UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE ENGENHARIA DE COMPUTAÇÃO

JOÃO PAULO FERREIRA SOARES

UM AMBIENTE DE SIMULAÇÃO DE MISSÕES *INDOOR* PARA ROBÔS TERRESTRES DE PNEUS DIFERENCIAIS

QUIXADÁ

2020

JOÃO PAULO FERREIRA SOARES

UM AMBIENTE DE SIMULAÇÃO DE MISSÕES *INDOOR* PARA ROBÔS TERRESTRES DE PNEUS DIFERENCIAIS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Orientador: Prof. Dr. Cristiano Bacelar de Oliveira

QUIXADÁ

2020

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S654a Soares, João Paulo Ferreira.
Um ambiente de simulação de missões indoor para robôs terrestres de pneus diferenciais / João Paulo Ferreira Soares. – 2020.
40 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Computação, Quixadá, 2020.
Orientação: Prof. Dr. Cristiano Bacelar de Oliveira.

1. Robôs móveis. 2. Métodos de simulação. 3. Algoritmos. I. Título.

CDD 621.39

JOÃO PAULO FERREIRA SOARES

UM AMBIENTE DE SIMULAÇÃO DE MISSÕES *INDOOR* PARA ROBÔS TERRESTRES DE PNEUS DIFERENCIAIS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Computação.

Aprovada em: _____/_____/_____

BANCA EXAMINADORA

_____

Prof. Dr. Cristiano Bacelar de Oliveira (Orientador)
Universidade Federal do Ceará (UFC)

_____

Prof. Dr. André Ribeiro Braga
Universidade Federal do Ceará (UFC)

_____

Prof. Dr. Paulo Armando Cavalcante Aguilar
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

Aos meus pais, pelo apoio e sustento. Sem eles nada disso seria possível.

Aos meus irmãos, pela cumplicidade e companheirismo.

Ao Prof. Dr. Cristiano Bacelar de Oliveira por me orientar não somente durante esse trabalho, mas ao longo de todo o curso de graduação.

À aluna e amiga Ana Victória Araújo Maia pelo incentivo, paciência e auxílio ao longo desses anos desafiantes. Em especial o desafio de concluir o curso de graduação durante o distanciamento social em meio a pandemia da Covid-19.

À amiga Cláudia Ferreira Alves e o seu dom de revigorar os ânimos e transbordar de esperança mesmo nos momentos mais difíceis.

Ao meu primo, padrinho e amigo Jonas Costa Ferreira da Silva por me inspirar a estudar computação. Também pelo seu amparo, que para mim foi essencial nos primeiros anos de curso.

Por fim, a todos os funcionários da Universidade Federal do Ceará - Campus Quixadá por gerarem a atmosfera maravilhosa que cerca o campus.

RESUMO

Atualmente, existem aplicações de robôs móveis para os mais variados cenários. Em muitos casos, essas aplicações envolvem atividades em ambientes internos que apresentam constantes mudanças na disposição de obstáculos. Para que o robô possa navegar sem se perder ou sofrer danos, é necessário que ele saiba onde ele está e para onde deve ir. Para isso, um ambiente de simulação foi desenvolvido com o foco em robôs terrestres de pneus diferenciais. Dentre as funções implementadas estão a simulação de leituras de distância para observação de obstáculos próximos ao robô, a utilização das leituras simuladas para construção de um Mapa do Local, e a navegação do robô de um ponto a outro do mapa evitando colisões. O ambiente foi implementado de forma modular para permitir a análise de rotas geradas por diferentes algoritmos de *Path Planning*. As análises consideram métricas para avaliar a qualidade das rotas geradas com base no desempenho do robô no ambiente de simulação usado. Por fim, foi apresentada uma comparação dos resultados em cenários com diferentes níveis de complexidade, mostrando quais algoritmos se saíram melhor nos cenários testados.

Palavras-chave: Robôs móveis. Simulação. *Path Planning*. Ambientes internos.

ABSTRACT

Nowdays, there are applications for mobile robots for the most diverse scenarios. In many cases, these applications involve indoor activities, where obstacles constantly change. For the robot to be able to navigate without getting lost or suffering any damage, it needs to knows where it is and where it should go. For this, a simulation environment was developed with a focus on grounded differential-drive robots. Among the implemented features are the simulation of reading from a distance rangefinder sensor to detect obstacles close to the robot. The use of these simulated readings to build a Local Map, and the navigation of the robot from one point to another on the map, avoiding collisions. The system was implemented in a modular way to allow the analysis of routes generated by different Path Planning algorithms. The analyzes consider metrics to evaluate the quality of the generated routes based on the performance of the robot in the navigation task. Finally, a comparison of the results in scenarios with different levels of complexity was presented, showing which algorithms did better in the tested scenarios.

Palavras-chave: Mobile robots. Simulation. Path Planning. Indoor environments.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

FBM	*Feature-Based Map*
GBM	*Grid-Based Map*
LIDAR	*Light Detection And Ranging*
RRT	*Rapidly-exploring Random Tree*
SBP	*Sampling-Based Planning*
SLAM	*Simultaneous Localization and Mapping*

SUMÁRIO

1 INTRODUÇÃO

Hoje em dia, é possível encontrar robôs móveis em vários ambientes distintos. Eles podem se mover no solo, na superfície da água, embaixo d'água ou no ar. Das mais variadas aplicações para robôs móveis, podemos citar os *rovers* de Marte que exploram e coletam informações, os coletores e transportadores de lixo nuclear, detectores de minas terrestres para utilização em guerras, além de robôs utilizados em fábricas para translado de materiais de diversos tipos.

Diferentes aplicações utilizam robôs móveis para realização de atividades que são impraticáveis ao seres humanos por diferentes motivos. Seja por condições extremas de pressão como no fundo dos oceanos, temperaturas extremas como os vulcões ou as geleiras nos polos. Além disso, robôs podem ser utilizados em postos de vigilância ou patrulhamento de perímetros. Em todos os casos, para realizar as atividades a qual são designados evitando que se percam ou se danifiquem, os robôs devem possuir uma boa noção de sua localização e orientação além de rotas bem definidas (COOK, 2011).

Na fase de testes, muitas vezes é difícil reproduzir as condições em que os robôs irão operar. Como, por exemplo, reproduzir em laboratório as características do espaço para testes de satélites. Apesar de serem de difícil reprodução em um ambiente real, essas características, como diversas outras, podem ser implementadas em *software* a fim de avaliar o comportamento do sistema em um ambiente simulado.

Também é possível observar robôs móveis realizando atividades em ambientes internos. Aplicações envolvendo a organização ou translado de materiais pesados na indústria, ou, ainda, robôs autônomos responsáveis pela limpeza de pisos e piscinas. Nestes cenários, é necessário o conhecimento da disposição de obstáculos que impossibilitem ou dificultem a navegação, como paredes, objetos ou pessoas. Esse conhecimento deve ser constantemente atualizado pois, devido à presença humana, os ambientes internos podem apresentar mudanças na organização do local.

Dado o exposto, este trabalho apresenta um ambiente de simulação de missões *indoor* para robôs móveis, com foco em robôs terrestres de pneus diferenciais (Seção 3.1). No contexto deste trabalho, as missões *indoor* são definidas como a realização de uma viagem de um ponto a outro, em um ambiente interno. Além disso, é apresentada uma análise de algoritmos de planejamento de trajetórias. Tais análises consideram métricas para avaliar a qualidade das rotas geradas com base no desempenho do robô no ambiente de simulação usado. Os algoritmos

de *Path Planning* selecionados utilizam o *Sampling-Based Planning* (SBP) (Seção 3.3.1) como estratégia de geração de rota.

O ambiente de simulação proposto inclui as seguintes características:

- Restrições referentes à estratégia de movimento para robôs diferenciais.
- Navegação em rotas geradas por algoritmos do tipo *Sampling-Based Planning*.
- Construção modular, de forma a permitir análises com vários tipos de algoritmos e avaliação do impacto de cada um na execução da missão robô.
- Simulação de sensores de distância com leituras radiais, em ângulos inteiros de 1^{o} a 360^{o}.

Este trabalho está organizado da seguinte forma:

- **Capítulo 2**: apresenta trabalhos com características semelhantes aos utilizados nesse projeto;
- **Capítulo 3**: apresenta uma explicação de tópicos essenciais para esse trabalho;
- **Capítulo 4**: apresenta os passos adotados para realização do sistema proposto;
- **Capítulo 5**: apresenta um comparativo entre métricas para avaliar a qualidade das rotas geradas e o seu impacto no desempenho do robô;
- **Capítulo 6**: Apresenta a conclusão e possíveis trabalhos futuros.

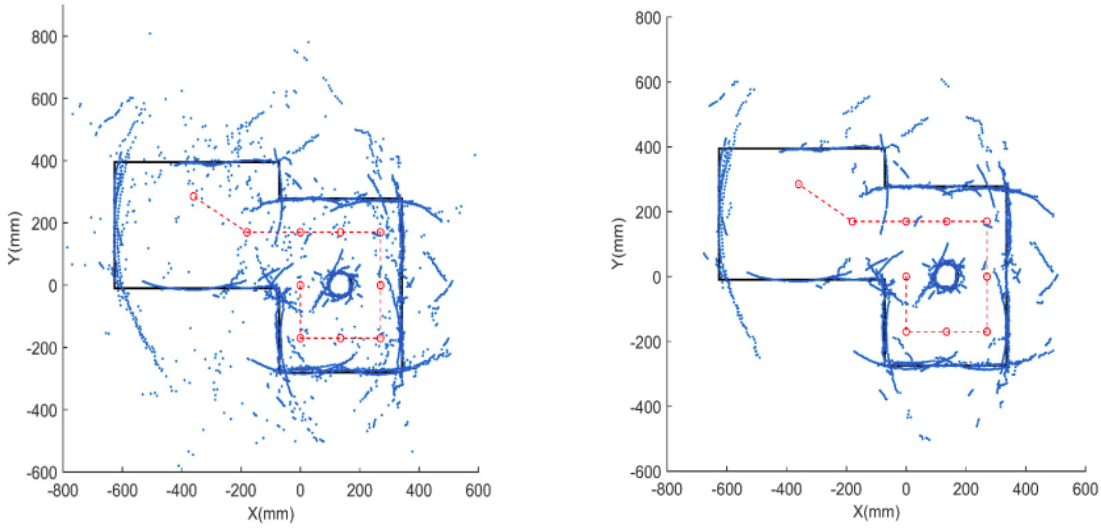
2 TRABALHOS RELACIONADOS

Neste Capítulo serão apresentados trabalhos que apresentam técnicas semelhantes ou que serviram de inspiração para o desenvolvimento desse trabalho. Os trabalhos envolvem mapeamento e extração de informação do ambiente, a construção de robôs terrestres diferenciais aplicados a *Simultaneous Localization and Mapping* (SLAM) e uma comparação entre rotas geradas por algoritmos de *Path planning*.

Para a construção de um mapa baseado na leitura de sensores ultrassônicos, Long *et al.* (2018) desenvolveram uma configuração de 4 sensores ultrassônicos posicionados em formato retangular com 90º de diferença entre eles. Foi desenvolvida uma plataforma que se movia em uma rota predeterminada. Essa plataforma continha um motor responsável por rotacionar uma base. A partir dessas leituras, foi proposto também um modelo de lógica *Fuzzy* para filtrar e classificar objetos. Foram utilizadas 3 categorias com base na forma de sua superfície, são elas: planas, cantos e cilindros. Para os testes, Long *et al.* (2018) utilizaram um conjunto de 10 grupos de leituras do sensor.

As leituras dos sensores podem ser observadas na Figura 1a. É possível observar regiões com considerável consistência e densidade de dados, e ruídos distribuídos por todas as partes. Para reduzir os ruídos, foi aplicado um filtro *K-means*, apresentado em (KANUNGO *et al.*, 2002). O resultado da aplicação do filtro pode ser observado na Figura 1b.

Figura 1 – Estágios do processo de mapeamento de Long *et al.* (2018)



(a) Dados observados pelos sensores

(b) Dados após filtragem usando *K-means*

Fonte: (LONG *et al.*, 2018)

Em Zhu *et al.* (2020) foi proposto um sistema de navegação baseando em SLAM para controle de um robô móvel embarcado. Esse sistema é baseado na fusão de sensores para obter melhores estimativas de localização e utilizar essas estimativas como entrada para um algoritmo SLAM de ponta. O sistema é composto por 2 *encoders* em conjunto com um acelerômetro e um giroscópio. A construção do mapa é feita através das medidas de distâncias obtidas por um sensor *Light Detection And Ranging* (LIDAR) e a localização do robô é estimada por meio de um filtro de partículas. O caminho foi traçado utilizando o DWN[1] que ajustava as rotas após o robô perceber e contornar obstáculos. O direcionamento foi feito através de 2 pneus diferenciais.

Foram realizadas comparações dos mapas gerados utilizando o sistema de navegação proposto e utilizando somente *enconders*. As comparações mostraram melhoras no mapeamento utilizando o sistema de navegação proposto por Zhu *et al.* (2020) principalmente em cenários em que o processo de mapeamento é demorado devido ao tamanho do espaço.

Em (NOREEN *et al.*, 2016) é apresentado um ambiente de simulação proposto para avaliar o *Rapidly-exploring Random Tree* (RRT), *RRT** e RRT*-Smart. Nesse trabalho os autores mostram uma comparação entre algoritmos baseados em amostragem aleatória. Os parâmetros de avaliação e resultados são mostrados na Tabela 1.

Tabela 1 – Resultados obtidos através de simulação

PARÂMETRO	**RRT**	**RRT***	**RRT*-Smart**
Custo de Caminho (m)	56	50	41
Tempo de Computação (s)	150	621	210
Número de nós na Árvore	2954	2972	2000

Fonte: Adaptado de Noreen *et al.* (2016)

Conforme mostram Noreen *et al.* (2016), o RRT apresentou o menor tempo de computação entre os algoritmos analisados, porém gerou o caminho com maior custo. O RRT* encontrou uma rota de menor custo, porém gerou a árvore com maior número de nós e o maior tempo de computação. Já o RRT*-Smart produziu o caminho com menor custo e também a árvore com menor número de nós.

O Quadro 1 mostra um comparativo entre os trabalhos mencionados neste capítulo. Na primeira coluna, podemos observar a forma como o sistema realiza o mapeamento, apresentando os principais sensores utilizados. A segunda coluna mostra a rota encontrada, e quais estratégias de planejamento foram utilizadas. Já na terceira coluna, são mostradas as características do movimento dos sistemas.

[1] Dynamic Window Approach

Quadro 1 – Características gerais dos trabalhos

	Aquisição de Mapa	**Planejamento de Rota**	**Estratégia de Movimento**
Noreen *et al.* (2016)	Fornecido como entrada	Computado usando SBP	–
Long *et al.* (2018)	Ultrassônicos em rotação	Fornecido como entrada	Plataforma para experimento
Zhu *et al.* (2020)	Lidar	Computado com DWA	Robô diferencial
Este projeto	Leituras simuladas	Computado usando SBP	Simulação de robô diferencial

Fonte: Elaborado pelo autor

3 FUNDAMENTAÇÃO TEÓRICA

Neste Capítulo, serão introduzidos os principais conceitos e tópicos relacionados ao trabalho. A Seção 3.1 apresenta alguns tipos de robôs terrestres, a Seção3.2 trata dos problemas de mapeamento e localização. Por fim, a Seção 3.3.1 o SBP é apresentado.

O ambiente proposto nesse trabalho simula a dinâmica de um robô terrestre movido por pneus que, estando em uma região desconhecida, é capaz de construir um mapa e realizar a missão de navegar de um ponto a outro. Para evitar colisões durante o percurso, rotas são computadas utilizando SBP.

3.1 Robôs Terrestres

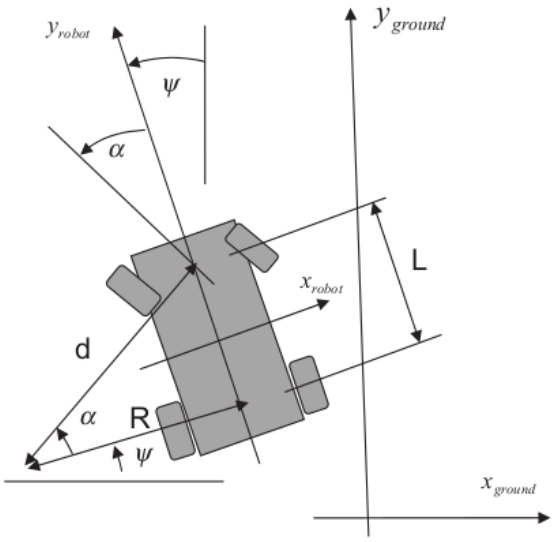
Existem diferentes modelos cinemáticos para robôs terrestres. Nesta seção serão apresentados dois modelos para robôs terrestres que utilizam pneus em contato com o solo para locomoção e orientação do movimento: a) Robôs guiados por pneus dianteiros e b) Robôs guiados por pneus diferenciais. As equações que regem a dinâmica de cada modelo estão disponíveis em (COOK, 2011). Em ambas as abordagens, são usados dois sistemas de coordenadas, sendo um referente ao ambiente e o outro ao robô. As coordenadas do ambiente são denotadas Y_{ground} e X_{ground}, esse sistema é por vezes chamado de coordenadas do mundo. São chamadas assim porque possuem sua origem em um ponto fixo do ambiente. As coordenadas do robô são chamadas de Y_{robot} e X_{robot} e possuem sua origem no centro do carrinho.

Nos robôs guiados por pneus dianteiros (Figura 2), os pneus dianteiros possuem atribuições diferentes dos pneus traseiros. Enquanto os pneus dianteiros são responsáveis pela direção do movimento, os traseiros devem fornecer a tração necessária para o deslocamento.

O ângulo gerado entre o Y_{robot} e o Y_{ground} é denotado ψ. O ângulo formado entre a direção das rodas dianteiras e o Y_{robot} é chamado α. Ambos os ângulos são representados em sentido anti-horário. O centro instantâneo pelo qual o robô esta girando pode ser calculado através da intersecção de duas retas que passam pelos eixos dos pneus dianteiros e traseiros. A reta R passa pelo eixo dos pneus traseiros e L equivale a distancia entre os eixos dianteiros e traseiros do robô (COOK, 2011).

Os robôs guiados por pneus diferenciais (Figura 3) permitem o controle separado entre os pneus do lado direito e esquerdo. Os pneus dianteiros estão fixos ao corpo do carrinho e a direção do robô é controlada através da velocidade dos pneus do lado direito e esquerdo. O ψ

Figura 2 – Robôs direcionados pelos pneus dianteiros

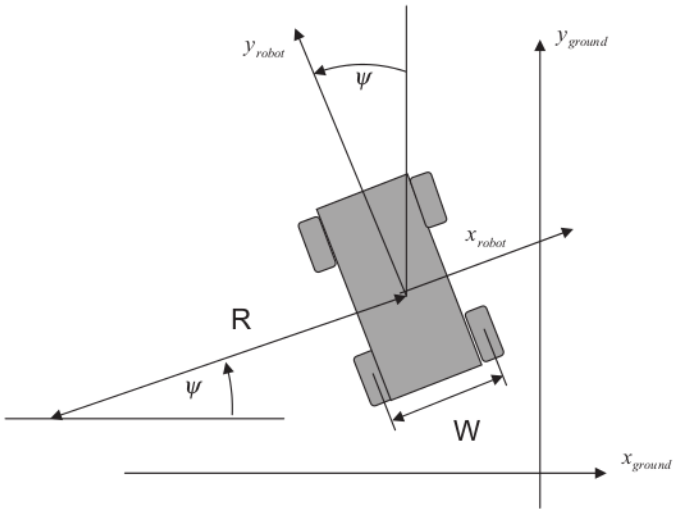


Fonte: (COOK, 2011)

é formado entre o Y_{robot} e o Y_{ground}. A largura do robô é denotada W e raio ao longo da trajetória do robô em determinado instante de tempo é chamado R.

Os robôs guiados por pneus diferenciais são capazes de rotacionar em torno do seu próprio eixo, mudando o ângulo ψ sem alterar sua localização no sistema de coordenadas do mundo. Para gerar uma rotação em seu próprio eixo no sentido antihorario, é necessário rotacionar os pneus do lado direito no sentido antihorário e os pneus do lado esquerdo no sentido horário. Ambos os lados devem apresentar a mesma velocidade de rotação, porém em sentidos contrários.

Figura 3 – Robôs direcionados por pneus diferenciais



Fonte: (COOK, 2011)

Devido a essa característica, é possível guiar um robô diferencial até um destino

qualquer rotacionando o robô em seu próprio eixo direcionando-o para o destino, depois atribuir o mesmo valor de velocidade para ambos os lados do robô e no mesmo sentido, resultando em um movimento em linha reta rumo ao objetivo. Esse estratégia de movimento é chamada *Turn-and-Travel* (COOK, 2011).

Como alternativa ao *Turn-and-Travel*, é possível encontrar valores distintos de velocidade para os pneus do lado direito e esquerdo do robô para que ele siga em movimento circular até o objetivo, esse tipo de movimento é denotado *Turn-while-traveling* (COOK, 2011).

Os robôs móveis podem ser autônomos ou parcialmente autônomos. Os parcialmente autônomos são aqueles que são controlados através de instruções externas, porém permitindo que os detalhes de navegação sejam decididos pelo robô.

Já os robôs autônomos recebem os objetivos e procuram cumprir suas missões sem auxílio externo.

Em ambos os cenários, para que os objetivos sejam cumpridos, o robô deve possuir uma boa noção de sua localização e orientação além de bons comandos de movimentos (COOK, 2011). Os comandos de movimento são instruções fornecidas ao *hardware* do sistema que o fazem se mover, tendo como exemplo os motores de tração de um veículo terrestre movido por pneus. Em um cenário onde o robô deve alcançar uma posição do mapa, os comandos de movimento devem ser gerados a fim de conduzir o robô em uma rota que o leve a posição desejada.

3.2 Mapeamento e Localização

Dado um robô que se encontra em um ambiente completamente desconhecido e não possui qualquer informação sobre a sua localização, o problema de SLAM consiste em construir um mapa do ambiente em que se encontra e simultaneamente localizar-se nele. São dois problemas indissociáveis pois para se localizar é necessário possuir um mapa preciso e, para conseguir um mapa preciso é necessário saber exatamente onde o robô se encontra (STACHNISS, 2009).

Como não dispõe de conhecimento algum do ambiente, o robô necessita de percepções que são obtidas através dos sensores. Essas percepções são limitadas pelo tipo de sensor adotado porém, independente do sensor, sempre haverão incertezas associadas a essas leituras (DURRANT-WHYTE; BAILEY, 2006). Além das incertezas provenientes das observações do meio, existem também incertezas referentes ao movimento do robô. Independente da estratégia

de movimento, sempre existirão fatores de imprecisão que podem estar relacionados a ruídos no comando recebido pelo atuador ou ainda desgaste do sistema motor (THRUN *et al.*, 2006).

Existem diferentes tipos de mapa para modelar o ambiente em sistemas computacionais para aplicações de robótica. São estratégias consolidadas nesse ramo os *Feature-Based Map* (FBM) e os *Grid-Based Map* (GBM) (STACHNISS, 2009).

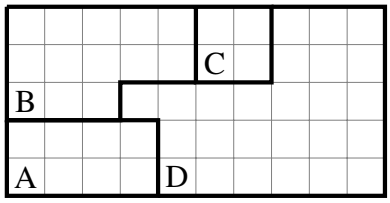
No GBM o ambiente é discretizado em pequenas porções de espaço e cada porção é representado como uma variável independente. Essas variáveis são chamadas células. Cada célula armazena um valor que corresponde à probabilidade de estar ocupada.

Um fator de extrema importância na construção de um GBM é a resolução. Essa resolução define a porção de espaço que é definida por cada célula. Quanto menor o tamanho de célula, maior quantidade de células serão necessárias para discretizar o ambiente e portanto maior resolução. A resolução, por sua vez, impacta na quantidade de memória. Os mapas GBM com grande resolução demandam grande quantidade de memória (STACHNISS, 2009).

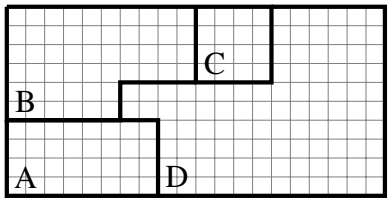
A Figura 4 apresenta o impacto da resolução na construção de um GBM. Assumindo que ambos os mapas discretizam o mesmo ambiente, as possibilidades de representatividade do mapeamento na Figura 4a são consideravelmente inferiores a representatividade do mapeamento na Figura 4b. Por exemplo, considerando o cômodo C, no primeiro caso, somente 4 células foram usadas para discretizar o cômodo, assim um obstáculo detectado, ocupará no mínimo, 1/4 do espaço total do cômodo. Por outro lado, o exemplo da Figura 4b utiliza 16 células para discretizar o mesmo espaço e pode ser mais próximo da realidade. Contudo, supondo que cada célula necessite de 6 bytes de memória, o primeiro mapa exige no 300 bytes enquanto o segundo exige 1.2Mb de memória.

Figura 4 – Possíveis representações de *Grid-Based Map*(GBM) com diferentes tamanhos de células

(a) GBM com célula de 1m



(b) GBM com célula de 0.5m



Fonte: Elaborado pelo autor

Os FBM coletam informações do ambiente e as armazena. Essas informações são chamadas de *features*, e podem ser usadas para diferenciar ambientes (STACHNISS, 2009). Para

cada tipo de sensor utilizado, *features* diferentes podem ser escolhidas para serem coletadas. Para aplicações que utilizam sensores ultrassônicos, as principais informações buscadas são linhas e cantos. Para aplicações que utilizam câmeras, existem *features* específicas que podem ser adotadas.

Esse tipo de mapa apresenta menor custo computacional por não discretizar todo o ambiente em células (DIAZ; ALVARES, 2010). Um exemplo de FBM foi apresentado na Figura 1 em que após realizadas as leituras do sensor, Long *et al.* (2018) efetuaram uma busca por linhas, cantos e cilindros.

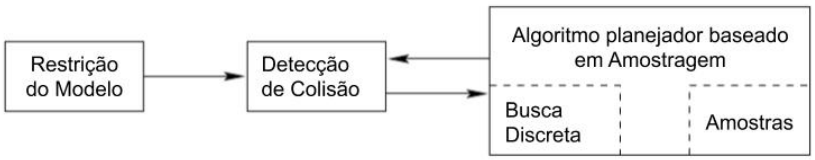
3.3 *Path Planning*

O problema de encontrar rotas é conhecido como *Path Planning* e aplicações de *Path planning* podem envolver carros autônomos, veículos aéreos não-tripulados (VANTs), serviços de vigilância ou ainda missões de exploração espacial (NOREEN *et al.*, 2016).

3.3.1 Sample-based Planning

Esta seção é baseada principalmente em (LAVALLE, 2006). Em robótica, planejamento se refere ao movimento de sistemas dinâmicos que possuem sensores, atuadores e capacidades computacionais. Uma das estratégias utilizadas para planejamento de movimento é o SBP. Uma das ideias principais dessa estratégia é evitar construção explícita de um conjunto de espaços ocupados por obstáculos e um conjunto de regiões livres, a fim de selecionar uma rota que percorra as regiões livres e evite regiões ocupadas. Em vez disso, propõe uma busca em todas as regiões por meio de amostragem aleatória e um módulo de detecção de colisão. O módulo de detecção de colisão é alheio ao planejador, e por essa razão, qualquer restrição de modelo pode ser implementado nele e assim gerar soluções que as satisfaçam.

Figura 5 – Funcionamento geral de um sistema de *Sampling-Based Planning* (SBP)



Fonte: Adaptado de (LAVALLE, 2006)

Como mostra a Figura 5, o módulo de colisão é implementado à parte do planejador.

Isso permite que as soluções geradas pelo algoritmo planejador possam ser implementadas em qualquer modelo dinâmico, contanto que suas restrições sejam consideradas na detecção de colisão.

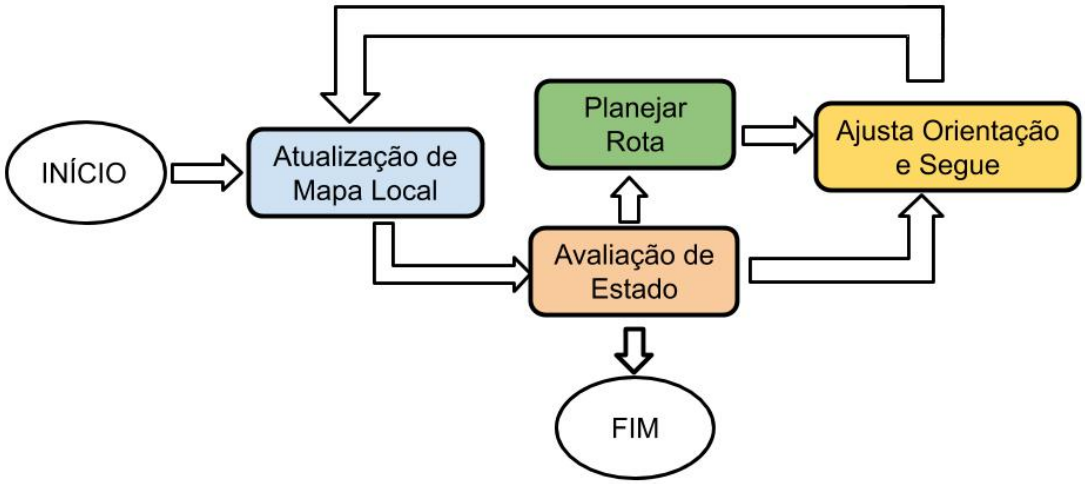
Em geral, os SBP não garantem que uma solução será encontrada contudo, em um espaço de busca finito, as chances de encontrar uma solução aumenta conforme o número de amostras tende ao infinito. Caso não haja uma solução, a execução do algoritmo pode não ter fim a menos que um teto de amostras seja definido.

4 METODOLOGIA

Esta seção apresenta o ambiente desenvolvido como alternativa para a construção de um robô físico. Esse ambiente simula um robô móvel real que se encontra numa região desconhecida e necessita extrair informações que o auxiliem a atingir um objetivo. A posição inicial do robô (Origem) é fornecida ao sistema, bem como o ponto de chegada (Destino).

Durante o processo de navegação do robô, são computadas rotas que, partindo de sua localização atual, o conduzem ao ponto de destino. Diferentes algoritmos foram utilizados para encontrar tais rotas. A Figura 6 apresenta o fluxo de execução do simulador.

Figura 6 – Fluxo geral do ambiente de simulação



Fonte: Elaborado pelo autor

Inicialmente, é necessário fornecer ao sistema o ponto em que o robô se encontra e um mapa completo do local, contendo todos obstáculos. Esse mapa não será utilizado na tomada de decisão do robô, mas servirá de base para as leituras que o robô deve fazer. É necessário também fornecer o ponto de objetivo que deve ser alcançado pelo robô. O ambiente de simulação foi dividido em 4 partes principais descritas pelas regiões coloridas na Figura 6, e são explicadas nas seções a seguir.

4.1 Atualização de Mapa Local

Para navegar pelo ambiente de forma autônoma, um robô móvel necessita extrair informações do local em que se encontra por meio de sensores e usar essas informações para construir um mapa que será utilizado no processo de definição de uma rota. Neste trabalho, esse mapa será chamado de Mapa Local e o seu processo de construção será simulado utilizando

processamento de imagem. As simulações não consideram as incertezas referentes aos sensores e atuadores.

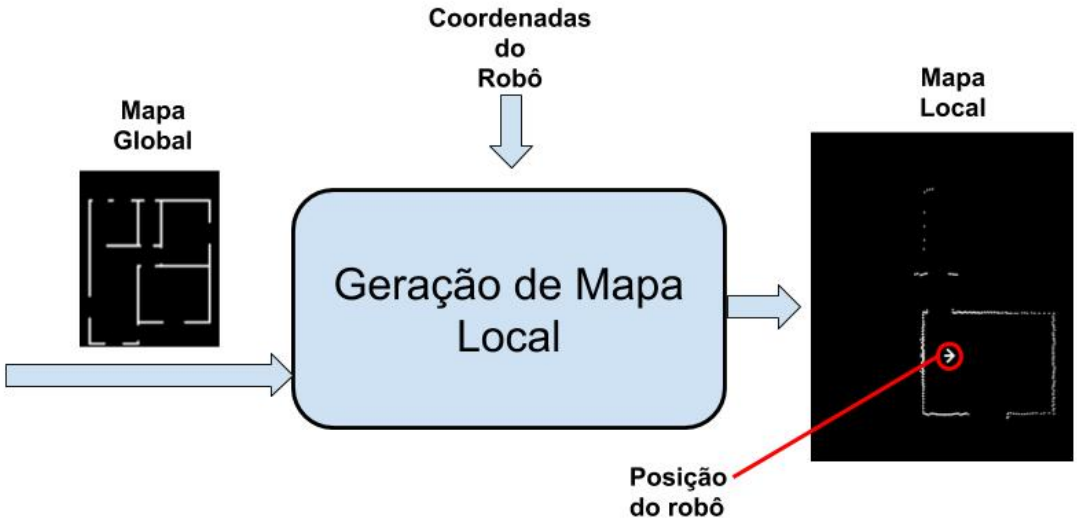
Na primeira iteração do processo de atualização do Mapa Local, esse mapa é gerado a partir de um mapa vazio. O sistema volta à etapa de Atualização de Mapa Local após execução de um comando de movimento, como mostrado na Figura 6. Nas demais iterações o Mapa Local é atualizado a partir das informações acumuladas das iterações anteriores.

Para gerar o Mapa Local, contendo informações visíveis ao robô, é necessário primeiro fornecer ao sistema uma imagem binária contendo todas as áreas livres e ocupadas. Essa imagem será chamada de Mapa Global e, a partir dela, um GBM é gerado em forma de imagem binária, onde cada pixel representa uma célula que pode assumir os valores 0 ou 1. As células com valor 0 representam as regiões livres enquanto as células de valor 1 representam os obstáculos.

Em aplicações reais, é comum que os sensores embutidos nos robôs percebam somente as informações que estão mais próximas a eles, não sendo capazes de observar obstáculos oclusos por paredes e/ou objetos. Por esse motivo a simulação de leituras também fornece ao robô apenas as informações que estão próximas e possíveis de observar.

A Figura 7 apresenta as entradas usadas para gerar o Mapa Local. O Mapa Global, a esquerda, é de onde são extraídas as leituras, a partir das coordenadas do robô e o Mapa Local obtido é apresentado a direita. A posição do robô marcada em vermelho é meramente ilustrativa. Percebe-se que em regiões mais próximas a qualidade dos obstáculos percebidos é maior.

Figura 7 – Geração de Mapa Local

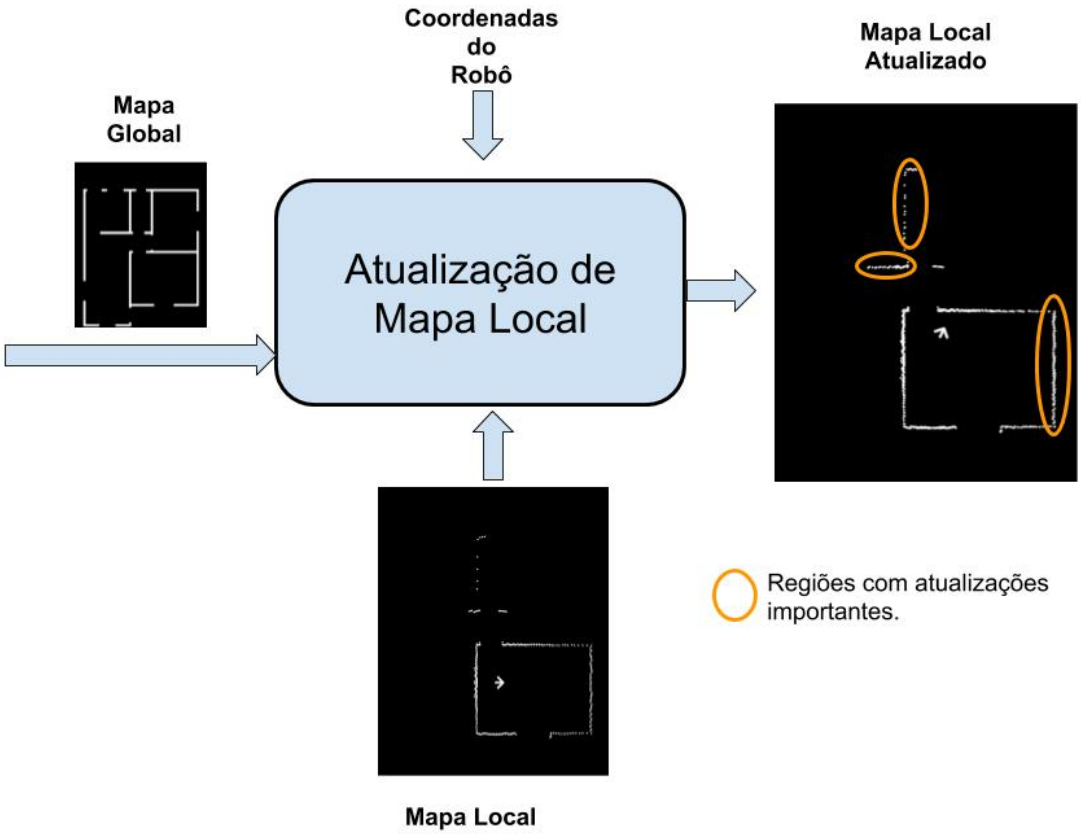


Fonte: Elaborado pelo autor

Dada a localização do robô no Mapa Global, é computada uma distância ρ e um ângulo θ entre cada ponto discretizado no mapa e o robô. Apenas ângulos inteiros são considerados, num intervalo que varia entre 1^o e 360^o. Esta abordagem representa um sistema onde o sensor é instalado de forma a permitir leituras em várias direções no plano, a exemplo do que ocorre em Long *et al.* (2018). Se para um determinado ângulo θ existirem múltiplos pontos no Mapa Global, somente o ponto de menor distância é considerado no Mapa Local. Por fim, os pontos que restam são desenhados no Mapa Local em torno da localização do robô.

Uma vez gerado o Mapa Local, o processo de atualização considera as informações já obtidas e adiciona novas leitura ao Mapa Local. Na Figura 8, o Mapa Local é entrada para o processo de atualização. É possível perceber ainda que, devido ao movimento realizado pelo robô e à atualização de sua localização, informações importantes foram adquiridas. Essas informações estão destacadas em amarelo na Figura 8.

Figura 8 – Atualização de Mapa Local



Fonte: Elaborado pelo autor

4.2 Planejamento de Rotas

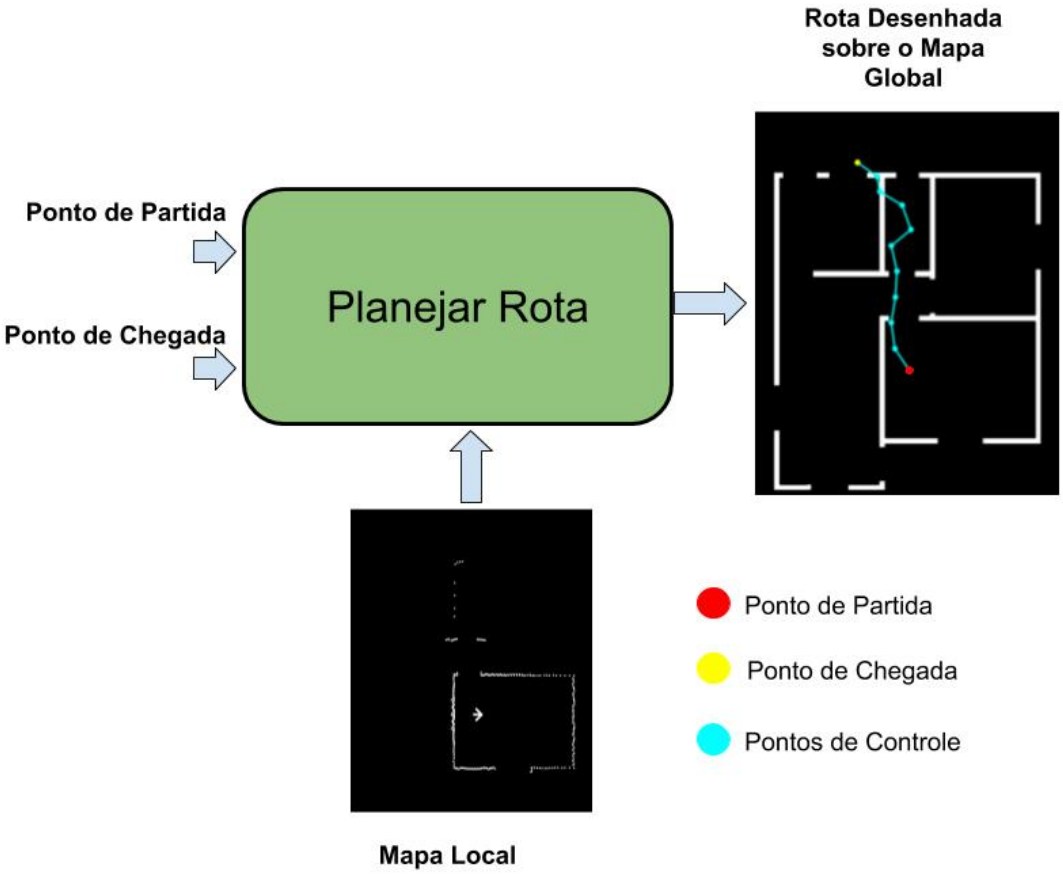
Neste trabalho, uma rota consiste em uma sequência de pontos que iniciam no ponto de origem e terminam no ponto de destino. Esses pontos serão chamados Pontos de Controle, e são gerados de forma aleatória, possuindo limiar de distância máxima entre eles.

A etapa de planejamento de rota, utiliza o Mapa Local e os pontos de origem e destino para realizar o cálculo da rota. Diferentes algoritmos foram utilizados para o cálculo da rota e serão detalhados na seção 4.2.1. A Figura 9 mostra o resultado de uma execução desta etapa.

Como podemos observar, a saída do processo é composta por uma sequência de pontos, que, para fins de visualização, foram desenhados sobre o Mapa Global. Nas extremidades, temos a origem (Ponto de Partida) em vermelho e o destino (Ponto de Chegada) em amarelo. Ambos são fornecidos como entrada do processo.

Entre a os pontos de Partida e Chegada estão os Pontos de Controle em ciano gerados pelo algoritmo de cálculo de rota.

Figura 9 – Fluxo da etapa Avaliação de Estado



Fonte: Elaborado pelo autor

Ainda sobre o exemplo da Figura 9, o Mapa Local utilizado para cálculo de rota possui informações limitadas de obstáculos distantes de sua localização atual, neste caso, o Ponto de Partida. Como resultado dessa baixa representatividade do Mapa Local, é possível observar que próximo ao ponto de chegada, os Pontos de Controle estão atravessando paredes.

Conforme o avanço do robô nos pontos inciais da rota, a atualização do Mapa Local deve tornar mais visíveis os obstáculos que impedem o seguimento dessa rota até o ponto de Chegada e, eventualmente, uma nova rota será calculada.

Neste trabalho foram utilizados diferentes algoritmos SBP para o cálculo de rotas, sendo eles: RRT, RRT*, RRT-Connect e RRT-Dynamic.

4.2.1 RRT

O RRT é um algoritmo amplamente utilizado em robótica pela sua simplicidade e capacidade de resolver problemas que apresentem muitas variáveis (LAVALLE, 1998). O Algoritmo 1 mostra o funcionamento do RRT.

Algoritmo 1: RRT

Início: T$= (V,E) \leftarrow RRT(Z_{inicio})$

1: T $\leftarrow InicializaArvore()$;
2: T $\leftarrow InsereNo(\emptyset, Z_{inicial}, T)$;
3: **for** $i = 0$ ate $i = N$ **do**
4: $\quad Z_{aleatorio} \leftarrow Amostra(i)$;
5: $\quad Z_{maisProximo} \leftarrow MaisProximo(T, Z_{aleatorio})$;
6: $\quad (Z_{novo}, U_{novo}) \leftarrow Guia(Z_{maisProximo}, Z_{aleatorio})$;
7: $\quad$ **if** $SemColisoes(Z_{novo})$ **then**
8: $\quad\quad T \leftarrow InsereNo(Z_{min}, Z_{novo}, T)$;
9: $\quad$ **end if**
10: **end for**
11: **return** T

O RRT cria uma árvore com raiz no ponto de Partida que cresce usando amostragem aleatória no espaço. A medida que novas amostras são geradas a árvore se expande. Se uma amostra aleatória $Z_{aleatorio}$ for disposta em uma região livre de obstáculos, então é selecionado o nó pertencente a árvore que mais se aproxima de $Z_{aleatorio}$ chamado $Z_{maisProximo}$. Se a distância entre $Z_{aleatorio}$ e $Z_{maisProximo}$ for menor que o tamanho máximo λ e não houver obstáculos entre eles, então $Z_{aleatorio}$ é conectado a árvore através de $Z_{maisProximo}$. Caso a distância entre $Z_{aleatorio}$ e $Z_{maisProximo}$ seja maior que λ então um novo nó é gerado Z_{novo} entre $Z_{aleatorio}$ e $Z_{maisProximo}$.

Se o caminho entre Z_{novo} e $Z_{maisProximo}$ for livre de obstáculos, Z_{novo} é conectado a árvore por meio de $Z_{maisProximo}$. Se a amostra aleatória $Z_{aleatorio}$ for disposta sobre um obstáculo, então ela é descartada.

A função *Amostra()*, na linha 4, retorna uma nova amostra aleatória no espaço. Dado uma árvore e uma amostra, a função *MaisProximo()*, na linha 5, retorna o nó pertencente a árvore que mais se aproxima da amostra.

A função *Guia()*, na linha 6, recebe um nó e uma amostra, caso a distância entre a amostra e o nó seja maior que o valor limite λ, um novo nó que respeite a restrição de distância é gerado e retornado. Caso essa distância seja menor que λ então o novo nó é gerado na posição da amostra. A função *Guia()* retorna ainda uma instrução de movimento que pode ser utilizado para atingir o nó gerado partindo do nó recebido.

O retorno da função *SemColisoes()*, na linha 7, é um valor de verdadeiro se o caminho entre o nó passado e a árvore estiver livre de obstáculos. Caso contrário, *SemColisoes()* retornará falso.

Na linha 10, *InsereNo()* adiciona um novo nó á árvore conectando-o ao nó pai.

4.2.2 RRT*

Baseado no RRT, o RRT* se propõe a aprimorar o caminho resposta através de duas novas funcionalidades chamadas de *seleciona melhor pai* e *religamento*. O Algoritmo 2 apresenta em detalhes o funcionamento do RRT*.

Algoritmo 2: RRT*

Inicio: T$= (V,E) \leftarrow RRT^*(Z_{inicial})$

1: T $\leftarrow InicializaArvore()$;
2: T $\leftarrow InsereNo(\emptyset, Z_{inicial}, T)$;
3: **for** $i = 0$ ate $= N$ **do**
4: $\quad Z_{aleatorio} \leftarrow Amostra(i)$;
5: $\quad Z_{maisProximo} \leftarrow MaisProximo(T, Z_{aleatorio})$;
6: $\quad (Z_{novo}, U_{novo}) \leftarrow Guia(Z_{maisProximo}, Z_{aleatorio})$;
7: $\quad$ **if** $SemColisoes(Z_{novo})$ **then**
8: $\quad\quad Z_{proximo} \leftarrow Proximo(T, Z_{novo}, |V|)$
9: $\quad\quad Z_{min} \leftarrow EscolhePai(Z_{proximo}, Z_{maisProximo}, Z_{novo})$
10: $\quad\quad T \leftarrow InsereNo(Z_{min}, Z_{novo}, T)$;
11: $\quad\quad T \leftarrow Religa(T, Z_{proximo}, Z_{min}, Z_{novo})$;
12: $\quad$ **end if**
13: **end for**
14: **return** T

A função *Proximo()*, na linha 8, recebe a árvore, o novo nó gerado por *Guia()*, na linha 6, e o conjunto com todos os nós e retorna o subconjunto de nós mais próximos de Z_{novo}.

Na linha 9, *EscolhePai()* retorna o melhor pai para Z_{novo} entre todos os que foram retornados por *Proximo()*.

Uma vez atualizada a árvore, na linha 10, por meio da função *InsereNo()*, a função *Religa()*, na linha 11, avalia os nós próximos a Z_{novo} e verifica se o custo de liga-los a Z_{novo} é menor que o custo dos seus respectivos pais, se sim, o religamento é feito através de Z_{novo}.

As demais funções do RRT* são idênticas as do RRT.

4.2.3 RRT-Connect

Também baseado no RRT, o RRT-*Connect* foi proposto utilizando uma heurística de conexão para gerar trajetórias mais longas em uma mesma direção. Além disso, o RRT-*Connect* utiliza uma árvore com raiz no ponto de Partida e outra árvore com raiz no ponto de Chegada. Assim, quando as árvores se encontram é possível extrair um caminho. O Algoritmo 3 apresenta o funcionamento do RRT-*Connect*.

Algoritmo 3: RRT-Connect

Inicio: $T_a = (V,E) \leftarrow RRT^*(Z_{inicial})$; $T_b = (V,E) \leftarrow RRT^*(Z_{final})$

1: **for** $i = 0$ ate $= N$ **do**
2: $Z_{aleatorio} \leftarrow Amostra(i)$;
3: **if not** $Expansao(T_a, Z_{aleatorio})$ = "Colisao" **then**
4: **if** $Conecta(T_b, Znovo)$ = "Alcancado" **then**
5: **return** Caminho(T_a, T_b);
6: **end if**
7: $SWAP(T_a, T_b)$;
8: **end if**
9: **end for**
10: **return** Falha

A árvore T_a é iniciada no ponto de Partida, enquanto T_b tem inicio no ponto de Chegada.

A função *Expansao()*, na linha 3, é similar a função *Guia()* dos Algoritmos 1 e 2. Caso a distância entre $Z_{aleatorio}$ e $Z_{maisProximo}$ seja menor que λ, *Expansao()*, na linha 3, adiciona $Z_{aleatrio}$ à árvore e retorna "Alcancado". Caso essa distância seja maior que λ, então *Expansao()* gera um novo nó que satisfaça a restrição λ, adiciona na árvore e retorna "Avancado". Se existir algum obstáculo que impeça *Expansao()* de conectar o nó na árvore, então ela retornará

"Colisao".

Uma vez que um novo nó Z_{novo} for adicionado na árvore T_a por meio de *Expansao()*, uma tentativa de conexão ligando a árvore T_b a Z_{novo} é iniciada por meio de *Conecta()* na linha 4 do Algoritmo 3. A função *Conecta()* pode ser observado no Algoritmo 4.

Enquanto o retorno de *Expansao()*, na linha 2 do Algoritmo 4, for diferente de "Avancado", um novo nó é gerado em *Expansao()* e conectado a árvore T, diminuindo assim um caminho em linha reta entre T e o nó q.

Algoritmo 4: Conecta(T, q)

```
1: repeat
2:     S ← Expansao(T, q);
3: until not (S = "Avancado")
4: return S;
```

Se ao final do *loop*, a função *Conecta()* retornar "Alcancado", então o caminho ligando T_a e T_b passando por Z_{novo} é retornado. Caso contrário uma troca é feita entre T_a e T_b e o processo é retomado da geração de um novo nó Z_{novo}, na linha 2 do Algoritmo 3.

4.2.4 RRT-Dynamic

Para planejar caminhos em ambientes com obstáculos móveis e fixos, Seif Roudabe andOskoei (2015) propôs um fluxograma apresentado na Figura 10. A rota resultante é gerada utilizando o RRT*.

No Dynamic-RRT, os obstáculos móveis e estáticos são identificados. Em seguida, a posição dos obstáculos móveis é estimada em função do tempo. As estimativas de obstáculos são consideradas para definir se uma rota é válida ou não.

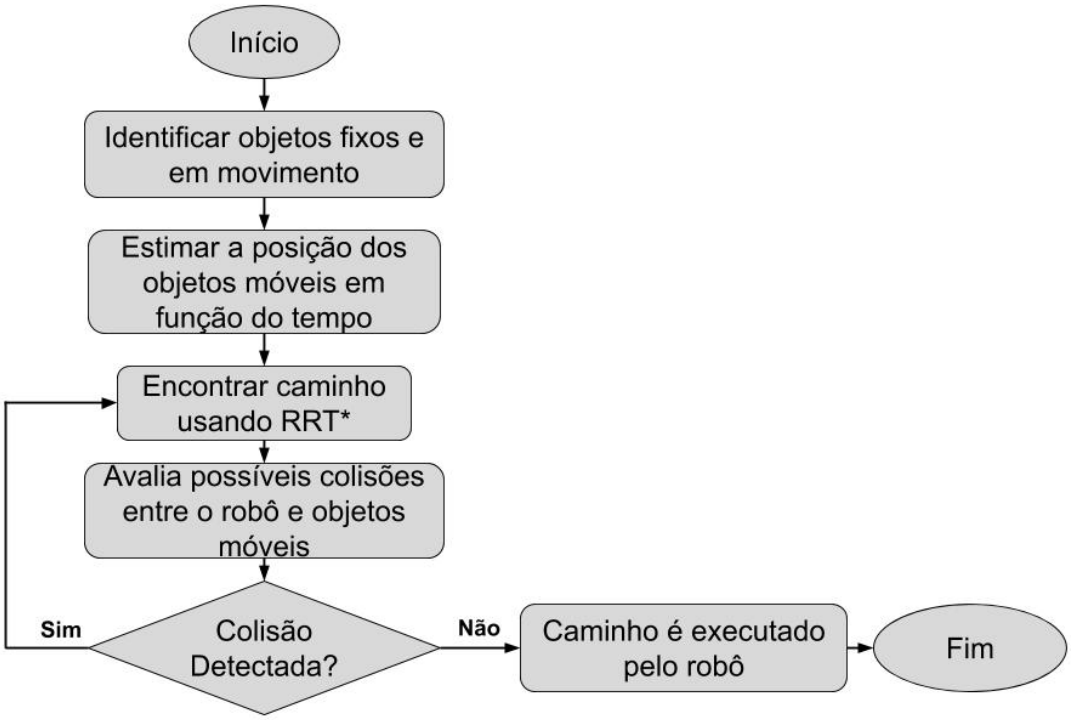
4.3 Avaliação de Estado

Uma vez gerado o Mapa Local, diferentes possibilidades de ações são possíveis a depender do Estado do robô. Essas decisões são tomadas no etapa de Avaliação de Estado.

A Figura 11 mostra o fluxo de decisões da etapa. É realizado um teste para verificar se o robô atingiu a localização objetivo, caso sim a simulação é encerrada.

Na primeira iteração, como não existe caminho até o objetivo, o sistema é conduzido para Planejar Rota. Nas iterações subsequentes, enquanto o objetivo não for alcançado e não

Figura 10 – Dynamic RRT



Fonte: Elaborado pelo autor adaptado de (SEIF ROUDABE ANDOSKOEI, 2015).

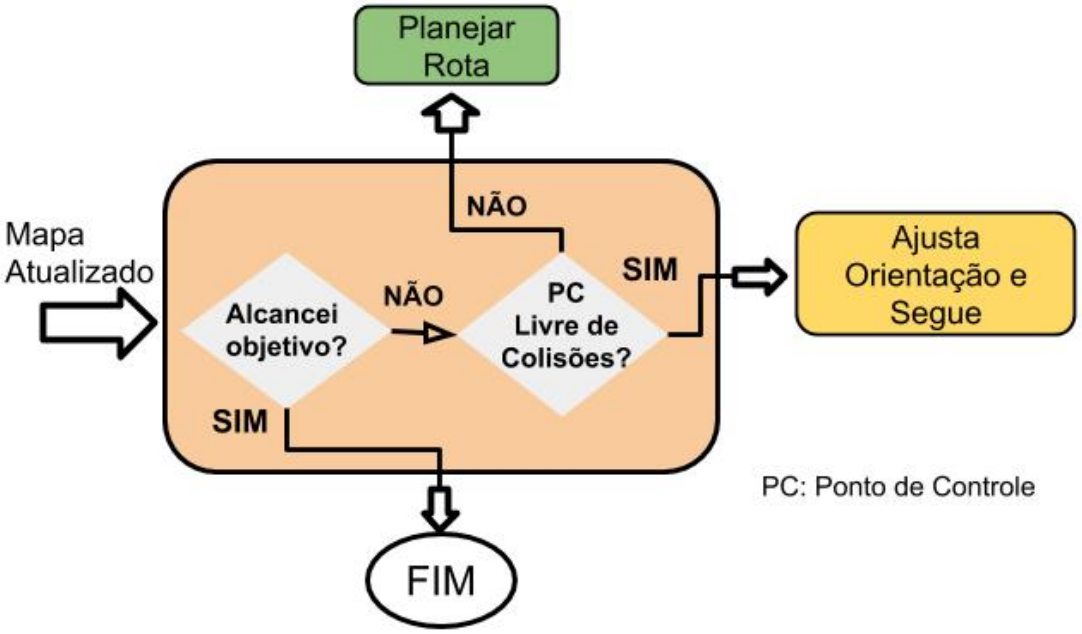
houver colisão que impeça o robô de alcançar o próximo Ponto de Controle, o robô continua a seguir o caminho planejado. Em caso de obstáculo entre a localização do robô e o próximo Ponto de Controle, a rota é atualizada.

4.4 Dinâmica do robô

Os testes foram realizados utilizando o modelo robôs com rodas diferenciais e a estratégia *Turn-and-travel*, mostrados em 3.1. Para fins de simulação, a velocidade do robô foi considerada constante, podendo ser no sentido horário ou anti-horário, e os efeitos da aceleração foram desprezados.

Em todos os cenários de simulação, o robô é criado no ponto inicial fornecido e com angulação $\psi = 0$. Após a construção do primeiro do Mapa Local e definição de rota até o objetivo, o robô deve ir até o próximo ponto de Controle. Para isso, primeiro passo deve ser ajustar o ψ na direção desejada, em seguida seguir em linha reta até o ponto. O caminho percorrido pelo robô é armazenado começando do ponto de Partida e cada ponto de Controle alcançado é adicionado ao caminho percorrido.

Figura 11 – Fluxo da etapa Avaliação de Estado



Fonte: Adaptado de Seif Roudabe andOskoei (2015)

5 RESULTADOS

Para realização deste trabalho, os algoritmos apresentados na seção 4.2.1 foram adaptados[1] para serem utilizados com o simulador proposto e seus desempenhos analisados através de quatro variáveis, sendo elas:

- Planejamento;
- Distância percorrida;
- Pontos de Controle; e
- Tempo de Computação.

,

A variável *Planejamento* representa o número de vezes que o algoritmo gerou um novo caminho. Uma vez alcançada a meta, a variável *Pontos de Controle* armazena o número de nós que o robô percorreu até atingir o objetivo. A distância máxima entre dois nós é 50 *pixels*. A *Distância Percorrida* é o somatório de todas as distâncias entre os Pontos de Controle do caminho, normalizada pela distância mínima (em linha reta) entre os pontos de Partida e de Chegada. Por fim, o tempo que o algoritmo utilizou para encontrar as rotas é salvo em *Tempo de Computação*. Os testes foram aplicados em dois Cenários A e B.

5.1 Cenários de Teste e Parâmetros de execução

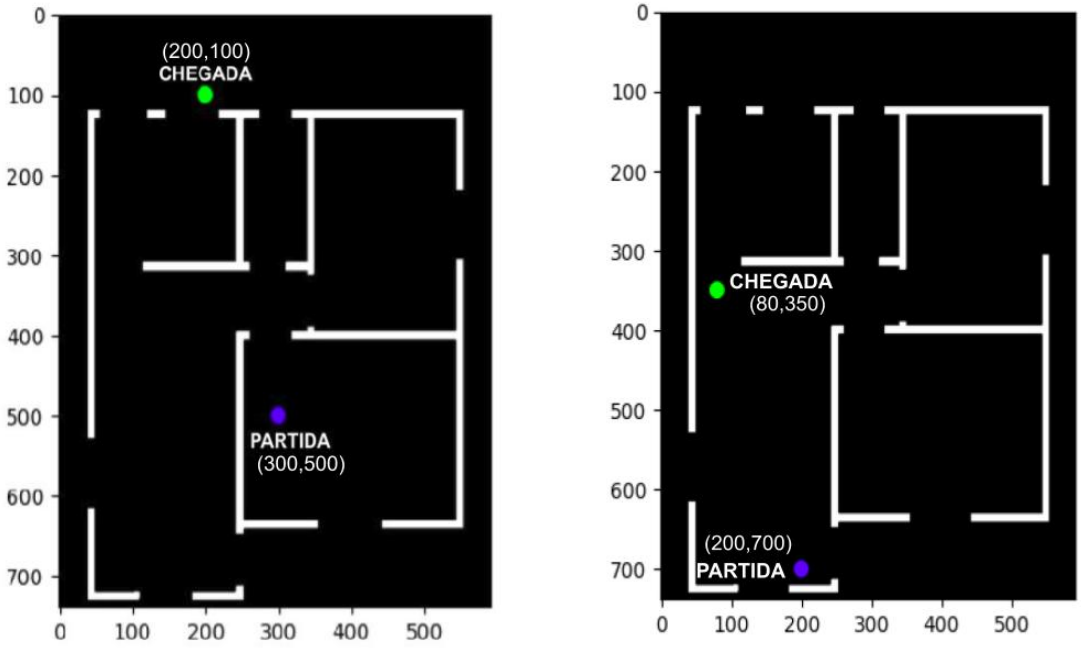
Para avaliar o desempenho dos algoritmos, dois cenários diferentes foram escolhidos para testes. A Figura 12 mostra o Ponto de Partida e o Ponto de Chegada para cada cenário.

O Cenário A foi pensado para ser mais complexo e analisar o comportamento dos algoritmos em um ambiente com muitos obstáculos oclusos entre os pontos iniciais e finais. O Cenário A é mostrado na figura 12a. Já no Cenário B, na Figura 12b, os pontos de Partida e Chegada se encontram no mesmo cômodo e nenhum obstáculo impede os algoritmos de traçarem rotas em linha reta.

Um mesmo algoritmo foi executado 100 vezes em cada cenário. Ao termino de cada execução, a árvore gerada pela amostragem aleatória foi descartada e uma nova árvore iniciada.

[1] Código-fonte original disponível em https://github.com/zhm-real/PathPlanning

Figura 12 – Exemplos de cenários de testes



(a) Cenário A

(b) Cenário B

Fonte: Elaborado pelo autor

5.2 Resultados Cenário A

A Tabela 2 apresenta as médias obtidas pelas 100 execuções de missão para cada algoritmo. O RRT e o RRT* apresentam médias semelhantes em todas as variáveis analisadas, com o RRT obtendo o menor número de planejamentos e também de Pontos de Controle do caminho gerado. O RRT* encontrou em média os caminhos mais curtos porém gastou mais tempo de computação.

Tabela 2 – Média para Cenário A

ALGORITMO	Planejamento	Pontos de controle	Distância Percorrida	Tempo de Computação (s)
RRT	2,94	19,03	2,31	0,1631
RRT*	3,01	19,19	2,18	2,6609
RRT-Dynamic	15,03	125,97	14,36	1,5089
RRT-Connect	42,37	124,08	10,04	0,2539

Fonte: Elaborado pelo autor

O RRT-Dynamic junto com o RRT-Connect apresentaram valores consideravelmente elevados se comparados ao RRT e RRT*. O RRT-Connect teve média 42,37 planejamento de rotas e o RRT-Dynamic gerou caminhos com 125 pontos de controle em média.

5.3 Resultados Cenário B

Em um cenário mais simples, como no Cenário B, os algoritmos apresentam resultados semelhantes para as variáveis analisadas. A Tabela 3 mostra as variáveis para esse cenário.

Tabela 3 – Média para Cenário B

ALGORITMO	Planejamento	Pontos de controle	Distância Percorrida	Tempo de Computação (s)
RRT	1,31	9,39	1,31	0,0626
RRT*	1,23	10,66	1,33	1,4156
RRT-Dynamic	1,30	10,94	1,51	0,0710
RRT-Connect	1,0	9,24	1,05	0,0099

Fonte: Elaborado pelo autor

Nesse cenário mais simples, o RRT-Connect apresentou as médias mais baixas para todas as variáveis. Em todas as 100 execuções, o algoritmo encontrou um caminho válido na primeira execução. O tempo de computação do RRT-Connect também foi consideravelmente menor que os demais. O RRT* também obteve o maior média de tempo de computação e o RRT apresentou caminhos curtos e com poucos pontos de controle em média.

5.4 Comparativo entre Cenários

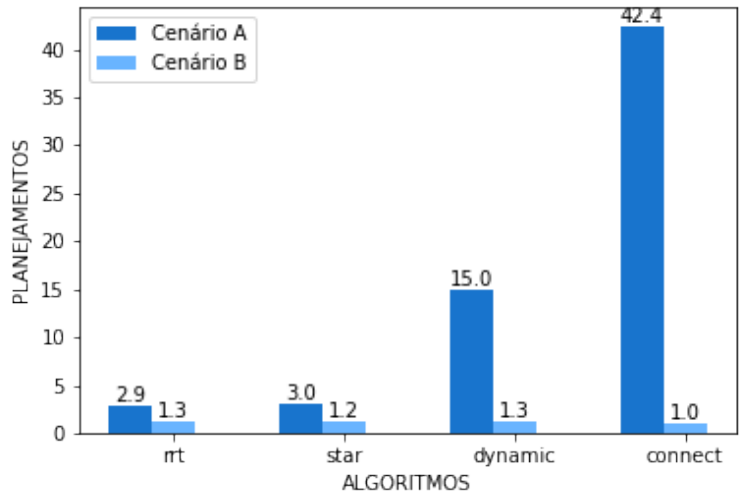
Nesta seção, é realizada uma análise do impacto da mudança de cenário nos valores obtidos pelos algoritmos. No eixo horizontal dos gráficos são mostrados os algoritmos de planejamento e no eixo vertical as médias para cada variável.

O gráfico de barras da figura 13 mostra a variável Planejamento. A variável Planejamento apresentou comportamento semelhante em ambos os cenários para o RRT e RRT*. O RRT-Dynamic não se saiu bem no Cenário A porém no Cenário B houve uma melhora considerável. O RRT-Connect apesar de obter média de planejamento igual a 1 no Cenário B, se mostrou instável com a mudança de cenário.

O comportamento da variável Planejamento pode ser observado também para as variáveis Pontos de Controle e Distância Percorrida mostradas nas figuras 14 e 15 respectivamente. Nelas observamos que o RRT e RRT* apresentam uma redução nos valores obtidos no Cenário B em relação ao Cenário A. Essa redução era esperada pois para realizar a missão no Cenário B, o robô não necessita muitas atualizações de mapa e também não há obstáculos que impeçam a rota de ser traçada em linha reta.

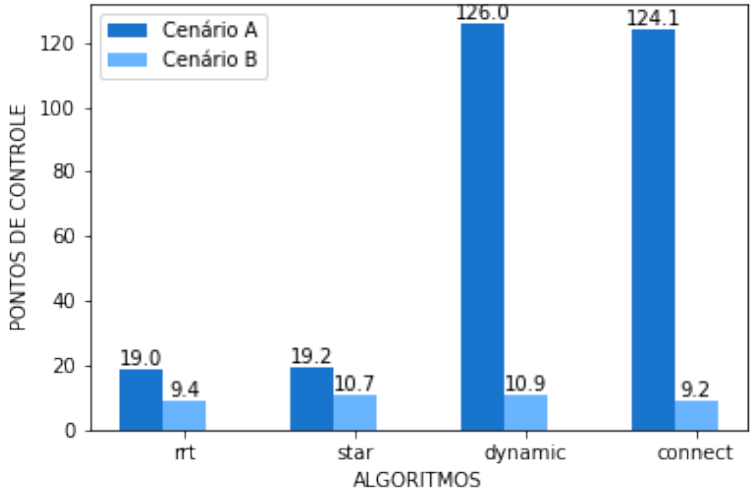
Contudo, ao analisar o RRT-Connect e RRT-Dynamic percebemos uma redução

Figura 13 – Comparação dos algoritmos por cenário em relação a métrica *Planejamento*



Fonte: Elaborado pelo autor

Figura 14 – Comparação dos algoritmos por cenário em relação a métrica *Ponto de Controle*

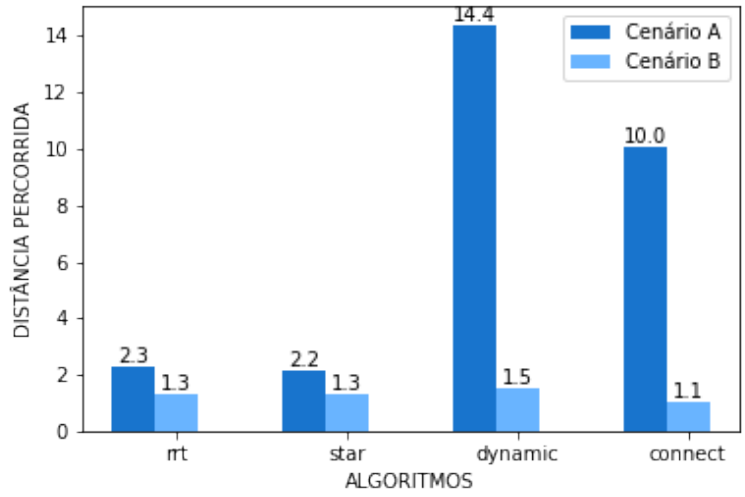


Fonte: Elaborado pelo autor

drástica nos valores do Cenário B em comparação ao Cenário A. Ambos os algoritmos tiveram um desempenho inferior no Cenário A apresentando as maiores médias para as variáveis Planejamento, Pontos de Controle e Distância Percorrida. No Cenário B os valores foram baixos, semelhantes aos obtidos por RRT e RRT*. A heurística de conexão utilizada pelo RRT-Connect se provou eficiente para o Cenário B, uma vez que o RRT-Connect gerou as médias mais baixas para todas as variáveis.

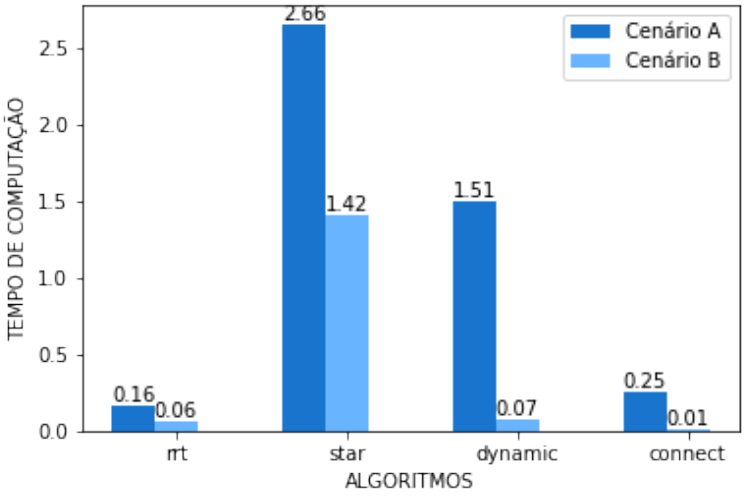
A figura 16 apresenta a variável Tempo de Computação. Nela podemos observar que o RRT* obteve média elevada para ambos os cenários, seguida pelo RRT-Dynamic que apresentou valores maiores que o RRT e RRT-Connect em ambos os cenários.

Figura 15 – Comparação dos algoritmos por cenário em relação a métrica *Distância Percorrida*



Fonte: Elaborado pelo autor

Figura 16 – Comparação dos algoritmos por cenário em relação a métrica *Tempo de Computação*



Fonte: Elaborado pelo autor

6 CONCLUSÃO

Existem inúmeras aplicações possíveis para robôs móveis e é possível notar a presença deles nos mais variados cenários. Porém, para que eles executem bem as funções para as quais foram pensados, necessitam de informações de rota bem definidas, de forma a serem conduzidos por regiões seguras, respeitadas as suas restrições e características físicas.

Neste trabalho foi apresentado um ambiente de simulação onde foram analisados os impactos de diferentes algoritmos de geração de rotas para missões de robôs terrestres de pneus diferenciais. Dentre as ações simuladas estão a construção e atualização de um mapa, a tomada de decisão para evitar colisões com obstáculos ou ainda computar uma nova rota utilizando o mapa atualizado.

Esse tipo de simulação pode ser útil para realização de testes em ambientes que implementem as características que serão encontradas pelo robô em sua aplicação, sendo possível analisar o seu comportamento ainda em laboratório corrigindo e aprimorando o sistema a fim de evitar que o robô se perca ou seja danificado quando utilizado em uma situação real.

Por fim, foi apresentada uma comparação dos resultados em cenários com diferentes níveis de complexidade, mostrando quais algoritmos se saíram melhor nos cenários testados.

Trabalhos futuros podem incluir a expansão do ambiente de simulação para avaliar outros tipos de robôs móveis, como os veículos aéreos não-tripulados (VANTs), robôs aquáticos ou subaquáticos. Ou ainda realizar testes utilizando outras estratégias de movimento para robôs terrestres, como os mencionados na seção 3.1, ou, ainda, outros algoritmos de *Path Planning*.

REFERÊNCIAS

COOK, G. **Mobile robots:** navigation, control and remote sensing. [S.l.]: John Wiley & Sons, 2011.

DIAZ, C. P. O.; ALVARES, A. J. A 2d-mapping strategy based on line segment extraction. In: IEEE. **2010 IEEE ANDESCON**. [S.l.], 2010. p. 1–6.

DURRANT-WHYTE, H.; BAILEY, T. Simultaneous localization and mapping: part i. **IEEE robotics & automation magazine**, IEEE, v. 13, n. 2, p. 99–110, 2006.

KANUNGO, T.; MOUNT, D. M.; NETANYAHU, N. S.; PIATKO, C. D.; SILVERMAN, R.; WU, A. Y. An efficient k-means clustering algorithm: Analysis and implementation. **IEEE Transactions on Pattern Analysis & Machine Intelligence**, IEEE, n. 7, p. 881–892, 2002.

LAVALLE, S. M. Rapidly-exploring random trees: A new tool for path planning. **Citeseer**, 1998.

LAVALLE, S. M. **Planning algorithms**. [S.l.]: Cambridge university press, 2006.

LONG, Z.; HE, R.; HE, Y.; CHEN, H.; LI, Z. Feature extraction and mapping construction for mobile robot via ultrasonic mdp and fuzzy model. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 18, n. 11, p. 3673, 2018.

NOREEN, I.; KHAN, A.; HABIB, Z. A comparison of rrt, rrt* and rrt*-smart path planning algorithms. **International Journal of Computer Science and Network Security (IJCSNS)**, International Journal of Computer Science and Network Security, v. 16, n. 10, p. 20, 2016.

SEIF ROUDABE ANDOSKOEI, M. A. Mobile robot path planning by rrt* in dynamic environments. **International journal of intelligent systems and applications**, Modern Education and Computer Science Press, v. 7, n. 5, p. 24, 2015.

STACHNISS, C. **Robotic mapping and exploration**. [S.l.]: Springer, 2009. v. 55.

THRUN, S.; BURGARD, W.; FOX, D. Probabilistic robotics (intelligent robotics and autonomous agents series). **Intelligent robotics and autonomous agents, The MIT Press (August 2005)**, 2006.

ZHU, H.; ZHAO, P.; LIU, S.; KE, X. A slam based navigation system for the indoor embedded control mobile robot. In: IEEE. **2020 4th INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION SCIENCES (ICRAS)**. [S.l.], 2020. p. 22–27.