Hiparco Lins Vieira

Redução do Custo Computacional do Algoritmo RRT Através de Otimização por Eliminação

Dissertação apresentada à Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Mestre em Ciências, Programa de Engenharia Elétrica.

Área de Concentração: Sistemas Dinâmicos
Orientador: Prof. Dr. Valdir Grassi Junior

São Carlos
2014

Trata-se da versão corrigida da dissertação. A versão original se encontra disponível na EESC/USP que aloja o programa de Pós-Graduação de Engenharia Elétrica.

AUTORIZO A REPRODUÇÃO TOTAL OU PARCIAL DESTE TRABALHO,
POR QUALQUER MEIO CONVENCIONAL OU ELETRÔNICO, PARA FINS
DE ESTUDO E PESQUISA, DESDE QUE CITADA A FONTE.

V658r Vieira, Hiparco Lins
 Redução do custo computacional do algoritmo RRT
 através de otimização por eliminação / Hiparco Lins
 Vieira; orientador Valdir Grassi Junior. São Carlos,
 2014.

 Dissertação (Mestrado) - Programa de Pós-Graduação
 em Engenharia Elétrica e Área de Concentração em
 Sistemas Dinâmicos -- Escola de Engenharia de São
 Carlos da Universidade de São Paulo, 2014.

 1. Ambientes dinâmicos. 2. Otimização. 3.
 Planejamento de trajetórias. 4. Replanejamento de
 trajetórias. 5. RRT. 6. Primitivas de movimento. 7.
 Custo computacional. I. Título.

FOLHA DE JULGAMENTO

Candidato: Engenheiro **HIPARCO LINS VIEIRA.**

Título da dissertação: "Redução do custo computacional do algoritmo RRT através de otimização por eliminação".

Data da defesa: 15/07/2014

Comissão Julgadora:

Resultado:

Prof. Dr. **Valdir Grassi Júnior (Orientador)**
(Escola de Engenharia de São Carlos/EESC)

Aprovado

Prof. Associado **Denis Fernando Wolf**
(Instituto de Ciências Matemáticas e Computação/ICMC)

Aprovado

Prof. Dr. **Luiz Chaimowicz**
(Universidade Federal de Minas Gerais/UFMG)

Aprovado.

Coordenador do Programa de Pós-Graduação em Engenharia Elétrica e
Presidente da Comissão de Pós-Graduação:
Prof. Titular **Denis Vinicius Coury**

Dedicatória

É com carinho que dedico este trabalho aos meus pais, Lisa Mara e Hiparco, e às minhas irmãs, Andrezza, Maria Clara e Livia, pelo constante apoio, carinho, amizade e palavras de motivação.

Agradecimentos

Ao meu orientador, Prof. Dr. Valdir Grassi Jr, meus sinceros agradecimentos pelo empenho, pela dedicação, pelas valiosas discussões durante o andamento da pesquisa e pela confiança depositada em mim.

À instituição Universidade de São Paulo em São Carlos por toda excelente infraestrutura oferecida.

Aos amigos e afins do grupo Léon Denis, por todo carinho, consideração, companhia, apreço, amizade, grandes momentos vividos durante o mestrado.

Aos amigos do Laboratório de Sistemas Inteligentes - LASI pelo companheirismo e grandes momentos em que estivemos juntos, que me trouxeram muita alegria e muito aprendizado.

À CAPES (Comissão de Aperfeiçoamento de Pessoal de Nível Superior) e à FAPESP (Fundação de Amparo à Pesquisa do Estado de São Paulo) pela bolsa de mestrado concedida para o desenvolvimento deste trabalho.

Aos amigos anônimos que muito contribuíram para que eu pudesse realizar mais esta etapa da minha vida, onde obtive valiosos ensinamentos.

Aos meus pais, avós e avôs, tios, tias e irmãs, por estarem ao meu lado frente às dificuldades.

A Deus pela benção, saúde, pelas constantes oportunidades de aprendizado, amizades desenvolvidas, e valiosos ensinamentos.

Resumo

Vieira, H. L. Redução do Custo Computacional do Algoritmo RRT Através de Otimização por Eliminação. Dissertação (Mestrado) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2014.

A aplicação de técnicas baseadas em amostragem em algoritmos que envolvem o planejamento de trajetórias de robôs tem se tornado cada vez mais difundida. Deste grupo, um dos algoritmos mais utilizados é chamado Rapidly-exploring Random Tree (RRT), que se baseia na amostragem incremental para calcular de forma eficiente os planos de trajetória do robô evitando colisões com obstáculos. Vários esforços tem sido realizados a fim de reduzir o custo computacional do algoritmo RRT, visando aplicações que necessitem de respostas mais rápidas do algoritmo, como, por exemplo, em ambientes dinâmicos. Um dos dilemas relacionados ao RRT está na etapa de geração de primitivas de movimento. Se várias primitivas são geradas, permitindo o robô executar vários movimentos básicos diferentes, um grande custo computacional é gasto. Por outro lado, quando poucas primitivas são geradas e, conseqüentemente, poucos movimentos básicos são permitidos, o robô pode não ser capaz de encontrar uma solução para o problema, mesmo que esta exista. Motivados por este problema, um método de geração de primitivas de movimento foi proposto. Tal método é comparado com os métodos tradicional e aleatório de geração de primitivas, considerando não apenas o custo computacional de cada um, mas também a qualidade da solução obtida. O método proposto é aplicado ao algoritmo RRT, que depois é aplicado em um caso de estudo em um ambiente dinâmico. No estudo de caso, o algoritmo RRT otimizado é avaliado em termos de seus custos computacionais durante planejamentos e replanejamento de trajetória. As simulações são realizadas em dois simuladores: um desenvolvido em linguagem Python e outro em Matlab.

Palavras-Chave: *ambientes dinâmicos, otimização, planejamento de trajetórias, replanejamento de trajetórias, RRT, primitivas de movimento, custo computacional.*

Abstract

*Vieira, H. L. **Reduction in the Computational Cost of the RRT Algorithm through Optimization by Elimination.** Dissertation (Master's Degree) - Engineering School of Sao Carlos, University of Sao Paulo, Sao Carlos, 2014.*

The application of sample-based techniques in path-planning algorithms has become year-by-year more widespread. In this group, one of the most widely used algorithms is the Rapidly-exploring Random Tree (RRT), which is based on an incremental sampling of configurations to efficiently compute the robot's path while avoiding obstacles. Many efforts have been made to reduce RRT computational costs, targeting, in particular, applications in which quick responses are required, e.g., in dynamic environments. One of the dilemmas posed by the RRT arises from its motion primitives generation. If many primitives are generated to enable the robot to perform a broad range of basic movements, a significant computational cost is required. On the other hand, when only a few primitives are generated, thus, enabling a limited number of basic movements, the robot may be unable to find a solution to the problem, even if one exists. To address this quandary, an optimized method for primitive generation is proposed. This method is compared with the traditional and random primitive generation methods, considering not only computational cost, but also the quality of local and global solutions that may be attained. The optimized method is applied to the RRT algorithm, which is then used in a case study in dynamic environments. In the study, the modified RRT is evaluated in terms of the computational costs of its planning and replanning. The simulations were developed to assess the effectiveness and efficiency of the proposed algorithm.

Keywords: *dynamic environments, optimization, path planning, path replanning,*

RRT, motion primitives, computational cost.

Lista de Figuras

FIGURA 2.1	Exemplo de uma árvore RRT. (Imagem extraída de LaValle (1998).)	7
FIGURA 2.2	Diagrama de Voronoi de uma árvore gerada pelo algoritmo RRT. Imagem adaptada de Yershova et al. (2005).	9
FIGURA 2.3	Geração da amostra aleatória e no espaço livre de configuração do robô e expansão das primitivas de movimento.	11
FIGURA 2.4	Eliminação das primitivas que geram colisão.	11
FIGURA 2.5	Adição da primitiva de menor custo e livre de colisão à árvore.	12
FIGURA 2.6	Uma nova configuração aleatória é amostrada no espaço livre de configurações.	12
FIGURA 2.7	A árvore expande até encontrar o destino ou até que o número limite de amostras aleatórias seja alcançado.	13
FIGURA 2.8	A trajetória de menor custo que conecta o estado inicial ao estado final é retornada pelo algoritmo.	13
FIGURA 4.1	O intervalo de valores que cada entrada pode assumir é dividido inicialmente dois subintervalos e são geradas três amostras espaçadas uni- formemente: uma situada no limite superior do intervalo, outra do limite inferior e a última no meio do intervalo.	26
FIGURA 4.2	Primitivas geradas na primeira iteração do algoritmo otimizado.	27
FIGURA 4.3	Eliminação de área não-promissora.	28

FIGURA 4.4	Uma nova amostra é feita no meio do intervalo remanescente da entrada.	29
FIGURA 4.5	Uma nova primitiva é gerada.	29
FIGURA 4.6	A primitiva livre de colisão de menor custo é retornada ao algoritmo RRT.	30
FIGURA 4.7	(a) Amostras realizadas em uma entrada do algoritmo GOP para $m = 1$, com resolução $r = 0,5$. (b) Amostras realizadas em uma entrada do algoritmo GOP para $m = 2$, com resolução $r = 0,25$. (c) Amostras realizadas em uma entrada do algoritmo GTP com resolução $r = 0,25$. . .	31
FIGURA 5.1	Modelo de bicicleta para robôs em forma de carro apresentado em dois sistemas de coordenadas: (x,y) e (u,w) . O sistema de coordenadas (u,w) é fixo no carro. Fonte: Moret (2003)	36
FIGURA 5.2	Forças resultantes nas rodas do veículo. Imagem extraída de Moret (2003)	38
FIGURA 6.1	Imagem de um ambiente gerado aleatoriamente.	44
FIGURA 6.2	Gráfico do aumento do tempo computacional em função do número de iterações m realizadas, comparando-se os algoritmos GTP, GOP e GAP. (a) Gráfico para as situações onde todas primitivas geradas estavam livres de colisões. (b) Gráfico para situações onde todas as primitivas geradas causam colisão.	46
FIGURA 6.3	Imagem do ambiente de simulação e da árvore obtida através do método RRT-O.	52
FIGURA 6.4	Gráfico com o tempo computacional em relação ao número m de iterações para os algoritmos RRT-T e RRT-O.	54
FIGURA 6.5	Gráfico com o tempo computacional em relação ao número de amostras aleatórias dos algoritmos RRT-T e RRT-O necessárias para obter uma solução ou retornar a ausência de uma.	55

- FIGURA 6.6 (a) Configuração inicial do ambiente, enquanto o robô executa o algoritmo RRT otimizado. (b) Planejamento obtido inicialmente pelo algoritmo RRT otimizado. (c) Início da movimentação do robô e dos obstáculos dinâmicos. (d) Sensor detecta a presença de um obstáculo dinâmico e avalia se a trajetória planejada. Neste caso, o robô detecta que haverá colisão com o obstáculo e replaneja a trajetória. 59
- FIGURA 6.7 (a) Robô detecta o obstáculo dinâmico 1. (b) Robô replaneja a trajetória e passa entre os dois obstáculos dinâmicos sem causar colisão. (c) Robô após passar entre os obstáculos dinâmicos. (d) Robô executando a trajetória em direção ao destino. 60
- FIGURA 6.8 (a) Robô se aproximando do destino. (b) Robô alcança o destino sem gerar colisões. 61
- FIGURA 6.9 Gráfico com o custo computacional de planejamento do algoritmo RRT-O em ambiente dinâmico em relação ao tempo de integração. 62
- FIGURA 6.10 Gráfico com o custo computacional de replanejamento do algoritmo RRT-O em ambiente dinâmico em relação ao tempo de integração. . . 63

Lista de Tabelas

TABELA 6.1	Tabela com o custos computacionais médios, em milissegundos, de cada um dos três métodos analisados, em função de m . Tais valores foram obtidos em ambientes onde todas as primitivas geradas estavam livre de colisão.	46
TABELA 6.2	Tabela com desvios padrão em milissegundos obtidos com a simulação para os casos onde todas primitivas estavam livres de colisão. . . .	47
TABELA 6.3	Tabela com o custos computacionais médios, em milissegundos, de cada um dos três métodos analisados, em função de m . Tais valores foram obtidos em ambientes onde todas as primitivas geradas causavam colisão.	48
TABELA 6.4	Tabela com desvios padrão em milissegundos obtidos com a simulação para os casos onde todas primitivas geradas colidiam.	48
TABELA 6.5	Comparativo dos algoritmos GTP e GOP para $m = 2$	49
TABELA 6.6	Comparativo dos algoritmos GAP e GOP para $m = 2$	50
TABELA 6.7	Comparativo dos algoritmos GTP e GOP para $m = 5$	51
TABELA 6.8	Comparativo dos algoritmos GAP e GOP para $m = 5$	51
TABELA 6.9	Valores máximo e mínimos de tempo de planejamento do algoritmo RRT-O em ambientes dinâmicos e os respectivos tempos de integração. .	62
TABELA 6.10	Valores máximo e mínimos de tempo de replanejamento do algoritmo RRT-O em ambientes dinâmicos e os respectivos tempos de integração. .	64

Lista de Abreviaturas e Siglas

CaRINA - **C**arro **R**obótico **I**nteligente para **N**avegação **A**utônoma

GAP - **G**eração **A**leatória de **P**rimitivas

GOP - **G**eração **O**timizada de **P**rimitivas

GTP - **G**eração **T**radicional de **P**rimitivas

PRM - **P**robabilistic **R**oad**M**aps

RRT - **R**apidly-exploring **R**andom **T**ree

RRT-O - **R**apidly-exploring **R**andom **T**ree **O**timizado

RRT-T - **R**apidly-exploring **R**andom **T**ree **T**radicional

SIPP - **S**afe **I**nterval **P**ath **P**lanning

Sumário

1	Introdução	3
	Introdução	3
1.1	Problema	5
1.2	Objetivos	5
1.3	Organização do Texto	6
2	Algoritmo <i>Rapidly-exploring Random Tree</i> (RRT)	7
2.1	Aspectos gerais do algoritmo RRT	8
2.2	Funcionamento do algoritmo RRT básico	10
2.3	Algoritmo RRT básico	12
2.4	Geração de primitivas de movimento	14
2.4.1	Método tradicional de geração de primitivas	14
2.4.2	Método aleatório de geração de primitivas	15
3	Trabalhos Relacionados	17
3.1	Otimização do Algoritmo RRT	17
3.2	Geração de Primitivas de Movimento	19
3.3	Ambientes Dinâmicos	20
4	Metodologia	25
4.1	Geração otimizada de primitivas de movimento	26
4.1.1	Geração inicial de primitivas	26
4.1.2	Identificação de áreas promissoras	27
4.1.3	Eliminação de áreas não-promissoras	28
4.2	Comparação entre o método tradicional e o método otimizado	29
4.3	Comparação entre o método aleatório e o método otimizado	32

5	Modelo matemático do veículo	35
5.1	Modelo cinemático	36
5.2	Modelo dinâmico	38
5.3	Método de Runge-Kutta de Quarta Ordem	40
6	Resultados	43
6.1	Otimização da geração de primitivas de movimento	43
6.1.1	O ambiente de simulação	44
6.1.2	Modelo do robô	45
6.1.3	Comparação entre os custos computacionais	45
6.1.4	Análise dos custos de extensão	48
6.2	Otimização do algoritmo RRT	52
6.2.1	O ambiente de simulação	52
6.2.2	Modelo do robô	53
6.2.3	Comparação entre RRT-T e RRT-O	53
6.3	Estudo de caso: Replanejamento de trajetória em ambiente dinâmico utilizando o algoritmo RRT-O	56
6.3.1	O ambiente de simulação	56
6.3.2	Modelo do robô	56
6.3.3	Replanejamento de trajetória em ambiente dinâmico	57
6.3.4	Análise do tempo de replanejamento do algoritmo	61
7	Conclusão	65
	Conclusões	65
	Referências Bibliográficas	67

Capítulo 1

Introdução

Dentre as maiores dificuldades relacionadas à robótica autônoma está o planejamento de trajetórias de forma automática (Choset et al.; 2005). Este planejamento consiste em gerar uma sequência de comandos de controle que leva o robô a se deslocar de um ponto inicial até o ponto desejado evitando colisões com obstáculos presentes no ambiente. Os algoritmos de planejamento de trajetórias tem sido aplicados em diversas áreas, como, por exemplo, em robôs aéreos (Gang et al.; 2011), (Hota e Ghose; 2010), (Wu et al.; 2009), em robôs marítimos (Hausler et al.; 2009), (Lee et al.; 2011) e em veículos autônomos terrestres (Kuwata et al.; 2008), (Baek et al.; 2010), (Schroder et al.; 2008).

Para realizar o planejamento, o robô utiliza dados que caracterizam o ambiente no qual está inserido. Nos dados referentes ao ambiente podem estar contidas informações como: limites do ambiente, presença e localização de obstáculos, incertezas, etc. Tais dados podem ser obtidos através de leituras dos sensores do robô ou de parâmetros fornecidos por um projetista.

Durante o planejamento e sua execução em ambiente real, o robô está sujeito a incertezas. Dentre alguns fatores que podem gerar incerteza, destacam-se: incertezas do robô (modelos, sensores e atuadores), incertezas do ambiente (presença de obstáculos não previstos, mudança no coeficiente de atrito com a superfície de navegação do robô, obstáculos em movimento, ambiente previamente desconhecido) e incertezas relacionadas ao algoritmo (tempo de planejamento e comportamentos inesperados do *software*). Caso o sistema não esteja preparado para lidar com incertezas, a realização da tarefa pode ser

comprometida.

O tempo de planejamento e execução da trajetória é de suma importância na escolha do algoritmo. Para demonstrar, cita-se o exemplo de um automóvel autônomo no trânsito. Caso o automóvel não realize o planejamento de trajetória em tempo hábil, podem ocorrer congestionamentos e acidentes de trânsito.

A obtenção de uma trajetória ótima também merece grande atenção. Ao levar-se em conta a geração de uma trajetória ótima, abre-se mão da eficiência computacional e vice-versa. No exemplo do automóvel autônomo, a escolha de uma trajetória não-ótima pode levar o veículo a percorrer caminhos mais custosos.

Para suprir tais necessidades, diversos algoritmos para o planejamento de trajetórias foram desenvolvidos. Vários destes métodos podem ser encontrados em Choset et al. (2005) e LaValle (2006). Algumas das técnicas apresentadas são: *bug algorithms*, campos potenciais artificiais, *probabilistic roadmaps* e *rapidly-exploring random trees*.

A escolha do método de planejamento influencia na navegação do robô, no tempo de resposta e na atitude a ser tomada quando o robô se depara com um obstáculo não previsto no mapa. Alguns métodos representam o robô pelas coordenadas do seu centro (x,y) e utilizam um modelo omnidirecional de movimento para o robô, ou seja, o robô não possui restrições de movimento e pode mover-se em todas as direções. Com isso desprezam-se as restrições dinâmicas e cinemáticas do robô. Um automóvel comum, por exemplo, não pode mover-se na direção perpendicular às rodas traseiras. Tal restrição é chamada restrição não-holonômica. Logo, em tal caso, o automóvel precisa realizar várias manobras para alcançar determinadas posições.

Além disso, para a realização de determinada trajetória é necessário o conhecimento da dinâmica do veículo, como torque, atrito com o solo, massa, inércia, etc. Um carro em movimento, por exemplo, não consegue realizar frenagem instantânea. Baseando-se nisso, percebe-se a importância, em muitos casos, de se trabalhar com as restrições dinâmicas e cinemáticas do robô, utilizando os métodos de planejamento apropriados, garantindo que o robô seja capaz de executar a trajetória planejada.

Um método de planejamento que leva em consideração as equações cinemáticas e

dinâmicas do robô é o algoritmo *rapidly-exploring random trees* (RRT), que realiza amostragens no espaço de trabalho livre, gerando uma árvore a partir do ponto inicial até o destino. Tal algoritmo tem sido usado em diversos ramos de pesquisa, dentre eles destacam-se aplicações no planejamento de trajetórias de veículos em ambientes urbanos (Kuwata et al.; 2008), veículos aéreos autônomos (Kim e Ostrowski; 2003), robôs humanóides (Xia et al.; 2009), engenharia celular (Ju et al.; 2011), robôs manipuladores (Lacevic et al.; 2011) e em futebol de robôs (Bruce e Veloso; 2002).

1.1 Problema

Durante o procedimento de expansão da árvore do algoritmo RRT, onde são geradas as denominadas primitivas de movimento, o algoritmo RRT básico enfrenta também o problema do balanceamento do custo computacional com a qualidade da solução obtida. A quantidade de primitivas geradas está relacionada à obtenção da primitiva livre de colisão e de menor custo de expansão. Quanto mais primitivas são geradas, maiores são as opções de movimento dadas ao robô, o que pode levar a obtenção de primitivas de menor custo e a maiores possibilidades de se evitar uma colisão. Contudo, um aumento do número de primitivas geradas leva a um aumento do custo computacional. Assim, percebe-se a necessidade de uma melhoria em tal etapa do algoritmo RRT afim de atenuar o problema.

1.2 Objetivos

O presente trabalho tem como objetivos:

- propor um método para geração de primitivas no algoritmo RRT que diminua com custo computacional reduzido, sem perda significativa de qualidade das primitivas geradas;
- comparar o método de proposto com métodos comuns de geração de primitivas em termos de custo computacional e qualidade de solução;

- avaliar a influência do método proposto no algoritmo RRT

1.3 Organização do Texto

O texto está organizado no seguinte formato:

- Capítulo 2: apresenta o funcionamento do algoritmo RRT e os algoritmos tradicional e aleatório de geração de primitivas de movimento;
- Capítulo 3: apresenta as principais características do algoritmo RRT, geração de primitivas de movimento e ambientes dinâmicos;
- Capítulo 4: apresenta o método otimizado proposto para a geração de primitivas de movimento no algoritmo RRT;
- Capítulo 5: apresenta o modelo matemático do robô e o método numérico de integração utilizados nas simulações;
- Capítulo 6: apresenta o ambiente de simulação, as simulações e os resultados obtidos com a otimização do algoritmo RRT e um estudo de caso do algoritmo em ambientes dinâmicos;
- Capítulo 7: apresenta as conclusões obtidas durante o presente trabalho e propostas para trabalhos futuros.

Capítulo 2

Algoritmo Rapidly-exploring Random Tree (RRT)

O algoritmo RRT básico, proposto em LaValle (1998), de uma forma geral, gera uma árvore de configurações através de amostras aleatórias uniformes no espaço livre de configuração e expande a árvore em direção ao destino evitando colisões com obstáculos. Os estados da árvore representam as configurações do robô e estão ligados através das entradas de comando que levam o robô a se dirigir de um estado até o próximo. Na Figura 2.1 é mostrada uma árvore construída pelo algoritmo RRT.

Figura 2.1: Exemplo de uma árvore RRT. (Imagem extraída de LaValle (1998).)

2.1 Aspectos gerais do algoritmo RRT

Dentre as principais propriedades do algoritmo RRT, estão (LaValle; 1998), (Kuffner e LaValle; 2000), (Bruce e Veloso; 2002):

1. A tendência de explorar áreas ainda não exploradas do espaço de configurações de forma rápida;
2. A característica de ser um algoritmo probabilisticamente completo. Isto quer dizer que a chance de o algoritmo retornar uma solução (ou a ausência de uma) tende a 1 quando o tempo de execução do algoritmo tende ao infinito;
3. Simplicidade, facilitando análises de performance;
4. Todos os estados da árvore estão conectados;
5. As entradas necessárias para realizar a trajetória estão disponíveis no final do planejamento;
6. Não requer um pré-processamento do ambiente ao realizar o planejamento;
7. Possibilidade de resolver problemas com vários graus de liberdade;

O algoritmo RRT consegue explorar partes inexploradas do ambiente. Uma das formas de se realizar tal tarefa é utilizando o diagrama de Voronoi. O diagrama é construído a partir dos estados que compõem a árvore RRT, conforme proposto em Yershova et al. (2005). A probabilidade de uma configuração da árvore ser escolhida como pai de uma nova configuração é proporcional ao volume de sua respectiva região de Voronoi. Na Figura 2.2, é mostrado o gráfico de Voronoi de uma árvore RRT. Percebe-se que as configurações mais externas da árvore possuem uma região de Voronoi maior do que aquelas situadas na região central da árvore. Assim, a tendência da árvore é explorar regiões do espaço livre ainda não exploradas (Yershova et al.; 2005). Baseando-se nesta propriedade, resulta-se que a árvore não está sujeita à mínimos locais, ao contrário do tradicional método de Campos Potenciais.

O RRT não garante que uma solução ou a ausência de uma seja encontrada em tempo finito. Contudo, caso seja dado ao RRT um tempo suficientemente grande, este pode

Figura 2.2: Diagrama de Voronoi de uma árvore gerada pelo algoritmo RRT. Imagem adaptada de Yershova et al. (2005).

cobrir o espaço livre de trabalho com um número suficientemente grande de amostras, analisando, assim, o ambiente e retornando uma resposta para o problema. Percebe-se então que, quanto maior o tempo, maior é a probabilidade de o algoritmo retornar uma solução para o problema. Assim, o RRT é considerado um algoritmo probabilisticamente completo.

Na geração da árvore, é possível levar em consideração as equações dinâmicas e cinemáticas do robô, fazendo com que as configurações adicionadas à árvore sejam alcançáveis pelo robô. Isso ocorre porque, ao adicionar-se uma configuração à árvore, são adicionadas também as entradas de controle necessárias para levar o robô da configuração inicial à configuração desejada. Desta forma, a árvore permanece conectada. Após a obtenção da trajetória, como as entradas de controle já foram encontradas, teoricamente, tudo o que o sistema deve fazer é passar tais entradas aos controladores do robô. Contudo, isto dificilmente acontece na prática, em virtude das incertezas tanto no modelo do robô, quanto em sua localização, imprecisões dos sensores, dentre outras.

Por ser um método baseado em amostragens e por não precisar realizar um pré-processamento do ambiente, o RRT consegue trabalhar com vários graus de liberdade. Isso o torna uma solução rápida para o planejamento cinemático-dinâmico de trajetórias. Para a resolução de problemas com altos graus de liberdade, os métodos que não se baseiam em amostragem, em geral, apresentam como desvantagem uma explosão computacional da complexidade do algoritmo, tornando impraticável ou inviável a sua utilização.

2.2 Funcionamento do algoritmo RRT básico

No início da execução, o algoritmo RRT recebe os seguintes parâmetros: o mapa do ambiente, o estado inicial (c_{ini}) e final (c_{fim}) do robô, além do número máximo (n_{max}) de amostras (configurações aleatórias do robô no espaço livre de trabalho) a serem geradas.

A posição dos obstáculos está especificada no mapa do ambiente. Tal mapa, a menos que seja atualizado, não leva em consideração a mudança de posicionamento dos obstáculos. O estado inicial e final do robô representam o ponto de partida e chegada do robô, respectivamente. O número máximo de amostras geradas limita o tempo de execução do algoritmo. O algoritmo gerará amostras aleatórias até atingir o número máximo estabelecido ou até encontrar uma trajetória que leve o robô a partir do ponto inicial até o ponto final, sem gerar colisões com obstáculos, ou seja, uma solução para o problema. Caso não seja encontrada uma solução, o algoritmo pode ser reexecutado ou retornar a inexistência de uma solução.

O funcionamento do algoritmo RRT é descrito nas Figuras 2.3-2.8, onde o robô é apresentado em vermelho, o destino em azul, o obstáculos em cinza e as primitivas de movimento e as amostras aleatórias em preto.

Inicia-se o algoritmo com o procedimento *INICIA_ARVORE*, situando-se o estado inicial c_{ini} como raiz da árvore e gerando uma árvore com um único estado. Gera-se, então, a partir do procedimento *ESTADO_ALEATORIO*, com probabilidade $1-p$, uma configuração aleatória c_{ale} , pertencente ao espaço livre de configuração. A configuração final c_{fim} desejada para o robô é amostrada com uma probabilidade p . Nota-se que a soma a probabilidade de amostragem do estado final com a dos estados aleatórios é igual a 1. Assim, quanto maior for o valor de p , mais rápido a árvore cresce em direção à configuração final, necessitando de um menor número de iterações. No entanto, caso o valor de p seja muito alto, o RRT pode sofrer com o problema de mínimos locais.

Em seguida, é invocado o procedimento *ARV_PROXIMO*, onde os estados pertencentes à árvore são avaliados quanto à suas respectivas distâncias euclidianas (ou quanto à uma outra função custo) em relação à configuração c_{ale} . O estado c_{pai} de menor distância em relação à c_{ale} é selecionado. Em seguida, executa-se o procedimento

Figura 2.3: Geração da amostra aleatória e no espaço livre de configuração do robô e expansão das primitivas de movimento.

EXPANDE_NO, onde gera-se, a partir de c_{pai} , um número n de primitivas de movimento c_{ger} , mostrado na Figura 2.3. Estas trajetórias podem ser alcançadas em um intervalo limitado de tempo Δt . Acrescenta-se ainda que as primitivas geradas levam em consideração as equações cinemáticas e dinâmicas do robô. Isso garante que o robô será capaz de executar as trajetórias planejadas pelo algoritmo.

Figura 2.4: Eliminação das primitivas que geram colisão.

As trajetórias geradas a partir de c_{pai} que levam a c_{ger} são analisadas pelo procedimento *CHECA_COLISAO*. Aquelas trajetórias que resultam em colisão com obstáculos são eliminadas, como mostrado na Figura 2.4. Então, é invocado o procedimento *NO_PROXIMO*, que calcula o custo de navegação entre cada estado c_{ger} remanescente em relação a c_{ale} . O estado c_{novo} que possuir menor custo de extensão dentre os estados remanescentes de c_{ger} é adicionado à árvore, como filho de c_{pai} , através do procedimento *NOVO_ESTADO*, como mostrado na Figura 2.5. Juntamente com c_{novo} são também incluídas à árvore as entradas de controle que fazem com que o robô alcance c_{novo} a partir de c_{pai} .

Figura 2.5: Adição da primitiva de menor custo e livre de colisão à árvore.

Figura 2.6: Uma nova configuração aleatória é amostrada no espaço livre de configurações.

Em seguida é amostrada novamente uma configuração c_{ale} , como mostrado na Figura 2.6 e a iteração se repete até que seja obtida uma solução para o problema ou até que o número de configurações amostradas seja igual ao número de configurações limite. Neste último caso, o algoritmo retorna a ausência de uma solução.

Quando a árvore A alcança o destino, como mostrado na Figura 2.7 busca-se, utilizando a função $BEST_TRAJ$, a trajetória de menor custo que leve o robô de c_{ini} a c_{fim} , como mostrado na Figura 2.8.

2.3 Algoritmo RRT básico

O pseudo código que representa o algoritmo RRT básico está apresentado no algoritmo

1. Os procedimentos que compõem esse algoritmo são:

Figura 2.7: A árvore expande até encontrar o destino ou até que o número limite de amostras aleatórias seja alcançado.

Figura 2.8: A trajetória de menor custo que conecta o estado inicial ao estado final é retornada pelo algoritmo.

Algoritmo 1: Algoritmo RRT

```

 $A \leftarrow INICIA\_ARVORE(c_{ini});$ 
while  $NOT(Solucao \text{ OR } NumMax)$  do
     $c_{ale} \leftarrow ESTADO\_ALEATORIO();$ 
     $c_{pai} \leftarrow ARV\_PROXIMO(A, c_{ale});$ 
     $c_{ger} \leftarrow EXPANDE\_NO(c_{pai});$ 
     $c_{ger} \leftarrow CHECA\_COLISAO(c_{ger});$ 
     $c_{novo} \leftarrow NO\_PROXIMO(c_{ger});$ 
     $NOVO\_ESTADO(A, c_{novo}, c_{pai});$ 
if  $Solucao$  then
    |  $return \text{ BEST\_TRAJ}(A, c_{ini}, C_{fim});$ 
else
    |  $return \text{ NULL};$ 

```

INICIA_ARVORE(c_{ini}): Inicia a árvore A situando a raiz na configuração inicial c_{ini} ;

ESTADO_ALEATORIO(): Gera uma configuração aleatória c_{ale} para o robô, através de uma distribuição uniforme no espaço livre de configuração;

ARV_PROXIMO(A, c_{ale}): Identifica o estado c_{pai} pertencente a árvore A mais próximo à c_{ale} ;

EXPANDE_NO(c_{pai}): Recebe como entrada um estado do robô c_{pai} e calcula um número n de primitivas de movimento, representadas por c_{ger} , que podem ser alcançadas pelo robô, de acordo com as equações cinemáticas e dinâmicas. Cada primitiva gerada está relacionada a um vetor de entrada u , aplicado ao sistema durante um tempo Δt ;

CHECA_COLISAO(c_{ger}): Elimina as primitivas c_{ger} que estão em colisão, retornando as demais primitivas;

NO_PROXIMO(c_{ger}): Escolhe, dentre os estados finais de cada uma das primitivas c_{ger} , o estado c_{novo} com menor custo de navegação em relação a c_{ale} e livre de colisões;

NOVO_ESTADO(A, c_{novo}, c_{pai}): Adiciona c_{novo} à árvore A , como filho de c_{pai} , e as respectivas entradas de controle u necessárias para a realização da trajetória.

BEST_TRAJ(A, c_{ini}, c_{fim}): Retorna a trajetória de menor custo de navegação da árvore A que leva o robô de c_{ini} a c_{fim} .

2.4 Geração de primitivas de movimento

Nesta seção são descritas duas maneiras de gerar as primitivas de movimento: o método tradicional e o método aleatório. Tais métodos podem ser utilizados na função *EXPANDE_NO* do algoritmo RRT.

2.4.1 Método tradicional de geração de primitivas

O método tradicional de geração de primitivas, utilizado no algoritmo RRT básico, realiza uma combinação de um conjunto de amostras das entradas do sistema para gerar

as primitivas de movimento.

Para cada entrada do sistema, são geradas um determinado número de amostras igualmente espaçadas uma da outra. À partir de tais amostras são formados diferentes vetores de entrada do sistema. Os vetores gerados e o estado inicial do robô são utilizados para calcular as primitivas de movimento através da integração das equações do sistema durante um intervalo de tempo Δt .

Após a geração das primitivas, descartam-se aquelas que gerarem colisões com obstáculos. No grupo de primitivas remanescentes, avalia-se o custo de extensão de cada uma. O termo custo de extensão está relacionado com o custo que o robô tem de executar a primitiva gerada, como, por exemplo, o tempo de execução, o gasto de energia ou combustível e a distância. Então, retorna-se como solução aquela primitiva que possuir menor custo de extensão.

Uma desvantagem do método tradicional está no fato de que caso para um determinado estado inicial não haja possibilidade de movimento, todas as primitivas serão geradas e avaliadas quanto a existência de colisões. Dessa forma, um considerável tempo computacional é perdido.

2.4.2 Método aleatório de geração de primitivas

O método aleatório de geração de primitivas de movimento, assim como o método tradicional, realiza uma combinação de um conjunto de amostras das entradas do sistema para gerar as primitivas de movimento.

Contudo, as amostras de cada entrada são geradas de forma aleatória. Em seguida, a partir das amostras, são formados diferentes vetores de entrada. Tais vetores e os estados iniciais do robô são utilizados na integração das equações do robô em Δt . Como resultado, são geradas as primitivas de movimento. As primitivas que geram colisão são descartadas e, dentre as remanescentes, é retornada aquela com menor custo de extensão.

Capítulo 3

Trabalhos Relacionados

Vários trabalhos foram realizados com o intuito de melhorar o algoritmo RRT básico, tanto em relação ao custo computacional quanto à qualidade das soluções obtidas, visto que ambos estão diretamente relacionados. Alguns destes trabalhos são abordados neste capítulo.

3.1 Otimização do Algoritmo RRT

Em Karaman e Frazzoli (2011) é mostrado que a chance de o algoritmo RRT convergir para uma solução ótima é aproximadamente zero. No mesmo artigo foi proposta uma variação do algoritmo RRT, conhecida como RRT*, que possui a propriedade de ser assintoticamente ótimo. O RRT* realiza uma reconexão dos nós pertencentes à árvore em conexões de menor custo de navegação, dessa forma, melhorando a solução ao longo do tempo. Contudo, para problemas que envolvem restrições cinemáticas e dinâmicas, o RRT* necessita de um tempo substancial para melhorar a solução obtida. Isso torna-o inviável para aplicações que necessitem de uma rápida resposta.

Em Urmson e Simmons (2003), é apresentada uma técnica que visa gerar trajetórias de menor custo de navegação, através de alterações no algoritmo RRT. Os autores apresentam uma heurística que expande a RRT preferencialmente para regiões de menor custo de navegação, diminuindo-se o custo da solução obtida. Durante a escolha do nó mais

próximo à amostra, o algoritmo RRT básico seleciona apenas um nó. No mesmo trabalho é proposta uma técnica que seleciona mais de um nó para expansão, identificando os nós mais próximos da amostra e expandindo-os. A expansão de melhor qualidade é escolhida. Como consequência, há uma melhoria na qualidade da solução obtida. Em contrapartida, há um significativo aumento no custo computacional.

No início da execução do algoritmo RRT, quando o nó final desejado para o robô é definido, estipula-se uma tolerância para o robô alcançar este nó, a fim de facilitar a convergência do algoritmo. Em problemas com vários graus de liberdade, pode haver dificuldade do algoritmo alcançar uma solução para o problema. Esse problema ocorre devido ao fato de se gerar um estado aleatório que combine vários graus de liberdade em que todos estejam dentro da tolerância permitida para a configuração final. Neste caso, a árvore gerada pelo algoritmo RRT pode expandir até uma região muito próxima do nó final, mas sem conseguir alcançá-lo.

Assim, para que o algoritmo obtenha a solução em um menor tempo, é inserido um determinismo na taxa de amostragem do RRT. Ao gerar-se uma amostra aleatória no espaço, o algoritmo escolhe como amostra, com uma probabilidade p , a configuração final desejada. As demais configurações são geradas com probabilidade $1 - p$. Quanto maior for o valor de p , mais rápido a RRT tende a se expandir em direção à configuração desejada.

Contudo, caso o valor de p seja elevado, o RRT pode sofrer com o problema de mínimos locais, o que poderia dificultar ou mesmo impedir a obtenção de uma solução. Caso o valor de p seja muito baixo, a RRT pode chegar próximo do nó final desejado mas, pode não alcançá-lo. Além disso, neste último caso, o tempo para o algoritmo retornar uma solução é maior, necessitando gerar mais nós e, assim, ocasionando um maior sobrecarregamento computacional. Em Urmson e Simmons (2003), é mostrada a diferença entre os resultados de duas RRTs: uma com o valor de $p = 0,05$ e outra com o valor $p = 0,10$. Pode-se observar no trabalho que o aumento do valor de p claramente reduz o número de amostras necessárias para obter uma solução, tanto em um ambiente com obstáculos como em um ambiente livre deles.

Outro problema está na complexidade computacional do RRT no decorrer do tempo. Após o algoritmo gerar uma configuração aleatória, é feita uma busca nos nós da RRT a

fim de identificar o nó mais próximo da configuração amostrada. No início do algoritmo, há apenas um nó para ser analisado. Contudo, ao longo do tempo, o número de nós presentes na árvore aumenta. Com isso, aumenta-se o número de nós a serem analisados pelo algoritmo, aumentando-se o tempo computacional necessário para a execução de cada iteração. Ou seja, o RRT, no decorrer do planejamento, necessita de um tempo crescente para realizar cada iteração.

Em Svenstrup et al. (2011), é proposto um método para atenuar este problema. Em tal método, divide-se, inicialmente, o espaço de trabalho em regiões. Na realização da busca pelo nó, são realizadas buscas apenas nas regiões mais próximas à região na qual a configuração aleatória está contida. Com isso, não é sempre necessário que o algoritmo realize uma busca em todos os nós da árvore. Essa técnica faz com que o algoritmo demande menos tempo de processamento ou que mais vértices sejam adicionados à árvore no mesmo intervalo de tempo, melhorando, portanto, a qualidade da solução.

3.2 Geração de Primitivas de Movimento

No RRT, na etapa de geração das primitivas há também uma desvantagem. As primitivas de movimento são uma discretização do espaço de configuração do robô em um conjunto de estados, conectados entre si (Likhachev e Ferguson; 2009). Cada uma dessas conexões representa uma trajetória alcançável pelo robô, em virtude da utilização das equações cinemáticas e dinâmicas do robô. Logo, as primitivas, apesar de colaborarem com a diminuição da complexidade computacional, limitam a movimentação do robô (Pivtoraiko et al.; 2009).

Baseando-se nesse fato, para sistemas com muitos graus de liberdade, o problema fica ainda maior. Se são geradas muitas primitivas, restringe-se o mínimo possível a movimentação do robô e aumenta-se drasticamente a complexidade computacional, em virtude da necessidade de geração de um maior número de primitivas, análises de colisão e de custo de extensão. Caso sejam geradas poucas primitivas, mesmo que a complexidade do problema seja menor, pode-se limitar a movimentação do robô de tal forma que, mesmo havendo uma solução, o algoritmo não a encontre.

Em Ugur et al. (2012) e Orthey et al. (2013), as primitivas de movimento são geradas *offline* através de técnicas de aprendizado de máquina. Contudo, o treinamento necessário para a geração de um pequeno grupo de primitivas pode levar várias horas. Em Cohen et al. (2011), um algoritmo é proposto para a geração de primitivas multidimensionais, permitindo que as primitivas de movimento sejam utilizadas online. O algoritmo, contudo, visa aplicações em manipuladores robóticos.

Em Jeon et al. (2011), é utilizado o *Shooting Method* para geração de primitivas no algoritmo RRT*. Tal método utiliza o algoritmo da bissecção ou do falso positivo para achar as entradas de controle do sistema e retornar a primitiva que alcança a vizinhança de um determinado nó. Contudo, o algoritmo não lida de forma otimizada com a presença de obstáculos. Caso, por exemplo, expanda-se as primitivas a partir de um nó onde não haja possibilidade de movimento, o algoritmo gerará primitivas em colisão até que um determinado número de iterações seja alcançado, o que consome desnecessariamente tempo computacional.

3.3 Ambientes Dinâmicos

Em geral, os algoritmos tradicionais de planejamento de trajetória são ineficientes ao realizar um planejamento num ambiente dinâmico, necessitando serem combinados com algoritmos específicos para solucionar tais problemas. Um ambiente é considerado dinâmico quando contém obstáculos (pessoas, animais, veículos, etc.) que podem se mover ou estão em movimento, ou quando o estado final desejado para o sistema está em movimento (Gopikrishna e Huber; 2009). Fornecer boas trajetórias em um ambiente dinâmico e em tempo real não é simples. Caso o tempo de planejamento e resposta não sejam rápidos o suficiente, poderão ocorrer colisões (Narayanan et al.; 2012).

Os algoritmos de planejamento de trajetórias em ambientes dinâmicos, em comparação aos de ambiente estáticos, necessitam, em geral, de um maior poder computacional para gerar uma solução adequada em um curto espaço de tempo (Phillips e Likhachev; 2011). Isto está no fato de que o robô necessita realizar uma constante avaliação em tempo-real do ambiente e utilizar tais informações para realizar um replanejamento de trajetória,

caso necessário, a fim de evitar colisões com obstáculos que tenham mudado de posição após o planejamento.

Faz-se necessário, também, planejar trajetórias, aumentando consideravelmente o espaço de busca explorado pelo robô (Phillips e Likhachev; 2011). O tempo pode ser adicionado como uma dimensão irreversível do espaço de configurações do robô, aumentando o número de configurações possíveis a serem alcançadas, bem como o espaço de busca por soluções (Gopikrishna e Huber; 2009). Isto gera uma busca lenta, visto que há um número ilimitado de espaços no tempo para realizar tal busca (Narayanan et al.; 2012), (Kushleyev e Likhachev; 2009). Assim quando tal solução for encontrada, esta poderá estar imprópria para a execução (Phillips e Likhachev; 2011).

Uma solução para lidar com os ambientes dinâmicos, com alta velocidade de execução, é tratar o ambiente como estático e incorporar o comportamento reativo ao robô. Assim, o robô planeja naturalmente a trajetória até o local desejado e, caso encontre um obstáculo fora do local esperado, utiliza o comportamento reativo para evitar a colisão. Após o desvio, o robô continua a executar a trajetória previamente gerada ou realiza um novo planejamento. Contudo, tal método, embora apresente uma boa eficiência computacional, perde na qualidade das trajetórias (Narayanan et al.; 2012), (Phillips e Likhachev; 2011) podendo, ainda, cair em mínimos locais (Fulgenzi et al.; 2008), (Petti e Fraichard; 2005), e estados inevitáveis de colisão (ICS) (Petti e Fraichard; 2005).

Considerando tais problemas, diversos métodos foram propostos para aplicações em tempo real e obtenção de solução ótima. Em Jianming et al. (2010), por exemplo, realiza-se o desvio de obstáculos dinâmicos em ambientes incertos através da combinação do Algoritmo Histograma de Campo Potencial com o Algoritmo de Predição de Kalman. O método proposto apresentou vantagens como: menor quantidade de cálculos, melhor capacidade em tempo-real e acurácia na predição da posição do obstáculo no próximo instante.

Em Phillips e Likhachev (2011), é apresentado um método para tratar do planejamento de trajetórias e desvio de obstáculos em ambientes dinâmicos. O algoritmo proposto realiza o desvio de obstáculos dinâmicos em tempo real, mantendo uma solução ótima. O trabalho introduz o conceito de planejamento de trajetórias baseado em intervalo de se-

gurança (SIPP), que se refere ao intervalo de tempo para uma configuração sem colisões e que, se estendido por mais um passo de tempo, resultará em colisão. Baseando-se neste conceito, o planejador constrói um espaço de busca que possui apenas algumas configurações válidas disponíveis. O planejamento no ambiente dinâmico é realizado utilizando-se configurações que estejam dentro do intervalo de segurança. Experimentos com o algoritmo foram realizados com sucesso em ambientes com 200 objetos dinâmicos com tamanho e ponto de início aleatórios.

Em Narayanan et al. (2012), o algoritmo SIPP é combinado com o algoritmo ARA*, formando o algoritmo *Anytime* SIPP. Tal algoritmo consegue melhorar a solução no decorrer do tempo de execução, podendo alcançar a solução ótima. Além disso, o algoritmo proposto fornece garantias quanto à completude do algoritmo e da qualidade da solução.

Os algoritmos SIPP e *Anytime* SIPP, no entanto, consideram que o robô pode acelerar ou parar instantaneamente, negligenciando as restrições dinâmicas do robô. Isso consiste em uma grande desvantagem, visto que, na maioria dos casos, desprezar tais restrições, especialmente em ambientes dinâmicos, pode levar a colisões.

Uma outra abordagem consiste em rodar o algoritmo de planejamento a cada iteração, atualizando, assim, o plano quando o ambiente sofre alguma alteração. Contudo, caso as alterações no ambiente sejam pequenas, gerar um novo planejamento para o problema torna-se custoso, visto que a solução anterior obtida poderia ser ainda válida. Com isso, tal abordagem é considerada ineficiente (Zucker et al.; 2007).

Outro método para planejamento de trajetórias em ambientes dinâmicos é apresentado em Kushleyev e Likhachev (2009). No trabalho os autores estimam as posições futuras e incertezas dos obstáculos em movimento através de técnicas, como o Filtro de Kalman Entendido. Obstáculos que estão a uma certa distância do robô não são levados em conta até que se aproximem. Em seguida gera-se uma malha de estados (Likhachev e Ferguson; 2009) e utiliza-se o algoritmo *Weighted A** para achar trajetórias de boa qualidade que não apresentam colisão com os obstáculos.

Em Shan et al. (2009), é proposta uma extensão do método RRT para ambientes dinâmicos, considerando-se as equações cinemáticas e dinâmicas do sistema. O algoritmo proposto inicialmente constrói uma RRT. Quando algum obstáculo em movimento

encontra-se em colisão com um ou mais ramos da árvore, tais ramos são excluídos e a busca segue com os demais ramos. Para aumentar a velocidade do algoritmo, são expandidas duas árvores: uma a partir da configuração inicial e outra a partir da configuração final. Quando as duas se encontram, é retornada uma solução. Apesar de simples, tal abordagem sofre com a obtenção de trajetórias sub-ótimas em relação ao tempo, visto que em alguns casos, basta que o robô atravessasse a região de colisão de forma segura apenas aumentando a velocidade atual ou, então, espere o obstáculo passar para continuar seguindo a trajetória, sem a necessidade de replanejamento (Phillips e Likhachev; 2011).

Em Zucker et al. (2007), é introduzida uma extensão do algoritmo RRT, chamada MP-RRT, para a solução de problemas relacionados a ambientes dinâmicos. No algoritmo, inicialmente é gerada uma RRT. Em seguida, caso o estado inicial ou o estado final mude substancialmente, é gerada uma nova RRT. Caso contrário, são identificados os trechos da árvore que apresentam colisões. Estes trechos são excluídos da árvore gerando uma árvore conectada ao nó inicial e outra árvore solta no ambiente. É realizada, então, uma reconexão entre as árvores, aproveitando-se parte já construída da árvore. Tal abordagem, apesar de trazer a vantagem da reutilização de trajetórias já encontradas, sofre também com a geração de trajetórias sub-ótimas no tempo.

Capítulo 4

Metodologia

Na etapa de geração de primitivas do algoritmo RRT, dois métodos podem ser usados. O primeiro é o método tradicional de geração de primitivas. Neste método, amostras uniformemente espaçadas de cada entrada são geradas para formar um determinado número de vetores que serão aplicados no sistema. O segundo método de geração de primitivas é o método aleatório, menos comum que o primeiro. Este gera amostras aleatórias de cada entrada, formando vetores de entradas e aplicando-os ao sistema.

O algoritmo RRT gera as primitivas de movimento em todas as iterações e checa se as primitivas geradas estão em colisão. Logo, reduzir o custo computacional dos métodos implica em reduzir consideravelmente o custo computacional do algoritmo RRT no decorrer de suas iterações.

Dessa forma, a fim de reduzir o custo computacional da geração de primitivas do método tradicional e do aleatório, sem considerável redução da qualidade da primitiva obtida, um algoritmo foi desenvolvido utilizando-se o método de otimização por eliminação.

Os métodos tradicional e aleatório de geração de primitivas e o método otimizado desenvolvidos durante o mestrado são descritos à seguir. O enfoque será voltado para a geração de primitivas no algoritmo RRT, durante o procedimento de extensão da árvore.

4.1 Geração otimizada de primitivas de movimento

O método proposto para geração otimizada de primitivas de movimento possui três etapas: geração inicial de primitivas, identificação de áreas promissoras e eliminação das áreas menos promissoras.

As três etapas são executadas durante um número m de iterações. Quanto maior o número de iterações executadas, maior tende a ser a qualidade da solução obtida.

4.1.1 Geração inicial de primitivas

A primeira etapa tem como objetivo gerar um pequeno número de primitivas, calcular o custo de extensão de cada uma delas e checar se há colisões.

Para isso, considera-se que o sistema possui um vetor de entrada n -dimensional, onde cada entrada u_k , com $k = 1, 2, \dots, n$, pode assumir valores dentro de um intervalo finito conhecido. Em seguida, para cada entrada u_k , três valores v_{k1} , v_{k2} , e v_{k3} são pré-definidos, onde v_{k1} é o respectivo limite inferior, v_{k3} é o limite superior e v_{k2} é um valor situado no meio do intervalo, como mostrado na Figura 4.1.

Figura 4.1: O intervalo de valores que cada entrada pode assumir é dividido inicialmente dois subintervalos e são geradas três amostras espaçadas uniformemente: uma situada no limite superior do intervalo, outra do limite inferior e a última no meio do intervalo.

Como cada entrada do sistema teve três valores distintos pré-definidos, é possível gerar 3^n diferentes vetores de entrada e, conseqüentemente, 3^n primitivas. Percebe-se que, após a geração dos vetores de entrada, cada um dos três valores v_{kx} de cada entrada estará contido em 3^{n-1} vetores de entrada. Como cada vetor de entrada gera uma única primitiva, cada valor v_{kx} está diretamente relacionado a 3^{n-1} diferentes primitivas. Nomeia-se esta relação de relação g .

Em seguida, para cada vetor de entrada é realizada a integração numérica das equações do robô. Como resultado, são obtidas 3^n primitivas de movimento. Calcula-se, então, o custo de extensão das primitivas, como mostrado na Figura 4.3, e checa-se quais primitivas geram colisão. Tais dados são utilizados como base para a identificação das áreas promissoras.

Figura 4.2: Primitivas geradas na primeira iteração do algoritmo otimizado.

4.1.2 Identificação de áreas promissoras

A segunda etapa visa identificar áreas promissoras para extensão de novas primitivas, caso necessário, e determinar em que áreas não é necessário mais gerar primitivas. Para cada entrada, são identificados os subintervalos que levam à geração de primitivas livres de colisão e de baixo custo de extensão.

As áreas promissoras são regiões onde se considera que há maiores chances de se encontrar a primitiva de menor custo livre de colisão. Analogamente, as áreas não-promissoras ou pouco promissoras são aquelas em que as chances de se encontrar a primitiva de menor custo e livre de colisão são pequenas.

Para determinar as áreas promissoras de cada entrada, é realizada uma seleção mediante a quatro critérios. O primeiro critério, de caráter eliminatório, consiste em que apenas intervalos de entradas que possam gerar pelo menos uma primitiva livre de colisão podem ser considerados para a avaliação. Assim, se nenhum dos dois subintervalos de uma entrada geraram primitivas livres de colisão, significa que todas as primitivas geradas causaram colisão. Neste caso, o algoritmo considera que não há possibilidade de gerar primitivas livres de colisão.

Por outro lado, se ambos intervalos produzem pelo menos uma primitiva livre de colisão, utiliza-se o segundo critério, onde o melhor intervalo é determinado através do custo de extensão. O intervalo que possuir o menor custo de extensão, baseando-se em uma função custo, é selecionado como mais promissor.

Nos casos em que os custos de extensão forem iguais, o terceiro critério estipula que o intervalo que possuir o maior número de primitivas livres de colisão deve ser selecionado. Isso é feito porque considera-se que as chances da primitiva de menor custo de extensão e livre de colisão estar contida nesse intervalo são maiores.

Por último, caso o número de primitivas livre de colisão seja o mesmo, utiliza-se o quarto critério, que escolhe o intervalo aleatoriamente.

Dessa forma, o algoritmo seleciona, caso existam, os subintervalos das entradas que tendem a gerar primitivas de maior qualidade e livre de colisão.

4.1.3 Eliminação de áreas não-promissoras

A terceira etapa elimina intervalos e primitivas que levam a áreas não-promissoras utilizando os dados gerados na segunda etapa. Nos casos que nenhum intervalo gere uma primitiva livre de colisão, essa ausência é retornada para o algoritmo RRT.

Cada entrada do sistema possui dentro dos intervalos selecionados v_{k2} e, ou v_{k1} ou v_{k3} . Todas as primitivas g -relacionadas dos valores não-selecionados e os respectivos intervalos são eliminados, como mostrado nas Figura 4.3.

Figura 4.3: Eliminação de área não-promissora.

O algoritmo proposto não gera nenhuma primitiva que já havia sido gerada. Assim, após a eliminação de áreas não-promissoras, se o número limite de iterações m ainda não

tiver sido alcançado, novas primitivas são calculadas, como mostrado nas Figuras 4.4 e 4.5, e as áreas promissora são novamente eliminadas. O algoritmo, antes de calcular as primitivas, verifica que há primitivas que não foram eliminadas, não necessitando, portanto calculá-las novamente. Assim, o algoritmo reduz as perdas de custo computacional ao calcular primitivas previamente calculadas.

Figura 4.4: Uma nova amostra é feita no meio do intervalo remanescente da entrada.

Figura 4.5: Uma nova primitiva é gerada.

Se o limite m de iterações é alcançado, retorna-se para o RRT a primitiva livre de colisão de menor custo de extensão, baseando-se em uma função custo, e suas entradas de controle dentre as primitivas restantes, como mostrado na Figura 4.6. Enquanto o número máximo de iterações não é alcançado, as três etapas são reexecutadas.

4.2 Comparação entre o método tradicional e o método otimizado

Ao comparar-se a geração otimizada e tradicional de primitivas, (a partir de agora GOP e GTP, respectivamente), é possível identificar diferenças-chave.

A primeira consiste no fato de o algoritmo GOP gerar menos primitivas que GTP para valores onde $m \geq 2$. Com isso, reduz-se o custo computacional com a geração de

Figura 4.6: A primitiva livre de colisão de menor custo é retornada ao algoritmo RRT.

primitivas, cálculo de custo de extensão e testes de colisão.

Para comparar ambos métodos, as amostras geradas em cada entrada por ambos devem estar na mesma resolução. No GTP, quanto maior o número de amostras de cada entrada, maior a resolução das primitivas geradas. No GOP, o aumento da resolução está diretamente ligado ao número de iterações m , visto que, ao longo do número de iterações, novas amostras v_k vão sendo feitas para cada entrada.

Nas Figuras 4.7(a)-4.7(b) é possível perceber melhor o conceito. A Figura 4.7(a) mostra o GOP em sua primeira iteração ($m = 1$). Considera-se que o intervalo da entrada possui comprimento R . Quando são geradas as três amostras igualmente espaçadas, o espaçamento entre duas amostras consecutivas é $0,5R$. Na segunda iteração do GOP, após a eliminação de um intervalo, uma nova amostra é realizada no meio do intervalo remanescente. Assim, a distância entre duas amostras consecutivas passa a ser $0,25R$, como mostrado na Figura 4.7(b). Assim, quanto mais próximas estão duas amostras da entrada consecutivas, maior é a resolução. Na Figura 4.7(c), é mostrado o número de amostras geradas pelo GTP à mesma resolução que o GOP para $m = 2$.

Para o cálculo da resolução, utiliza-se

$$r = \frac{C_a}{C_t} \quad (4.1)$$

Onde r é a resolução, C_a é a menor distância entre duas amostras consecutivas de uma dada entrada e C_t é o comprimento do intervalo.

Figura 4.7: (a) Amostras realizadas em uma entrada do algoritmo GOP para $m = 1$, com resolução $r = 0,5$. (b) Amostras realizadas em uma entrada do algoritmo GOP para $m = 2$, com resolução $r = 0,25$. (c) Amostras realizadas em uma entrada do algoritmo GTP com resolução $r = 0,25$.

Assim, qual deve ser o número de amostras que o GTP deve gerar em cada entrada para alcançar a mesma resolução que o GOP para um determinado número de iterações m ? Primeiro, considera-se que todas as entradas estejam à mesma resolução. Em seguida, podemos verificar que para $m = 1$, $m = 2$, $m = 3$ o GTP precisa gerar, respectivamente, 3, 5 e 9 amostras por entrada. Supõe-se, assim, que a relação entre o número de amostras por entrada v em função de m seja descrita por

$$v = 2^m + 1 \quad (4.2)$$

Sabe-se que, se para um dado valor de m , se GTP gerou ao todo k amostras, então para $m + 1$, o GTP necessita gerar outras $k - 1$ amostras. Demonstra-se por indução que a Equação (4.2) representa a relação desejada. Fazendo-se então A_{m+1} o número de amostras geradas pelo GTP para $m + 1$ iterações do GOP, $A_m = v$ e N_a o número de novas amostras que devem ser geradas ao se realizar uma nova iteração do GOP, então

$$A_{m+1} = A_m + N_a \quad (4.3)$$

Substituindo-se N_a por A_m e utilizando-se a relação descrita na Equação (4.2), tem-se

$$2^{m+1} + 1 = 2^m + 1 + (2^m + 1) - 1 \quad (4.4)$$

Como a relação mostrada na Equação (4.4) é verdadeira, a relação entre o número de amostras que devem ser geradas no GTP em relação ao número de iterações m do GOP é dada pela Equação (4.2).

Assim, se o sistema possui n entradas e para cada entrada gerou-se o mesmo número de amostras na mesma resolução, obtém-se por uma simples permutação o número de primitivas P_{trad} geradas no GTP para um dado valor de m , dado por

$$P_{trad} = (2^m + 1)^n \quad (4.5)$$

Uma das desvantagens do GOP está no fato de que se todas as primitivas geradas na primeira iteração gerarem colisão, o algoritmo entende que não há possibilidade de gerar primitivas livres de colisão. Como na primeira iteração a resolução obtida é mínima, é possível que se mais primitivas fossem geradas, uma delas estivesse livre de colisão. As chances de tal fato acontecer, especialmente em casos onde o tempo de integração do sistema é curto, são pequenas, visto que as primitivas geradas tendem a estar geometricamente bem próximas.

Contudo, nos casos em que não haja possibilidade de gerar primitivas livres de colisão, o GOP gerará apenas algumas primitivas, enquanto o GTP gerará todas. Como a geração de primitivas implica no cálculo de custo de extensão e na checagem de colisões, o GOP acaba reduzindo o custo computacional em relação ao GTP.

4.3 Comparação entre o método aleatório e o método otimizado

Ao comparar-se os procedimentos otimizado e aleatório (GAP) algumas diferenças são notadas. A primeira consiste no fato de que o GAP não possui restrições quanto

à resolução das primitivas geradas, diferentemente do GOP. Assim, o GAP pode gerar primitivas com custos de extensão inferiores ao GOP. Contudo, pelo fato da geração de primitivas ser aleatória no primeiro, não há garantia de que esse fato ocorrerá.

Outro aspecto é que o GOP busca áreas promissoras para a geração de primitivas, possibilitando a geração de primitivas de menor custo de extensão ao longo da execução. O GAP, contudo, além de não realizar tal busca, não garante que sejam geradas primitivas em áreas promissoras, podendo tanto gerar todas primitivas em uma área promissora como todas fora dela. Assim, como as primitivas não são geradas de forma uniforme, durante a extensão das primitivas, o robô pode ter sua movimentação restringida ou direcionada apenas para uma região.

Capítulo 5

Modelo matemático do veículo

No planejamento cinemático-dinâmico de trajetórias de robôs móveis autônomos, o modelo base para o controlador é de fundamental importância. Como mencionado anteriormente, caso o modelo adotado não incorpore as características e limitações dinâmicas e cinemáticas do robô, a trajetória gerada não poderá ser seguida.

Caso o modelo não represente adequadamente o robô, além da geração de trajetórias não alcançáveis pelo robô, poderão ocorrer colisões com obstáculos. Ademais, as estimativas do robô quanto aos seus estados em um tempo futuro poderão apresentar um erro muito grande, prejudicando o planejamento em ambientes dinâmicos e ocasionando uma maior complexidade computacional no algoritmo em virtude da maior necessidade de correção de erros pelo controlador (Pepy et al.; 2006). As equações cinemáticas e dinâmicas do veículo serão utilizadas para a geração das primitivas de movimento no planejamento de trajetórias utilizando o algoritmo RRT otimizado.

Para o presente trabalho, escolheu-se utilizar o modelo cinemático-dinâmico de um veículo. Vários trabalhos propõem modelos cinemáticos e dinâmicos para veículos, dentre os quais destacam-se aqui os modelos propostos em Moret (2003), Pepy et al. (2006), Choset et al. (2005), Macek et al. (2007), Chen e Jiao (2011), Stotsky e Hu (1998) e Li e Trentini (2010). Neste capítulo, será realizada a derivação do modelo matemático do veículo de acordo com o raciocínio realizado em Moret (2003). Será apresentada, primeiramente, o modelo cinemático e, em seguida, o modelo dinâmico do veículo.

5.1 Modelo cinemático

Para iniciar o desenvolvimento das equações matemáticas, é preciso obter a equação que representa restrição não-holonômica de movimento do automóvel. Para ângulos pequenos do esterçamento é possível modelar o veículo como uma bicicleta, formando-se, assim, um modelo simplificado do automóvel. Em tal modelo, é realizada a projeção das rodas dianteiras e traseiras no centro de cada eixo. São formadas, assim, duas rodas virtuais. Tal simplificação é utilizada em diversos trabalhos, como, por exemplo, Pepy et al. (2006), Egerstedt et al. (1998), Chen e Jiao (2011), Tsai et al. (2004) e Economou et al. (2003).

Figura 5.1: Modelo de bicicleta para robôs em forma de carro apresentado em dois sistemas de coordenadas: (x, y) e (u, w) . O sistema de coordenadas (u, w) é fixo no carro. Fonte: Moret (2003)

Sabe-se que um automóvel não pode mover-se no eixo perpendicular as rodas, ou seja, na direção w do sistema de coordenadas mostrado na Figura 5.1(b). Sendo $\dot{u}$ a velocidade do veículo no eixo u , $\dot{w}$ a velocidade do veículo no eixo w e θ é o ângulo da roda traseira com respeito ao eixo x (pertencente ao eixo de coordenadas mostrado na Figura 5.1(a) e considerando-se (x, y) a posição do centro de gravidade (representado por CG na Figura 5.1(a), (x_1, y_1) como a posição do eixo da roda traseira e (x_2, y_2) o da dianteira, tem-se que

$$x_1 = x - b \cos(\theta) \quad x_2 = x + a \cos(\theta) \quad (5.1)$$

$$y_1 = y - b \sin(\theta) \quad y_2 = y + a \sin(\theta) \quad (5.2)$$

Derivando-se no tempo as Equações (5.1) e (5.2), tem-se

$$\dot{x}_1 = \dot{x} + b\dot{\theta}\sin(\theta) \quad \dot{x}_2 = \dot{x} - a\dot{\theta}\sin(\theta) \quad (5.3)$$

$$\dot{y}_1 = \dot{y} - b\dot{\theta}\cos(\theta) \quad \dot{y}_2 = \dot{y} + a\dot{\theta}\cos(\theta) \quad (5.4)$$

Escreve-se então a equação não-holonômica de cada uma das rodas

$$\dot{x}_1 \sin(\theta) - \dot{y}_1 \cos(\theta) = 0 \quad (5.5)$$

$$\dot{x}_2 \sin(\theta + \phi) - \dot{y}_2 \cos(\theta + \phi) = 0 \quad (5.6)$$

Substituindo-se as Equações (5.3) e (5.4) nas Equações (5.5) e (5.6), tem-se

$$\dot{x} \sin(\theta) - \dot{y} \cos(\theta) + b\dot{\theta} = 0 \quad (5.7)$$

$$\dot{x} \sin(\theta + \phi) - \dot{y} \cos(\theta + \phi) - a\dot{\theta} \cos \phi = 0 \quad (5.8)$$

Realizando-se a transformação das coordenadas no eixo (u, w) para (x, y) , obtém-se

$$\dot{x} = \dot{u} \cos \theta - \dot{w} \sin \theta \quad (5.9)$$

$$\dot{y} = \dot{u} \sin \theta + \dot{w} \cos \theta \quad (5.10)$$

Substituindo-se as Equações (5.9) e (5.10) em (5.7), obtém-se

$$\dot{w} = b\dot{\theta} \quad (5.11)$$

Substituindo-se as Equações (5.9) e (5.10) em (5.8), obtém-se

$$\dot{u} \sin \phi - \dot{w} \cos \phi - a\dot{\theta} \cos \phi = 0 \quad (5.12)$$

Substituindo-se a Equação (5.11) na Equação (5.12) e multiplicando-se o resultado obtido por $\sec \phi$, obtém-se o valor de $\dot{\theta}$, dado por

$$\dot{\theta} = \frac{\tan \phi}{l} \dot{u} \quad (5.13)$$

Derivando-se as Equações (5.12) e (5.13), obtém-se

$$\ddot{w} = \ddot{\theta}b \quad (5.14)$$

$$\ddot{\theta} = \frac{\tan \phi}{l} \ddot{u} + \frac{\dot{u} \dot{\phi} \sec^2 \phi}{l} \quad (5.15)$$

As Equações (5.14) e (5.15) serão utilizadas posteriormente na obtenção do modelo dinâmico do sistema.

5.2 Modelo dinâmico

Para a obtenção do modelo dinâmico, são feitas as seguintes suposições (Moret; 2003):

- não há forças de fricção entre a roda e o veículo;
- as rodas traseiras estão na mesma orientação do veículo;
- não há deslizamento nas rodas;
- a força de condução pode ser modelada como agindo na roda virtual traseira.

Na Figura 5.2, são mostradas as forças resultantes e entradas do veículo. A força F_D representa a força aplicada na roda traseira do carro, no eixo u . A força F_R representa a força resultante lateral que atua na roda traseira no eixo w . A força F_F representa a força resultante lateral na roda dianteira.

Figura 5.2: Forças resultantes nas rodas do veículo. Imagem extraída de Moret (2003)

Dessa forma, considerando-se m_s e J a massa e o momento de inércia, respectivamente, obtém-se as seguintes equações dinâmicas

$$\ddot{u} = \dot{u}\dot{\theta} - \frac{F_F \sin \phi}{m_s} + \frac{F_D}{m_s} \quad (5.16)$$

$$\ddot{w} = -\dot{u}\dot{\theta} + \frac{F_F \cos \phi}{m_s} + \frac{F_R}{m_s} \quad (5.17)$$

$$\ddot{\theta} = \frac{aF_F \cos \phi}{J} - \frac{bF_R}{J} \quad (5.18)$$

Isolando-se a variável F_R em (5.17), obtém-se

$$F_R = \frac{aF_F \cos \phi - J\ddot{\theta}}{b} \quad (5.19)$$

Substituindo-se as Equações (5.14) e (5.19) na Equação (5.17), tem-se

$$\ddot{\theta}b = -\dot{u}\dot{\theta} + \frac{lF_F \cos \phi - J\ddot{\theta}}{bm_s} \quad (5.20)$$

Isolando-se a variável F_F em (5.20) tem-se

$$F_F = \frac{(b^2m_s + J)\ddot{\theta} + bm_s\dot{u}\dot{\theta}}{l \cos \phi} \quad (5.21)$$

Substituindo-se as Equações (5.11), (5.13), (5.15), e (5.21) na Equação (5.16), obtém-se

$$\ddot{u} = \frac{l^2 \cos^2 \phi F_D - \dot{u}\dot{\phi}(b^2m_s + J) \tan \phi}{l^2m_s \cos^2 \phi + (b^2m_s + J) \sin^2 \phi} \quad (5.22)$$

Substituindo-se as Equações (5.11) e (5.13) nas Equações (5.9) e (5.10), obtém-se

$$\dot{x} = \left(\cos \theta - \frac{b \tan \phi \sin \theta}{l} \right) \dot{u} \quad (5.23)$$

$$\dot{y} = \left(\sin \theta + \frac{b \tan \phi \cos \theta}{l} \right) \dot{u} \quad (5.24)$$

Assume-se que o giro do volante é realizado por um servomotor, representado por um sistema linear de primeira ordem dado por

$$\dot{\phi} = \frac{\phi}{\tau_s} + c_s v_1 \quad (5.25)$$

Onde v_1 é a tensão aplicada no servomotor de giro do volante e τ_s e c_s são constantes.

Define-se, então, os estados do sistema como

$$X = \begin{bmatrix} x & y & \theta & \dot{u} & \phi \end{bmatrix}^t$$

As entradas do sistema são definidas como

$$U = \begin{bmatrix} F_D & v_1 \end{bmatrix}^t$$

Assim, as equações de estado do sistema podem ser escritas por

$$\dot{x} = \left(\cos \theta - \frac{b \tan \phi \sin \theta}{l} \right) \dot{u} \quad (5.26)$$

$$\dot{y} = \left(\sin \theta + \frac{b \tan \phi \cos \theta}{l} \right) \dot{u} \quad (5.27)$$

$$\dot{\theta} = \frac{\tan \phi}{l} \dot{u} \quad (5.28)$$

$$\ddot{u} = \frac{l^2 \cos^2 \phi F_D - \dot{u} \dot{\phi} (b^2 m_s + J) \tan \phi}{l^2 m_s \cos^2 \phi + (b^2 m_s + J) \sin^2 \phi} \quad (5.29)$$

$$\dot{\phi} = \frac{\phi}{\tau_s} + c_s v_1 \quad (5.30)$$

5.3 Método de Runge-Kutta de Quarta Ordem

O método de Runge-Kutta é um método numérico utilizado para aproximação de equações diferenciais ordinárias. Tal método será importante ao calcular as primitivas de movimento a partir da integração das equações cinemáticas e dinâmicas do veículo durante a execução do algoritmo RRT. Com frequência, utiliza-se o método de Runge-Kutta de quarta ordem, dado por:

$$\mathbf{y}_{a+1} = \mathbf{y}_a + h \frac{(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4)}{6} \quad (5.31)$$

onde,

$$\begin{aligned}\mathbf{k}_1 &= \mathbf{f}(t_a, \mathbf{y}_a) \\ \mathbf{k}_2 &= \mathbf{f}(t_a + 0.5h, \mathbf{y}_a + 0.5h\mathbf{k}_1) \\ \mathbf{k}_3 &= \mathbf{f}(t_a + 0.5h, \mathbf{y}_a + 0.5h\mathbf{k}_2) \\ \mathbf{k}_4 &= \mathbf{f}(t_a + h, \mathbf{y}_a + h\mathbf{k}_3)\end{aligned}\tag{5.32}$$

A variável h representa o passo dado no sistema. O erro por passo gerado pelo algoritmo Runge-Kutta de Quarta Ordem é da ordem de h^5 .

Capítulo 6

Resultados

Após a obtenção de um método otimizado de geração de primitivas, foram realizadas três simulações. A primeira comparou os custos computacionais e de extensão dos métodos de geração de primitivas tradicional (GTP), aleatório (GAP) e otimizado (GOP). Nesta simulação o algoritmo RRT não foi utilizado, apenas as funções de extensão.

Na segunda simulação, os métodos tradicional e otimizado são executados dentro do algoritmo RRT, a fim de avaliar o custo computacional do algoritmo e a qualidade da solução global obtida após o planejamento.

Por fim, foi realizado um estudo de caso, aplicando o algoritmo RRT-O em um ambiente com objetos dinâmicos onde é necessário um replanejamento de trajetória.

Os resultados foram obtidos a partir de simulações em um computador com processador Core i7, 3.40GHz e 8Gb de memória RAM.

6.1 Otimização da geração de primitivas de movimento

As simulações contidas nesta seção foram realizadas utilizando o software Matlab®. As simulações tiveram como objetivo fundamental mostrar as diferenças de custo computacional entre os métodos GOP, GAP e GTP, bem como de custo de extensão.

Nesta seção, os métodos são avaliados separadamente do algoritmo RRT. Os três algoritmos têm como função a geração das primitivas e retornar a primitiva de menor

custo de extensão e livre de colisão. Como não há, neste caso, planejamento de trajetória, apenas a extensão das primitivas, trataremos aqui como solução a melhor primitiva livre de colisão encontrada por cada um dos métodos.

6.1.1 O ambiente de simulação

Foi desenvolvido um ambiente com três obstáculos circulares (c_1 , c_2 e c_3) dispostos de forma aleatória em uma área de dimensões $10m \times 10m$. Os obstáculos c_1 , c_2 e c_3 possuíam, respectivamente, raio de $1.5m$, $3m$ e $2m$, e c_1 e c_2 eram tangentes. Todos os obstáculos eram estáticos e o robô possuía total conhecimento do ambiente. Na Figura 6.1 é possível ver uma ilustração de um semelhante aos utilizados na simulação. Em preto são mostrados os obstáculos, em azul o robô e em vermelho o destino.

Figura 6.1: Imagem de um ambiente gerado aleatoriamente.

A função de custo utilizada para calcular o custo da primitiva de movimento em relação a uma amostra é dada por

$$f_{custo} = ((D_{euclidiana})^2 + (Kc\Delta\theta)^2)^{0.5} \quad (6.1)$$

Onde $D_{euclidiana}$, dada em metros, é a distância euclidiana entre o estado final da primitiva e a amostra, $\Delta\theta$, dado em radianos, é a diferença de orientação entre estas amostras e K_c é uma constante de valor igual a $1m$.

6.1.2 Modelo do robô

Como modelo do robô, utilizou-se as Equações (5.26), (5.27) e (5.28), com o vetor de estados dado por $X = [x \ y \ \theta]^t$ e o vetor de entradas por $U = [\dot{u} \ \phi]^t$. Para representar o robô, foi utilizada a expansão proposta em (Latombe; 1991) de expansão dos obstáculos. Assim, foi possível representar o robô por um ponto, situado em seu centro de gravidade. Para realizar a integração do modelo, foi utilizado o método de Runge-Kutta de quarta ordem (RK4), mostrado na Equação (5.32), com passo de integração de 0.1s.

Considerou-se também que $b = 0,9m$, $l = 1,8m$, e que o robô poderia assumir velocidades entre -30 e $+30m/s$ e o ângulo de esterçamento poderia assumir valores entre -40° e $+40^\circ$.

6.1.3 Comparação entre os custos computacionais

Para realizar uma comparação entre os custos computacionais dos métodos, foram analisadas duas situações. Na primeira, o robô foi situado em posições onde todas as primitivas geradas estavam livres de colisão, com o intuito de garantir que o algoritmo otimizado obtivesse uma solução para o problema. Na segunda situação, o robô foi colocado em uma posição onde todas as primitivas geradas causavam colisão, a fim de avaliar a performance dos três métodos em tais casos.

Na Figura 6.2(a) e na Tabela 6.1, é mostrada a variação do custo computacional médio dos três métodos ao longo do número m de iterações do algoritmo GOP, em um ambiente onde todas as primitivas geradas estavam livres de colisões. Na Tabela 6.2, são mostrados os desvios padrão obtidos. Nas simulações, o valor do limite de iterações do algoritmo otimizado foi estipulado variando de 1 a 5. Para cada valor de m , foram realizadas 100 simulações com cada um dos três métodos. Assim, durante as 500 simulações, a cada 100 simulações a resolução das primitivas geradas é aumentada. Percebe-se que são gerados 5 grupos para simulação, cada um correspondente a um valor de m . O ambiente e o estado do robô mantém-se os mesmos ao longo das simulações.

Para cada grupo, o número de primitivas geradas pelos método tradicional e pelo método aleatório é configurado apropriadamente, objetivando, no método tradicional,

Figura 6.2: Gráfico do aumento do tempo computacional em função do número de iterações m realizadas, comparando-se os algoritmos GTP, GOP e GAP. (a) Gráfico para as situações onde todas primitivas geradas estavam livres de colisões. (b) Gráfico para situações onde todas as primitivas geradas causam colisão.

Tabela 6.1: Tabela com o custos computacionais médios, em milissegundos, de cada um dos três métodos analisados, em função de m . Tais valores foram obtidos em ambientes onde todas as primitivas geradas estavam livre de colisão.

Método	m = 1	m = 2	m = 3	m = 4	m = 5
Tradicional	35,55	87,25	272,8	969	3.718
Aleatório	26,78	73,45	239	863,3	3.409
Otimizado	44,46	74,91	106,4	139,4	175,1

Tabela 6.2: Tabela com desvios padrão em milissegundos obtidos com a simulação para os casos onde todas primitivas estavam livres de colisão.

Método	$m = 1$	$m = 2$	$m = 3$	$m = 4$	$m = 5$
Tradicional	3,3	1,3	2,0	5,8	32,8
Aleatório	0,3	0,7	1,8	2,5	5,3
Otimizado	1,2	1,0	1,1	2,6	1,6

gerar primitivas na mesma resolução do método otimizado e, no caso do método aleatório, gerar o mesmo número de primitivas que o método tradicional.

É possível perceber na Tabela 6.1, que, quando $m = 1$, o algoritmo GOP mostrou-se cerca de 1,25 e 1,66 vezes mais custoso computacionalmente que os métodos GTP e GAP, respectivamente. Quando $m = 2$, o algoritmo GOP mostra-se cerca de 14% menos custoso computacionalmente que o GTP e apenas cerca de 2% mais custoso que o GAP.

Para $m = 3$, o método GOP já mostra-se menos custoso computacionalmente que o GAP e o GTP. Percebe-se também que, quanto maior o valor de m , maior tende a ser a diferença entre os custos computacionais do algoritmo GOP com aqueles dos algoritmos GTP e GAP. Com $m = 5$, os algoritmos GTP e GAP mostraram-se aproximadamente 21,3 e 19,5 vezes mais custosos que o GOP.

Estes dados revelam que a variação do custo computacional dos algoritmos GTP e GAP ao longo de m é bem maior que a do GOP. Comparando-se a variação dos custos computacionais com $m = 1$ para $m = 5$, percebe-se que enquanto o número de amostras por entrada tornou-se 11 vezes maior, o custo computacional dos algoritmos GTP e GAP se tornou aproximadamente 104,6 e 127,3 vezes maior, respectivamente. O método otimizado, no entanto, tornou-se apenas 3,95 vezes maior.

Na Figura 6.2(b) e na Tabela 6.3, é mostrada a variação do custo computacional médio dos três métodos ao longo do número m de iterações, em um ambiente onde todas as primitivas geradas causavam colisões. Neste caso, foram realizadas outras 500 simulações, mudando apenas a configuração inicial do robô para uma onde todas as primitivas geradas causam colisão. Na Tabela 6.4, são mostrados os desvios padrão referentes à simulação.

Percebe-se que o custo computacional do algoritmo GOP permanece praticamente constante, enquanto os algoritmos GTP e GAP possuem o custo computacional crescente. Esta diferença ocorre porque o algoritmo GOP gera apenas algumas primitivas em

sua primeira iteração. Ao perceber que todas geram colisão, considera que não há possibilidade de gerar primitivas sem colisão, não dando continuidade às iterações seguintes e retornando a ausência de uma primitiva livre de colisão. Os métodos GTP e GAP, por outro lado, geram todas as primitivas e verificam se geram colisão.

No caso em que $m = 5$, onde os algoritmos GTP e GAP mostraram-se, respectivamente, aproximadamente 110,65 e 111,03 vezes mais custosos computacionalmente que o algoritmo GOP. Neste caso, enquanto os algoritmos GTP e GAP geram 1089 primitivas e verificam que todas estão causando colisão, o algoritmo GOP gera apenas 9 primitivas para chegar à mesma conclusão.

Tabela 6.3: Tabela com o custos computacionais médios, em milissegundos, de cada um dos três métodos analisados, em função de m . Tais valores foram obtidos em ambientes onde todas as primitivas geradas causavam colisão.

Método	$m = 1$	$m = 2$	$m = 3$	$m = 4$	$m = 5$
Tradicional	32,02	77,41	238,7	856,1	3.199
Aleatório	26,75	73,03	236	848,4	3.210
Otimizado	29,05	28,39	28,61	28,91	28,91

Tabela 6.4: Tabela com desvios padrão em milissegundos obtidos com a simulação para os casos onde todas primitivas geradas colidiam.

Método	$m = 1$	$m = 2$	$m = 3$	$m = 4$	$m = 5$
Tradicional	3,6	3,5	1,5	84,2	29,7
Aleatório	0,4	0,6	1,1	2,5	4,1
Otimizado	0,78	0,33	0,47	0,38	0,38

6.1.4 Análise dos custos de extensão

Simulações foram realizadas a fim de comparar os custos de extensão obtidos pelos algoritmos GTP, GOP e GAP. À mesma resolução, o conjunto de primitivas geradas no algoritmo GOP está contido no conjunto de primitivas geradas pelo método tradicional. Diante disso, à mesma resolução, percebe-se que a primitiva retornada pelo algoritmo GOP possui, no melhor dos casos, custo de extensão igual àquela retornada pelo GTP.

Primeiramente, para comparar os custos de extensão das primitivas retornadas por ambos algoritmos, foram realizadas simulações onde foram gerados 100 ambientes aleatoriamente, conforme descrito na Subseção 6.1.1. Para cada ambiente foram gerados

aleatoriamente 100 estados iniciais e 100 estados finais para o robô. Para estes estados não havia restrições quanto a estarem em colisão ou não.

Em seguida, para cada estado inicial e final, eram expandidas as primitivas de movimento utilizando ambos métodos aqui avaliados. O número limite de resolução estipulado foi para duas iterações $m = 2$ do algoritmo GOP.

Os dados obtidos são mostrados na Tabela 6.5. De 10.000 casos, em 7.981 o algoritmo GTP encontrou uma solução. Deste número, o algoritmo GOP encontrou uma solução em 7.927 casos, e portanto, não encontrou solução em 54 casos, o que corresponde a 0,68% dos casos onde uma solução. Em 7.217 casos, ambos algoritmos encontraram a mesma primitiva como solução, o que corresponde a 90,43% dos casos onde foi encontrada uma solução. Os dois algoritmos encontraram soluções diferentes para o problema em 710 casos (8,90% dos casos onde uma solução foi obtida). Em 2.019 nenhum dos dois métodos conseguiu gerar ao menos uma primitiva livre de colisão.

Nos casos em que ambos algoritmos divergiram na solução encontrada, constatou-se que a média das razões entre o custo de extensão da primitiva retornada pelo algoritmo GTP com o custo obtido pela primitiva retornada pelo algoritmo GOP foi de aproximadamente 0,9995, com um desvio padrão de aproximadamente 0,003. Isso mostra que o algoritmo proposto além de possuir eficiência muito superior que o algoritmo tradicional, possui quase a mesma eficácia.

Tabela 6.5: Comparativo dos algoritmos GTP e GOP para $m = 2$.

GTP x GOP			
Casos analisados	Número de casos	Porcentagem dos casos	Porcentagem em relação aos casos com solução
Casos onde ambos métodos não encontraram solução	2.019	20,19	-
Casos onde apenas o GTP encontrou uma solução	54	0,54	0,68
Casos onde ambos métodos encontraram a mesma solução	7.217	72,17	90,43
Casos onde ambos métodos encontraram soluções diferentes	710	7,1	8,9

O teste foi repetido com o intuito de comparar o algoritmo GOP com o GAP. Os

resultados obtidos são mostrados na Tabela 6.6. Dos 10.000 casos, 7.916 (99,5% dos casos onde uma solução foi encontrada), ambos algoritmos encontraram solução, onde a média das razões entre o custo de extensão obtido com a primitiva retornada pelo algoritmo GOP e o custo da primitiva retornada pelo GAP foi de aproximadamente 0,9975, com um desvio padrão de aproximadamente 0,0047.

Em 7 casos (0,088% dos casos com solução), apenas o algoritmo GOP encontrou solução. Em 33 casos (0,41% dos casos com solução), apenas o algoritmo GAP encontrou solução. Em 2.044 não foram encontradas solução através de ambos métodos.

Vale ressaltar que o algoritmo GAP não possui restrições quanto à resolução das primitivas geradas. O único limitante, neste caso, consiste no número de primitivas geradas. Ou seja, além de o algoritmo GAP gerar mais primitivas que o GOP, ele ainda tem a possibilidade de chegar a resoluções que não são alcançáveis pelo GOP, dado um número limite de iterações. Ainda assim, o algoritmo GOP, com um custo computacional bastante inferior, obteve soluções com qualidade muito próximas às obtidas pelo GAP.

Tabela 6.6: Comparativo dos algoritmos GAP e GOP para $m = 2$.

GAP x GOP			
Casos analisados	Número de casos	Porcentagem dos casos	Porcentagem em relação aos casos com solução
Casos onde ambos métodos não encontraram solução	2.044	20,44	-
Casos onde ambos métodos encontraram soluções	7.916	79,16	99,5
Casos onde apenas o GAP encontrou uma solução	33	0,33	0,41
Casos onde apenas o GOP encontrou uma solução	7	0,07	0,088

O mesmo experimento foi realizado para avaliar os custos de extensão para $m = 5$, comparando o GOP com GAP e GTP. Foram gerados aleatoriamente 100 ambientes e 10 estados iniciais e 10 finais para cada ambiente.

Os resultados da comparação do GOP com o GTP, para $m = 5$, são mostrados na Tabela 6.7. Dos 1.000 casos, em 766 ambos algoritmos encontraram a mesma solução, o que equivale a 93,87% dos casos com solução. Em apenas 4 casos o GOP não encontrou uma primitiva livre de colisão. Neste mesmos 4 casos, o GTP encontrou uma. Nos 46

casos onde ambos métodos encontraram primitivas diferentes, constatou-se que a média das razões entre o custo de extensão da primitiva retornada pelo algoritmo GTP com o custo obtido pela primitiva retornada pelo GOP foi de aproximadamente 0,9997, com um desvio padrão de aproximadamente 0,008.

Tabela 6.7: Comparativo dos algoritmos GTP e GOP para $m = 5$.

GTP x GOP			
Casos analisados	Número de casos	Porcentagem dos casos	Porcentagem em relação aos casos com solução
Casos onde ambos métodos não encontraram solução	184	18,40	-
Casos onde apenas o GTP encontrou uma solução	4	0,4	0,49
Casos onde ambos métodos encontraram a mesma solução	766	76,60	93,87
Casos onde ambos métodos encontraram soluções diferentes	46	4,6	5,64

Esta última simulação foi realizada novamente, contudo, comparando o GOP com o GAP. Os resultados obtidos são mostrados na Tabela 6.8. Dos 1.000 casos, em 780 ambos métodos encontraram uma solução. Em 10 casos apenas o GAP encontrou uma solução, o que equivale a 1,27% dos casos com solução. Nos casos que ambos métodos encontraram soluções, constatou-se que a média das razões entre o custo de extensão obtido com a primitiva retornada pelo algoritmo GOP e o custo da primitiva retornada pelo GAP foi de aproximadamente 0,9996, com um desvio padrão de aproximadamente 0,0008.

Tabela 6.8: Comparativo dos algoritmos GAP e GOP para $m = 5$.

GAP x GOP			
Casos analisados	Número de casos	Porcentagem dos casos	Porcentagem em relação aos casos com solução
Casos onde ambos métodos não encontraram solução	210	21,0	-
Casos onde ambos métodos encontraram soluções	780	78,0	98,73
Casos onde apenas o GAP encontrou uma solução	10	1,0	1,27
Casos onde apenas o GOP encontrou uma solução	0	0,0	0,0

6.2 Otimização do algoritmo RRT

O algoritmo GOP foi aplicado ao algoritmo RRT, formando o algoritmo RRT-O a fim de avaliar o impacto da otimização, considerando o custo computacional e o custo global de navegação, em relação ao RRT básico, chamado aqui de RRT-T. As simulações foram realizadas em um simulador desenvolvido em linguagem python.

6.2.1 O ambiente de simulação

Na Figura 6.3, é possível ver um dos ambientes utilizados para simulação. Os obstáculos, representados em preto, possuem formato retangular e estão estáticos. O robô é representado em verde e o seu destino por um ponto amarelo e a respectiva tolerância em vermelho. A árvore obtida com o algoritmo RRT-O é mostrada em azul. O ambiente possui dimensões de $40m \times 40m$. A função custo utilizada foi a distância euclidiana.

Figura 6.3: Imagem do ambiente de simulação e da árvore obtida através do método RRT-O.

6.2.2 Modelo do robô

O modelo cinemático-dinâmico do robô utilizado nas simulações é composto pelas Equações (5.26), (5.27), (5.28), (5.29), (5.30). O vetor de estados do sistema é dado por $X = [x, y, \theta, \dot{x}, \dot{y}]^t$ e o vetor de entradas por $U = [F_D, v_1]^t$. Considerou-se também que $b = 0,9m$, $l = 2,5m$, $m_s = 1.000kg$, $J = 1.300kgm^2$, $t_s = 1,2$ e $c_s = 1,1$. O torque foi limitado entre $-2,5kNm$ e $2,5kNm$ e a tensão do servomotor do esterçamento em $-5,0V$ e $5,0V$.

As dimensões do robô foram definidas em $2,50 \times 1,70m$. Os ângulos de esterçamento que o robô pode assumir variam de -40° a $+40^\circ$.

Para realizar a integração do modelo do robô, foi utilizado o método RK4 e o tempo de integração foi definido como $0,15s$.

6.2.3 Comparação entre RRT-T e RRT-O

O primeiro passo para comparar ambos algoritmos foi garantir que ambos tivessem o mesmo conjunto de amostras aleatórias em cada simulação. Para isso, antes de cada simulação foram geradas 1.500 amostras aleatórias. Estas amostras aleatórias foram passadas como parâmetros para os algoritmos RRT-T e RRT-O. Percebe-se que o tempo de execução de ambos algoritmos foi limitado em 1.500 amostras. O bias de expansão de ambos algoritmos em direção objetivo utilizado foi de 5%.

Nota-se que ambos algoritmos não geraram uma amostra aleatória a cada iteração do RRT, mas utilizaram sequencialmente as amostras previamente geradas. Assim, os algoritmos estão situados no mesmo ambiente, com as mesmas condições iniciais e com as mesmas amostras do algoritmo RRT. A única diferença está na geração das primitivas de movimento.

Foram realizadas 1.000 simulações e foram geradas para cada uma 1.500 amostras aleatórias. Em cada simulação, as primitivas geradas pelo RRT-T e RRT-O estavam à mesma resolução. O número de iterações limite m da geração de primitivas do RRT-O variou de 1 a 5, com acréscimo de uma unidade a cada duzentas simulações.

Na Figura 6.4, é possível comparar o custo computacional dos algoritmos em relação a m . Para $m = 1$, percebe-se que o RRT-O possui custo computacional levemente superior ao RRT-T. Contudo, para $m \geq 2$, o RRT-O mostra custo computacional inferior.

Outro fator que pode ser observado é que o RRT-O para $m = 5$ possui aproximadamente 60% do custo computacional do RRT-T com $m = 2$, porém, com melhor resolução.

Percebe-se, então, que enquanto o número m de iterações e a resolução das primitivas aumentam, os custos computacionais do método RRT-T mostram um considerável aumento. O RRT-O, ao contrário, mostra um modesto aumento.

Figura 6.4: Gráfico com o tempo computacional em relação ao número m de iterações para os algoritmos RRT-T e RRT-O.

Na Figura 6.5, é possível acompanhar as variações do custo computacional em relação ao número de amostras aleatórias que os algoritmos RRT-T e RRT-O necessitaram para encontrar ou retornar a ausência de uma solução.

É possível notar que o RRT-T, para $m = 3, 4, 5$, possui um maior aumento de custo computacional quando o número de amostras aumenta, quando comparado ao RRT-O. Além disso, no mesmo tempo que o RRT-T gera uma única árvore com 1.500 amostras, RRT-O poderia gerar muitas amostras a mais ou outras novas árvores, aumentando as chances de obter uma solução e melhorá-la. Assim, a redução no custo computacional

torna o algoritmo RRT-O mais apropriado que o RRT-T para uma aplicação que envolva rápido processamento, como, por exemplo, em ambientes dinâmicos.

Figura 6.5: Gráfico com o tempo computacional em relação ao número de amostras aleatórias dos algoritmos RRT-T e RRT-O necessárias para obter uma solução ou retornar a ausência de uma.

Em 36.5% das simulações, nenhum método encontrou solução para o problema. Em 58% das simulações, os dois métodos encontraram soluções diferentes. Em apenas 0.9% dos casos os métodos encontraram a mesma simulação. Em 2.3% dos casos, apenas o RRT-T encontrou uma solução e em outros 2.3% dos casos, apenas o RRT-O. Em casos onde ambos métodos encontraram soluções diferentes, o RRT-O obteve em média 71.99% do custo de navegação do RRT-T, com desvio padrão de aproximadamente 44.91%. Portanto, percebe-se que o RRT-O, além de possuir menor custo computacional que o RRT-T, para valores de $m \geq 2$, conseguiu obter soluções de menor custo de navegação que o RRT-T.

6.3 Estudo de caso: Replanejamento de trajetória em ambiente dinâmico utilizando o algoritmo RRT-O

Após a otimização do algoritmo RRT, foram realizadas simulações para replanejamento de trajetória em ambientes dinâmicos, a fim de avaliar o desempenho do algoritmo RRT-O em tais cenários. A abordagem apresentada nesta seção visa apenas tratar do planejamento ou replanejamento em ambientes onde hajam obstáculos dinâmicos, porém, o destino seja estático. Considera-se que o robô sabe a sua exata localização no ambiente.

6.3.1 O ambiente de simulação

No ambiente foram inseridos, além de obstáculos estáticos, dois dinâmicos. Ambos possuíam uma trajetória previamente calculada e cíclica, indo em linha reta de um ponto inicial até um ponto final e, em seguida, retornando. Os obstáculos permaneciam realizando este ciclo até que o robô encontrasse uma solução ou reportasse a ausência de uma. Os obstáculos dinâmicos são representados na cor lilás e numerados na mesma cor. O destino do robô é representado por um ponto amarelo, cercado pela tolerância, representada por um retângulo preto. O ambiente possui dimensões de $40m \times 40m$. A função de custo utilizada para extensão das primitivas foi a distância euclidiana e para a busca da melhor trajetória, escolheu-se aquela com menor tempo de navegação.

6.3.2 Modelo do robô

O robô teve como modelo dinâmico as Equações (5.26), (5.27), (5.28), (5.29), (5.30), onde $b = 0,9m$, $l = 2,5m$, $m_s = 1.000kg$, $J = 1.300kgm^2$, $t_s = 1,2$ e $c_s = 1,1$. O torque foi limitado entre $-2,5kNm$ e $2,5kNm$ e a tensão no servomotor de esterçamento em $-5,0V$ e $5,0V$. O tempo de integração foi de $0,10s$. Considerou-se também que o robô possui conhecimento de todas as trajetórias futuras dos obstáculos dinâmicos.

Foi considerado, também, que o robô possui tempo suficiente para realizar o planejamento e/ou replanejamento. Assim, toda vez que o robô necessita planejar ou replanejar, o ambiente inteiro para, até que seja terminado o planejamento ou replanejamento.

Tais considerações foram realizadas em virtude de não estar no escopo do presente trabalho a estimação tanto das futuras posições dos objetos dinâmicos quanto do tempo que o algoritmo RRT precisaria para encontrar uma solução.

O robô possui um sensor que capta o ambiente em um ângulo de 360° e $20m$ de distância. Os feixes do sensor são representados por pontos vermelhos. Os sensores tem como função apenas identificar os obstáculos dinâmicos. A identificação de um objeto dinâmico pelo sensor fornece ao robô as informações pertinentes aos futuros estados do objeto dinâmico até que este saia do alcance do sensor. Para o sensor detectar um obstáculo dinâmico, basta que um de seus feixes detectem-no.

6.3.3 Replanejamento de trajetória em ambiente dinâmico

Para a simulação do algoritmo RRT-O, as primitivas de movimento foram geradas para $m = 9$ e o bias de expansão foi estipulado em 5% e o número limite de amostras foi limitado em 1.200 amostras.

Na Figura 6.6(a), é mostrado o início da simulação. Inicialmente um obstáculo dinâmico 1 é detectado pelo sensor. Este obstáculo move-se apenas na direção vertical. O robô, então, adquire os dados a respeito das futuras posições do obstáculo dinâmico identificado. O obstáculo dinâmico 2, que se move apenas na horizontal, não é detectado pelo sensor.

Ao acessar os dados sobre o obstáculo dinâmico, o robô sabe as posições que o obstáculo percorrerá ao longo do tempo. Como o algoritmo planeja utilizando o algoritmo RRT-O, cada primitiva é gerada com a integração de um intervalo fixo e conhecido de tempo. Dessa forma, o robô tem conhecimento de seus futuros estados ao longo do tempo. A par de ambas informações, é possível verificar se haverá colisão com os obstáculos dinâmicos.

O algoritmo RRT-O no ambiente dinâmico funciona da seguinte forma. Quando uma amostra aleatória é gerada e o nó mais próximo é encontrado, as primitivas são geradas e o robô sabe o tempo que levará para chegar do estado inicial à cada uma das primitivas. Ao cruzar essas informações com as posições dos obstáculo nesse mesmo instante de tempo é possível verificar se haverá colisões. Se uma primitiva apresentar colisão com os objetos

estáticos e com os dinâmicos ela é descartada. As demais etapas do RRT-O seguem normalmente.

Dessa forma, o robô pode realizar o planejamento de forma que não haja colisão ao longo do tempo. A trajetória obtida é mostrada por uma linha vermelha na Figura 6.6(b). Após a trajetória ser obtida, inicia-se a movimentação do robô e dos obstáculos dinâmicos, conforme mostrado na Figura 6.6(c).

Na Figura 6.6(d), o robô detecta o obstáculo dinâmico 2, toma conhecimento de seus futuros estados e verifica se a rota é segura. Se for, continua seguindo o plano. Caso contrário, reexecuta o algoritmo RRT-O considerando os obstáculos detectados no presente momento. Durante o replanejamento, como mencionado anteriormente, considera-se que o robô pode replanear instantaneamente. Neste caso, o robô detecta que haverá colisão.

Após o replanejamento, o robô inicia a execução da nova trajetória. Contudo, logo detecta o obstáculo dinâmico 1, como pode ser visto na Figura 6.7(a). Percebe-se que o obstáculo dinâmico 1 não foi levando em consideração durante o primeiro replanejamento, visto que estava fora de alcance do sensor. Após a obtenção da nova trajetória, o robô inicia a sua execução.

Na Figura 6.7(b), o robô realiza o replanejamento de trajetória, após ter constatado que a trajetória que estava sendo executada geraria colisão. No replanejamento, como os dois obstáculos dinâmicos estão sendo detectados, o replanejamento leva em consideração ambos obstáculos. Durante a execução da nova trajetória, percebe-se que o robô passa entre os dois obstáculos dinâmicos.

Como nenhuma área de segurança foi definida em volta do robô, este pode planejar trajetórias que passam bastante perto dos obstáculos, no entanto, sem gerar colisão. Na Figura 6.7(c), é mostrado o robô após passar entre os obstáculos dinâmicos.

Nas Figuras 6.7(d) e 6.8(a), é possível ver o robô seguindo o novo plano em direção ao destino. Por fim, na Figura 6.8(b), é possível ver o robô alcançando o destino, sem necessitar de mais replanejamentos.

Figura 6.6: (a) Configuração inicial do ambiente, enquanto o robô executa o algoritmo RRT otimizado. (b) Planejamento obtido inicialmente pelo algoritmo RRT otimizado. (c) Início da movimentação do robô e dos obstáculos dinâmicos. (d) Sensor detecta a presença de um obstáculo dinâmico e avalia se a trajetória planejada. Neste caso, o robô detecta que haverá colisão com o obstáculo e replaneja a trajetória.

Figura 6.7: (a) Robô detecta o obstáculo dinâmico 1. (b) Robô replaneja a trajetória e passa entre os dois obstáculos dinâmicos sem causar colisão. (c) Robô após passar entre os obstáculos dinâmicos. (d) Robô executando a trajetória em direção ao destino.

Figura 6.8: (a) Robô se aproximando do destino. (b) Robô alcança o destino sem gerar colisões.

6.3.4 Análise do tempo de replanejamento do algoritmo

Mesmo considerando que o robô pode realizar um planejamento instantâneo, os tempos que o algoritmo RRT-O levou para planejar e replanear foram medidos neste mesmo ambiente dinâmico. Foram realizadas 700 simulações utilizando-se 7 tempos de integração diferentes: $0.1s$, $0.15s$, $0.20s$, $0.25s$, $0.30s$, $0.35s$ e $0.40s$. Para cada um dos sete tempos de integração foram realizadas 100 simulações, onde foram medidos os tempos de planejamento e os tempos de replanejamento global.

Avaliação do planejamento

O algoritmo, inicialmente, executa o RRT-O e tenta realizar apenas um planejamento até o destino. Caso não seja encontrada uma solução, o algoritmo retorna a ausência de uma. Dos 700 planejamentos realizados, em apenas 41 deles o RRT-O não encontrou uma solução. A média do custo computacional, variando de acordo com o tempo de integração utilizado é mostrada na Figura 6.9.

Observa-se que a menor média de custo computacional de planejamento é obtida para

Figura 6.9: Gráfico com o custo computacional de planejamento do algoritmo RRT-O em ambiente dinâmico em relação ao tempo de integração.

o tempo de integração de 0,1s, sendo esta de 2,99s. Os demais tempos de integração avaliados apresentaram custo computacional médio superior a 4,3s. O maior custo computacional médio, de 6,56s, foi obtido para o tempo de integração de 0,35s. Na Tabela 6.9, pode-se ver os custos computacionais máximos e mínimos obtidos em cada tempo de integração.

Percebe-se em alguns casos que o RRT-O consegue realizar o planejamento em menos de 1,0s. Contudo, percebe-se que há casos em que o planejamento é superior a 20s.

Tabela 6.9: Valores máximo e mínimo de tempo de planejamento do algoritmo RRT-O em ambientes dinâmicos e os respectivos tempos de integração.

Avaliação do tempo de planejamento							
Tempos de integração (s)	0,1	0,15	0,20	0,25	0,30	0,35	0,40
Custo computacional mínimo obtido (s)	1,34	1,51	1,16	1,22	0,86	0,77	0,87
Custo computacional máximo obtido (s)	7,87	31,61	23,75	25,37	25,08	18,47	28,12

Avaliação do replanejamento

Para os replanejamentos, o algoritmo realiza uma única tentativa de replanejamento. Se o algoritmo não encontrar solução, retorna-se a ausência de uma. Se encontrar, a simulação segue normalmente e, caso sejam necessários novos replanejamentos, executa-se o RRT-O até que este reporte uma ausência de solução ou que o destino seja alcançado. Durante as simulações, foram realizados 689 replanejamentos e em 195 casos o algoritmo RRT-O não conseguiu realizar o replanejamento. Destaca-se que é possível que seja realizado mais de um replanejamento em uma única simulação. A média do custo computacional, variando também em relação ao tempo de integração utilizado é mostrado na Figura 6.10.

Figura 6.10: Gráfico com o custo computacional de replanejamento do algoritmo RRT-O em ambiente dinâmico em relação ao tempo de integração.

Percebe-se que a menor média de custo computacional, de 4,31s, foi obtido com o tempo de integração de 0,40s. Os demais tempos de integração apresentaram custo computacional médio superior a 5,0s, chegando a 7,27s, no caso do tempo de integração sendo 0,15s. Na Tabela 6.10 pode-se ver os custos computacionais máximos e mínimos obtidos em cada tempo de integração.

Tabela 6.10: Valores máximo e mínimos de tempo de replanejamento do algoritmo RRT-O em ambientes dinâmicos e os respectivos tempos de integração.

Avaliação do tempo de replanejamento							
Tempos de integração (s)	0,1	0,15	0,20	0,25	0,30	0,35	0,40
Custo computacional mínimo obtido (s)	1,40	1,25	0,92	1,05	0,68	0,51	1,01
Custo computacional máximo obtido (s)	15,39	23,33	14,55	26,86	36,97	29,79	26,90

Percebe-se que há casos em que o algoritmo RRT-O consegue replanejar a trajetória em menos de 1s. Contudo, como o RRT é um algoritmo probabilístico, torna-se difícil prever o tempo que necessitará para encontrar uma solução. Percebe-se que há casos em que foram necessários mais de 20s, chegando, no pior dos casos, a 36,97s.

Ao comparar-se os tempos de replanejamento com aqueles de planejamento, percebe-se que a média total de planejamento é inferior a de replanejamento, sendo as médias 5,05s e 5,87s, respectivamente.

Uma provável explicação para isso está no fato de que o planejamento leva em consideração apenas um obstáculo dinâmico, além de iniciar a trajetória com velocidade nula. No replanejamento, o robô já está em movimento e em rota de colisão. Logo, as primitivas geradas durante o replanejamento possuem maiores chances de gerar colisão que as de planejamento. Assim, surge a necessidade de o algoritmo utilizar mais amostras para obter uma solução para o problema, se esta houver.

Capítulo 7

Conclusão

No mestrado propôs-se um método otimizado para geração de primitivas de movimento para o algoritmo RRT. Tal algoritmo gera inicialmente apenas poucas primitivas e, a partir dos cálculos de custo de extensão e verificação de colisões, são identificadas áreas de expansão consideradas promissoras para a geração de novas primitivas. Simulações feitas em um simulador desenvolvido em Matlab mostraram inicialmente apenas os algoritmos de expansão, sem sua utilização no algoritmo RRT. Os resultados mostraram que o método proposto possui maior eficiência computacional que os métodos tradicional e aleatório de geração de primitivas de movimento, mantendo ainda a qualidade das primitivas geradas. Além disso, quanto maior a resolução das primitivas geradas, maior tende a ser a diferença entre os custos computacionais entre o método proposto e os dois métodos tradicional e aleatório.

Posteriormente o algoritmo otimizado foi inserido no algoritmo RRT. Simulações foram realizadas em um simulador desenvolvido em linguagem python comparando o algoritmo RRT otimizado com o algoritmo RRT utilizando a geração de primitivas tradicional. Os resultados mostraram que o algoritmo RRT otimizado possui maior eficiência computacional que o RRT tradicional, mantendo a qualidade das soluções globais obtidas, e em vários casos, obteve soluções melhores.

Como estudo de caso foram realizadas simulações no simulador desenvolvido em Python para ambientes dinâmicos a fim de avaliar o desempenho do algoritmo RRT otimizado, dadas certas condições. Os resultados mostraram que o RRT otimizado conseguiu em

alguns casos realizar o replanejamento de trajetória em menos de 1s. Em outros casos, no entanto, esse tempo ultrapassou os 30s. Assim, apesar de não resolver o problema, o RRT otimizado consegue atenuar o problema, o que o torna mais viável para aplicações em ambientes dinâmicos que o algoritmo RRT tradicional.

A partir dos resultados, foi publicado um artigo científico no Simpósio Brasileiro de Automação Inteligente, (Vieira et al.; 2013). Para trabalhos futuros pretende-se obter provas matemáticas que demonstrem a eficiência e eficácia do método proposto, bem como melhorá-lo a fim de torná-lo mais viável para aplicações em ambientes menos estruturados e estreitos.

Referências Bibliográficas

- Baek, J. Y., S. H. Kim, e M. C. Lee (2010). Path planning of unmanned vehicle using potential field and lane detection. In *Proceedings of the SICE Annual Conference*, pp. 1339–1342.
- Bruce, J. e M. Veloso (2002). Real-time randomized path planning for robot navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Volume 3, pp. 2383 – 2388 vol.3.
- Chen, B. e Z. Jiao (2011). Adaptive path following control of car-like mobile robot using dynamic model. In *Proceedings of the 6th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pp. 239–244.
- Choset, H., K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, e S. Thrun (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Boston: MIT Press.
- Cohen, B., G. Subramanian, S. Chitta, e M. Likhachev (2011). Planning for manipulation with adaptive motion primitives. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 5478–5485.
- Economou, J. T., P. C. K. Luk, A. Tsourdos, e B. A. White (2003). Hybrid modelling of an all-electric front-wheel ackerman steered vehicle. In *Proceedings of the IEEE 58th Vehicular Technology Conference*, Volume 5, pp. 3294–3298 Vol.5.
- Egerstedt, M., X. Hu, e A. Stotsky (1998). Control of a car-like robot using a dynamic model. In *Proceedings of the IEEE International Conference on Robotics and Automation*, Volume 4, pp. 3273–3278 vol.4.

- Fulgenzi, C., C. Tay, A. Spalanzani, e C. Laugier (2008). Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1056 –1062.
- Gang, L., D. Min-zhou, X. Tao, e W. Liang (2011). Multi-agent path planning for unmanned aerial vehicle based on threats analysis. In *Proceedings of the 3rd International Workshop on Intelligent Systems and Applications (ISA)*, pp. 1–4.
- Gopikrishna, V. e M. Huber (2009). A temporal potential function approach for path planning in dynamic environments. In *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pp. 3605 –3611.
- Hausler, A. J., R. Ghabcheloo, I. Kaminer, e A. M. P. ans A. P. Aguiar (2009). Path planning for multiple marine vehicles. In *Proceedings of the Oceans*, pp. 1–9.
- Hota, S. e D. Ghose (2010). Optimal path planning for an aerial vehicle in 3d space. In *Proceedings of the 49th IEEE Conference on Decision and Control (CDC)*, pp. 4902–4907.
- Jeon, J. H., S. Karaman, e E. Frazzoli (2011). Anytime computation of time-optimal off-road vehicle maneuvers using the rrt*. In *Decision and Control and European Control Conference (CDC-ECC), 2011 50th IEEE Conference on*, pp. 3276–3282.
- Jianming, G., Z. Shouping, X. Jia, e Z. Shenghui (2010). Kalman prediction based VHF of dynamic obstacle avoidance for intelligent vehicles. In *Proceedings of the International Conference on Computer Application and System Modeling (ICCASM)*, Volume 3, pp. V3–6 –V3–10.
- Ju, T., S. Liu, J. Yang, e D. Sun (2011). Apply RRT-based path planning to robotic manipulation of biological cells with optical tweezer. In *Proceedings of the International Conference on Mechatronics and Automation (ICMA)*, pp. 221–226.
- Karaman, S. e E. Frazzoli (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research* 30(7), 846–894.

-
- Kim, J. e J. P. Ostrowski (2003). Motion planning a aerial robot using rapidly-exploring random trees with dynamic constraints. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Volume 2, pp. 2200–2205 vol.2.
- Kuffner, J. J. e S. M. LaValle (2000). RRT-connect: An efficient approach to single-query path planning. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Volume 2, pp. 995 –1001 vol.2.
- Kushleyev, A. e M. Likhachev (2009). Time-bounded lattice for efficient planning in dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1662 –1668.
- Kuwata, Y., G. A. Fiore, J. Teo, E. Frazzoli, e J. P. How (2008). Motion planning for urban driving using RRT. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1681–1686.
- Lacevic, B., P. Rocco, e M. Strandberg (2011). Safe motion planning for articulated robots using rrts. In *Proceedings of the XXIII International Symposium on Information, Communication and Automation Technologies (ICAT)*, pp. 1–7.
- Latombe, J.-C. (1991). *Robot Motion Planning*. Norwell, MA, USA: Kluwer Academic Publishers.
- LaValle, S. M. (1998). Rapidly-exploring random trees: A new tool for path planning.
- LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.
- Lee, T., H. Chung, e H. Myung (2011). Multi-resolution path planning for marine surface vehicle considering environmental effects. In *Proceedings of the IEEE OCEANS*, pp. 1–9.
- Li, H. e M. Trentini (2010). Tracking control of autonomous vehicles with slippage. In *Proceedings of the International Conference on Autonomous and Intelligent Systems (AIS)*, pp. 1–6.
- Likhachev, M. e D. Ferguson (2009). Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research* 28(8), 933–945.

- Macek, K., K. Thoma, R. Glatzel, e R. Siegwart (2007). Dynamics modeling and parameter identification for autonomous vehicle navigation. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3321–3326.
- Moret, E. N. (2003). Dynamic modeling and control of a car-like robot. Master thesis, Faculty of the Virginia Polytechnic Institute and State University.
- Narayanan, V., M. Phillips, e M. Likhachev (2012). Anytime safe interval path planning for dynamic environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4708 –4715.
- Orthey, A., M. Toussaint, e N. Jetchev (2013). Optimizing motion primitives to make symbolic models more predictive. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pp. 2868–2873.
- Pepy, R., A. Lambert, e H. Mounier (2006). Path planning using a dynamic vehicle model. In *Proceedings of the 2nd Information and Communication Technologies (ICTTA)*, Volume 1, pp. 781–786.
- Petti, S. e T. Fraichard (2005). Safe motion planning in dynamic environments. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2210 – 2215.
- Phillips, M. e M. Likhachev (2011). Sipp: Safe interval path planning for dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5628 –5635.
- Pivtoraiko, M., I. A. D. Nesnas, e A. Kelly (2009). Autonomous robot navigation using advanced motion primitives. In *Proceedings of the IEEE Aerospace conference*, pp. 1 –7.
- Schroder, J., T. Gindele, D. Jagszent, e R. Dillmann (2008). Path planning for cognitive vehicles using risk maps. In *Proceedings of the IEEE Intelligent Vehicles Symposium*, pp. 1119–1124.

-
- Shan, E., B. Dai, J. Song, e Z. Sun (2009). A dynamic RRT path planning algorithm based on b-spline. In *Proceedings of the Second International Symposium on Computational Intelligence and Design (ISCID)*, Volume 2, pp. 25 –29.
- Stotsky, A. e X. Hu (1998). Adaptive/variable structure control of car-like mobile robot in four wheel dynamical model framework. In *Proceedings of the 37th IEEE Conference on Decision and Control*, Volume 3, pp. 3111–3116 vol.3.
- Svenstrup, M., T. Bak, e H. J. Andersen (2011). Minimising computational complexity of the RRT algorithm a practical approach. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5602 –5607.
- Tsai, P.-S., L.-S. Wang, F.-R. Chang, e T.-F. Wu (2004). Point stabilization control of a car-like mobile robot in hierarchical skew symmetry chained form. In *Proceedings of the IEEE International Conference on Networking, Sensing and Control*, Volume 2, pp. 1346–1351 Vol.2.
- Ugur, E., E. Sahin, e E. Oztop (2012). Self-discovery of motor primitives and learning grasp affordances. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pp. 3260–3267.
- Urmson, C. e R. Simmons (2003). Approaches for heuristically biasing RRT growth. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Volume 2, pp. 1178 – 1183 vol.2.
- Vieira, H. L., V. G. Jr, e M. da Silva (2013). Redução de custo na geração de primitivas de movimento através de otimização por eliminação. In *Anais do XI Simpósio Brasileiro de Automação Inteligente (SBAI)*.
- Wu, L., Y. Tian, e Y. Yang (2009). Real-time path planning for aerial robot in real environments. In *Proceedings of the 35th Annual Conference of IEEE Industrial Electronics (IECON)*, pp. 2320–2325.
- Xia, Z., G. Chen, J. Xiong, Q. Zhao, e K. Chen (2009). A random sampling-based approach to goal-directed footstep planning for humanoid robots. In *Proceedings of*

the IEEE/ASME International Conference on Advanced Intelligent Mechatronics, pp. 168–173.

Yershova, A., L. Jaillet, T. Simeon, e S. M. LaValle (2005). Dynamic-domain rrts: Efficient exploration by controlling the sampling domain. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3856 – 3861.

Zucker, M., J. Kuffner, e M. Branicky (2007). Multipartite rrts for rapid replanning in dynamic environments. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pp. 1603 –1609.