondente ao nó final
Saidas:
conjunto de nós em ordem que devem ser visitados para atingir o nó final.

convert_graph_to_py_format() => hash

Descrição:
Converte o formato de grafo utilizado 'arr_nodes' para o formato de grafo em hash utilizado pelo padrão do python.
Ambiente:

Entradas:

Saidas:
Grafo em formato de hash onde a chave é o numero do nó e o valor e o conjunto de nós ligados.

find_shortest_path(graph, start, end, path=[]) => Array

Descrição:
Busca pelo caminho com menor quantidade de nós para se deslocar do nó inicial até o nó final.
Ambiente:

Entradas:
1-> graph(hash):
grafo em formato de hash, onde a chave é o numero do nó e o valor e o conjunto de nós ligados.
2-> start(int):
número do nó inicial
3-> end(int):
número do nó final
4-> path(Array):
nós que deseja-se obrigatóriamente visitar no meio da rota

Saidas:

print_all_nodes() => nil

Descrição:
Exibe no terminal todos os nós e seus atributos.
Ambiente:

Entradas:

Saidas:

move_logic.py

Descrição

Responsável por realizar a lógica de movimentação do robô.
Este modulo também é responsável por calcular o ponto intermediário para alcançar o ponto final, uma vez que esta função também faz parte da lógica de movimentação, assim como descrito no pseudo-código.

Variáveis

```
pub(geometry_msgs.msg.Twist)      # Publisher que escreve no nó do ros 'cmd_vel', alterando a velocidade do robô
```

Métodos

move() => boolean

Descrição:
Realiza a lógica de movimentação do robô para alcançar o ponto definido na variável params, fornecida pelo modulo 'data.py'.
Esta função realiza a movimentação por 1 periodo do rate, logo para alcançar um ponto final é necessário chama-la varias vezes.

Ambiente:

Entradas:

Saidas:
verdadeiro se o robô atingiu o ponto final.

new_waypoint() => nil

Descrição:
Calcula um novo ponto intermediário (como ilustrado no pseudo-código), alterando na variável params, fornecida pelo modulo 'data.py', as cordenadas x e y do ponto intermediário.
Ambiente:

Entradas:

Saidas:

occupational_map.py

Descrição

Este módulo corresponde à criação do mapa ocupacional de um nó.
O mapa ocupacional é constituido por uma matriz de tamanho váriavel com quantidade de linhas e colunas correspondente à escala do mapa e do tamanho do nó, onde cada célula corresponde à um pedaço do mapa real.
Toda a matriz é iniciada como indefinida, caracter X
Para cada leitura indicando presença de objeto é adicionado 0.25 à célula
Para cada leitura indicando ausência de objeto é subtraido 0.25 à célula
As células que possuem pontuação acima de 1 é considerada ocupada e as celulas que possuem pontuação abaixo de -1 são consideradas desocupadas.
A saida é escrita em um arquivo .txt indicado pelo numero do nó (map_numero.txt).

Variáveis

```
node (int)                          # nó do mapa ocupacional
x_max_global (float)                  # maior coordenada x da área do nó
x_min_global (float)                 # menor coordenada x da área do nó
y_max_global (float)                  # maior coordenada y da área do nó
y_min_global (float)                 # menor coordenada y da área do nó
blocks_number_x (int)                   # quantidade de colunas
blocks_number_y (int)                   # quantidade de linhas
occ_map_mtx = [[]] (Array[Array[float]])        # matriz do mapa ocupacional -> occ_map_mtx[x_point][y_point]
margin_size = 10 (int)                  # margem do mapa
block_size (int)                    # tamanho da divisão do mapa (resolução)
```

Métodos

initialize_new_map(node_param, block_size_param) => nil

Descrição:
Inicializa as variáveis, iniciando assim um novo mapa, relativo à um nó do grafo.
Ambiente:

Entradas:
1-> node_param (hash):
hash que contém as informações do nó para construção do mapa, este hash é o mesmo do grapho.py
2-> block_size_param (float):
escala dos blocos, 1.0 é escala real

Saidas:

print_map() => nil

Descrição:
Escreve no arquivo .txt a saida do mapa. O nome do arquivo é composto pelo número do nó (map_numero.txt).
Ambiente:

Entradas:

Saidas:

scanner() => nil

Descrição:
Responsável por fazer uma varredura utilizando os sensores, escaneando por objetos, preenchendo o mapa com espaços ocupados e desocupados
Ambiente:

Entradas:

Saidas:

set_graph.py

Descrição

Este é um script responsável por facilitar a criação dos nós do mapa topológico.
Este script não é um modulo, ele funciona idependente dos demais e possui um único objetivo de criar o arquivo contendo os nós do mapa topológico.
Abaixo segue como utiliza-lo.

Como Usar

1º Defina onde sera salvo o arquivo no linha 31, e lembre-se de colocar o endereço completo, padrão:(text/grafo.txt) 2º Rode o set_graph.py 3º Diga o número do novo nó 4º Diga os numeros dos nós adjacentes, separados por um único espaço 5º Mova o robô pelo contorno do quadrante retornando para o mesmo local de partida 6º O nó será salvo 7º Repita do 3º passo em diante ou pressione Ctrl + C para sair

OBS1: O algoritmo funciona salvando os rastros do robô com uma taxa de 1hz, e calcula o ponto central com base nos pontos salvos

FORMATO DE SAIDA:

numero_no [nos conectados] [ponto central x, ponto central y] [[passo1 x, passo1 y], [passo2 x, passo2 y], ...]

OBS: separados por TAB

OBS2: Para alterar o formato de saida mude as linhas 60 - 63

Variáveis

```
global_position(hash)        # Coordenadas X e Y atuais do robô
```

Métodos

callback_pose(odometry) => nil

Descrição:

Alimenta os dados de posição absoluta em x, y e z, e os dados de roll, pitch e yaw.

Ambiente:

Entradas:

1-> scan (sensor_msgs.msg.Odometry):
dados da odometria

Saidas:

append_to_file(msg_str) => nil

Descrição:

Adiciona ao final do arquivo a mensagem enviada por parametro

Ambiente:

Entradas:

1-> msg_str(string):
mensagem a ser adicionanda ao finla do arquivo

Saidas:

save_new_node(node_number, connected_nodes) => nil

Descrição:

Lógica para criar e salvar um novo nó. Realiza a leitura da posição do robô na frequencia de 1hz e obtem os dados que compõem um nó.

Ambiente:

Entradas:

1-> node_number(int):
número do nó a ser salvo
1-> connected_nodes(Array[int]):
nós conectado a este nó

Saidas:

is_in_range((p_x, p_y), p_range) => nil

Descrição:

Verifica se um dado ponto, esta contido no range ao redor do robô

Ambiente:

Entradas:

1-> (p_x, p_y) (tupla):
coordenadas x e y do ponto para verificação
1-> p_range (float):
distância ao redor do robô

Saidas:

calc_middle(arr_points) => nil

Descrição:
Dado um conjunto de pontos, que formam as laterais do nó, calcula o ponto do meio
Ambiente:

Entradas:
1-> arr_points (Array[Array[float]]):
conjunto de pontos, que formam a área do nó.

Saidas:

listener() => nil

Descrição:
Esta função é definida como raiz para o nó de comunicação do ros. Sua função é semelhante a de uma main em c, realizando a captuda de dados enseridos pelo usuário e iniciando a leitura dos dados do grapho.
Ambiente:

Entradas:

Saidas:

REFERÊNCIAS

[1] **Ros - Documentação**
[2] **Python 2.7.15 - Documentação**
[3] **Arquivos de aula**
[4] **Consulta a colegas de sala**
[5] **How to check if a given point lies inside or outside a polygon** - https://www.geeksforgeeks.org/how-to-check-if-a-given-point-lies-inside-a-polygon/