UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

Victoria Sayuri Sekimura Fujii

Planejamento de Caminho para Veículo Autônomo Articulado

São Carlos

2020

Victoria Sayuri Sekimura Fujii

Planejamento de Caminho para Veículo Autônomo Articulado

Monografia apresentada ao Curso de Engenharia Elétrica com Ênfase em Eletrônica, da Escola de Engenharia de São Carlos da Universidade de São Paulo, como parte dos requisitos para obtenção do título de Engenheiro Eletricista.

Orientador: Prof. Dr. Valdir Grassi Jr.

São Carlos

2020

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

F961p

Fujii, Victoria Sayuri Sekimura
Planejamento de Caminho para Veículo Autônomo Articulado / Victoria Sayuri Sekimura Fujii; orientador Valdir Grassi Jr. São Carlos, 2020.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2020.

1. Planejamento de Caminho. 2. Rapidly-Exploring Random Tree. 3. Veículo Articulado. 4. OMPL. 5. Detecção de Colisão. 6. Flexible Collision Library. 7. Reeds Shepp. 8. Truck-Trailer. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Victoria Sayuri Sekimura Fujii

Título: "Planejamento de Caminho para Veículo Autônomo Articulado"

Trabalho de Conclusão de Curso defendido e aprovado em 03 / 07 / 2020,

com NOTA 9,5 (nove , cinco), pela Comissão Julgadora:

Prof. Dr. Valdir Grassi Junior - Orientador - SEL/EESC/USP

Prof. Titular Marco Henrique Terra - SEL/EESC/USP

Mestre João Roberto Soares Benevides - Doutorando - SEL/EESC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

Dedico este trabalho à minha família, aos meus amigos e a todos aqueles interessados por automobilística, programação e sistemas inteligentes autônomos. Que este trabalho sirva de inspiração e apoio para novas pesquisas.

AGRADECIMENTOS

Primeiramente, agradeço ao meu professor e orientador Valdir Grassi Jr. pela oportunidade de realizar esse estudo e, principalmente, por todo o suporte, incentivo, inspiração e disposição empenhados no desenvolvimento deste projeto, acompanhando e auxiliando ao máximo possível para eu conseguir estruturar o conhecimento necessário para concretizar este trabalho final. Agradecimentos especiais à minha família e a todos os meus amigos que estiveram ao meu lado nessa jornada da graduação, oferecendo apoio, carinho e descontração e me inspirando tanto nos momentos bons quanto nos momentos difíceis. Agradeço a todos aqueles que acreditam em mim e no meu potencial e que, de alguma maneira, mesmo que singela, fizeram parte dos meus crescimentos pessoal e profissional até hoje.

"Há um grande desejo em mim de sempre melhorar.
Melhorar.
É o que me faz feliz."
Ayrton Senna

RESUMO

FUJII, V. **Planejamento de Caminho para Veículo Autônomo Articulado**. 2020. 87p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2020.

Considerando o contexto atual da era da tecnologia em que veículos autônomos circulando nas estradas já se tornaram cenas da realidade em diversos países, sabe-se que um grande passo como este consolidou-se graças às inúmeras pesquisas realizadas no ramo para conseguir aprimorar e garantir a segurança, viabilidade e eficiência dos veículos inteligentes. Neste trabalho, dessa forma, buscou-se explorar os conceitos acerca do planejamento de caminho para veículos articulados, a partir dos quais foi possível compreender determinadas técnicas relacionadas à implementação deste planejamento para estudos de caso simples, focando no veículo autônomo articulado. Esta espécie de veículo apresenta em sua mobilidade determinadas dificuldades e restrições em virtude de suas grandes dimensões e da característica de possuir mais de um componente em sua estrutura. Peculiaridades essas que tornam o seu planejamento de caminho mais complexo. Logo, o trabalho envolvido na concretização de um algoritmo de planejamento de caminho para um veículo articulado abordou conceitos sobre modelagens cinemáticas do veículo de estudo e sobre métodos e tipos de planejamento de caminho apropriados que existem na literatura. Tendo em vista que há diversas bibliotecas de programação desenvolvidas, em linguagens como C++, especificamente para realizar de maneira eficiente tal implementação, estudaram-se também duas bibliotecas que foram fundamentais para o desenvolvimento deste projeto: OMPL e FCL. Através da compatibilidade destas, o planejamento foi construído para simular determinadas situações comuns para veículos, como estacionamento e passagem estreita, analisando, com isso, a atuação de diferentes planejadores disponíveis e a influência de parâmetros relacionados ao veículo no resultado final.

Palavras-chave: Planejamento de caminho. Veículo autônomo. Veículo articulado. Detecção de colisão. OMPL. Planejamento de trajetória. Modelo cinemático. Caminhão autônomo. FCL. Dubins. Reeds Shepp. RRT. Benchmarking.

ABSTRACT

FUJII, V. **Path Planning for Autonomous Articled Vehicle**. 2020. 87p. Monografia (Trabalho de Conclusão de Curso) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2020.

Considering the current context of the technology era in which autonomous vehicles circulating on the roads have already become part of current reality in several countries, it is known that a major step like this has been consolidated thanks to the countless researches carried out in the field to be able to improve and guarantee the safety, feasibility and efficiency of smart vehicles widely. In this work, therefore, it was sought to explore the concepts about path planning for articulated vehicles from which it was possible to understand certain techniques related to the implementation of this planning for simple study cases, focusing on autonomous articulated vehicle. This kind of vehicle presents in its mobility certain difficulties and restrictions due to its large dimensions and the characteristic of having more than one component in its structure. These peculiarities make your path planning more complex. Therefore, the work involved in the realization of a path planning algorithm for an articulated vehicle addressed concepts about kinematic modeling of the desired vehicle and about appropriate methods and types of path planning that exist in the literature. Bearing in mind that there are several programming libraries developed, in languages such as C ++, specifically to carry out such implementation efficiently, two libraries were also studied that were fundamental for the development of this project: OMPL and FCL. Through their compatibility, the planning was built to simulate common situations for vehicles, such as parking and narrow passage, and finally to analyze the performance of different available planners and the influence of parameters related to the vehicle in the final result.

Keywords: Path Planning. Autonomous Vehicle. Articulated Vehicle. Collision Detection. OMPL. Motion Planning. Autonomous Truck-Trailer. FCL. Dubins. Reeds Shepp. RRT. Benchmarking..

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AHS	do inglês, *Automated Highway System*
ALV	do inglês, *Autonomous Land Vehicle*
API	do inglês, *Application Programming Interface*
CaRINA	Carro Robótico Inteligente para Navegação Autônoma
EUA	Estados Unidos da América
FCL	do inglês, *Flexible Collision Library*
GM	*General Motors*
GPS	do inglês, *Global Positioning System*
ICMC	Instituto de Ciências Matemáticas e de Computação
LRM	Laboratório de Robótica Móvel
OMPL	do inglês, *Open Motion Planning Library*
PQP	do inglês, *Proximity Query Package*
PRM	do inglês, *Probabilistic Roadmap Method*
ROS	do inglês *Robot Operating System*
RRT	do inglês, *Rapidly-exploring Random Tree*
USPSC	Campus USP de São Carlos
USP	Universidade de São Paulo
VA	Veículo Autônomo

SUMÁRIO

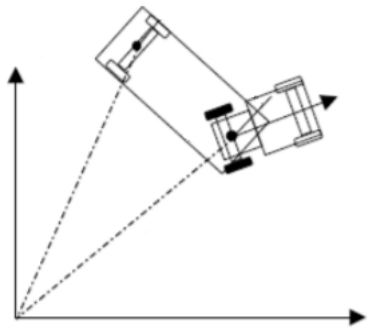
1 INTRODUÇÃO

Com a expansão do setor de veículos autônomos inteligentes nas últimas décadas, têm se tornado cada vez mais necessário desenvolver e estudar técnicas e metodologias de modo a possibilitar que tais veículos sejam capazes de determinar de maneira autônoma as trajetórias mais convenientes, seguras e eficientes entre um ponto de partida e o seu destino almejado.

Considerando a estatística dada pela World Health Organization (2020) de que, a cada ano, aproximadamente 1,35 milhão de pessoas morrem e entre 20 e 50 milhões sofrem ferimentos não letais devido a acidentes no trânsito causados em sua maioria por erros humanos, retirá-los do controle dos veículos tem chamado a atenção de grandes companhias de tecnologia e de montadoras de automóveis não somente em virtude dos previstos impactos positivos na redução de emissão de gases do efeito estufa e na eficiência para atender a crescente demanda por transporte de bens e de pessoas, mas preocupando-se também com a segurança no trânsito (LJUNGQVIST, 2019).

Como dito por Zips, Böck e Kugi (2015), em vários países, o transporte de bens tem um pilar extremamente relevante representado pelos caminhões de grande porte, os quais contribuem também na redução de emissão de poluentes visto que as carretas podem ser acopladas a eles, evitando aumentar o tráfego de veículos. Devido às suas grandes dimensões e à maior quantidade de graus de liberdade nesses sistemas, existem peculiaridades em seu planejamento de trajetória objetivando, além de evitar colisões com obstáculos presentes no ambiente externo, também evitar manobras inadequadas que podem provocar acidentes relacionados à própria obstrução do veículo em ruas estreitas ou curvas fechadas (posição conhecida do inglês *jackknife*, Figura 1) ou no momento de entrar em uma vaga de estacionamento.

Figura 1 – Ilustração da posição *jackknife* a ser evitada para veículos articulados.



Fonte: Bouteldja e Cerezo (2011).

Um dos desafios presentes na evolução deste setor consiste, então, em realizar estudos sobre os planejamentos de trajetória e caminho para que tais veículos encontrem a trajetória ou caminho ótimo entre dois pontos, podendo ser essa otimização baseada em critérios distintos como distância total percorrida, tempo de deslocamento ou consumo energético. Sendo que, dentre os diferentes tipos de veículos existentes, aquele desafio torna-se maior para o caso de veículos articulados, como caminhões ou qualquer veículo conectado a uma ou mais carretas. Zips, Böck e Kugi (2015) e Elhassan (2015) expõem um planejamento de caminho para caminhões articulados com apenas uma carreta objetivando atingir uma configuração final correspondendo a estacionamento em uma baia de carregamento, ressaltando a importância de inserir na estratégia de planejamento evitar colisões com os objetos externos, bem como, a posição de *jackknife*. Por outro lado, uma aplicação mais complexa consiste em caminhões cada vez maiores, como o modelo de duas carretas.

Tendo em vista a relevância em realizar a descrição do espaço com as configurações livre de colisão do objeto de estudo, existem na literatura diversos métodos de planejamento propostos para tentar facilitar esse processo, mas há o entrave do alto custo computacional em virtude dos métodos dependerem diretamente da dimensão do espaço de configuração. Com isso, foi estudado mais a fundo sobre métodos baseados em amostragem, cuja característica principal consiste em verificar se uma única configuração do corpo está ou não contida no espaço de configurações livre de colisão e, consequentemente, independe da dimensão desses espaços (BENEVIDES, 2015).

Nesta classe de métodos de planejamento, pode-se citar o algoritmo PRM (do inglês *Probabilistic Roadmaps Method*), construído como grafos, é um modelo eficiente para espaço de trabalho estático e se divide em duas fases básicas: fase de aprendizado e fase de questionamento. A primeira fase consiste em encontrar uma quantidade pré-definida de configurações livre de colisão e conectar os nós vizinhos entre si para construir o grafo. A segunda fase determinará, por meio de algoritmos de busca, o caminho através da ligação entre as configurações inicial e final a partir do mapa criado anteriormente. Entretanto, este processo de conexão ainda pode apresentar grande complexidade principalmente em sistemas dinâmicos e não-holonômicos (LAVALLE, 1998).

Além do PRM, a classe de planejamentos baseados em amostragem possui também um algoritmo mais adequado e eficiente para o caso de sistemas com restrições não-holonômicas e com elevados graus de liberdade: RRT (do inglês *Rapidly-exploring Random Tree*). Proposto por LaValle (1998), este planejamento se destaca por ter a tendência a crescer em direção às áreas ainda inexploradas do cenário, expandindo assim uma árvore de exploração do ponto inicial até o objetivo desejado através de sucessivas amostragens de estados aleatórias no espaço livre de colisão. Mas, como explanado detalhadamente por Kuffner e LaValle (2000), o RRT assemelha-se ao PRM por também ser probabilisticamente completo, o que significa possuir a probabilidade de encontrar uma solução que converge

para 1 conforme o tempo tende ao infinito.

1.1 Objetivos

Frente a tal contexto, este trabalho tem como objetivo geral fazer a implementação de um sistema de planejamento de caminho para um veículo articulado com apenas uma carreta, aplicando o planejador RRT a fim de analisar os parâmetros que influenciam no resultado e de estudar o comportamento do sistema em diferentes cenários simulados, como por exemplo fazer um desvio simples de obstáculos, passar por trechos estreitos e realizar seu estacionamento.

Os objetivos específicos, portanto, são:

- Modelagens matemática e cinemática do veículo articulado com uma carreta.
- Implementação de um algoritmo que realize o planejamento de caminho do veículo articulado tendo como base suas dimensões e restrições diferenciais para guiar o mesmo até seu destino.
- Simulação do algoritmo desenvolvido para diferentes situações cotidianas.

1.2 Organização

O estudo desenvolvido neste projeto foi distribuído nos seguintes capítulos:

1. No **Capítulo 2**, constam a fundamentação teórica sobre veículos autônomos e articulados, modelagem cinemática e planejamento de caminho e seus conceitos principais.
2. No **Capítulo 3**, há a descrição da série de etapas e testes do processo de implementação do algoritmo de planejamento, incluindo detalhes acerca das ferramentas, bibliotecas e técnicas utilizadas na elaboração do algoritmo.
3. No **Capítulo 4**, são expostos os resultados finais gerados pelo algoritmo de planejamento, bem como simulações, após aplicar a metodologia explanada. Trata-se também da discussão acerca dos resultados atingidos e avaliando a metologia aplicada.
4. No **Capítulo 5**, encontram-se as considerações finais sobre o estudo realizado neste projeto, sugerindo também as proposições de trabalhos futuros para este ramo de estudo.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo aborda os principais assuntos necessários para o entendimento deste trabalho. São abrangidos conceitos sobre veículos autônomos e articulados, sobre planejamento de trajetória e caminho, sobre algoritmos de controle usados nos sistemas inteligentes e também sobre trabalhos já realizados nesta área.

2.1 Veículos Autônomos E Articulados

2.1.1 Veículos Autônomos

O início da era da automação veicular e suas respectivas pesquisas é atribuído à exposição Futurama, em 1939, na Feira Mundial de Nova Iorque/EUA, onde, patrocinado pela *General Motors* (GM), Norman Melancton Bel Geddes apresentou um protótipo de AHS (Sistema de Rodovia Automatizado) de tamanho reduzido para demonstrar como seria a locomoção dos veículos nos próximos anos ao implementar rodovias inteligentes. Apesar do projeto não ter progredido na época, devido ao contexto da Segunda Guerra Mundial, foram criadas várias tecnologias com posteriores aplicações veiculares, a exemplo do GPS (Sistema de Posicionamento Global) (RODRIGUES, 2017).

Os experimentos com carros autônomos tiveram início desde a década de 50 quando foram feitos os primeiros testes promissores nessa área, remetendo aos carros-conceito desenvolvidos pela GM. Porém, apenas na década de 80, foram constatados os verdadeiros carros auto sustentáveis e autônomos, a exemplo de dois projetos: um de 1984 realizado pela *Carnegie Mellon University* com seus projetos Navlab e ALV (Veículo Terrestre Autônomo) e, em 1987, o projeto *Eureka Prometheus* da *Mercedes-Benz* e *Bundeswehr University Munich* (PISSARDINI; WEI; JR., 2013).

O conceito de veículo autônomo (VA) advém de um veículo capaz de se locomover de forma completamente autônoma, isto é, com capacidade de autogovernar-se. Isso se dá através da leitura e interpretação do mundo ao redor dele mesmo, tornando o próprio veículo, dessa forma, responsável pela sua interação com os elementos externos a ele e pela sua integração segura e otimizada ao sistema de trânsito. Com isso, pode-se concluir que um veículo autônomo é caracterizado por grandes tecnologias para conseguir integrar tais sistemas (RODRIGUES, 2017).

Logo, a existência e o desenvolvimento de veículos autônomos devem-se, principalmente, ao uso de três tecnologias principais que podem ser resumidas em: sensores, conectividade e algoritmos de controle.

Primeiramente, para conseguir obter as informações acerca do ambiente de condução do carro, é necessário que haja dispositivos de leitura de 360° em tempo real sobre as

características do meio externo, o que corresponde à função dos sensores, como radar, câmera e ultrassônicos, posicionados em diversas partes do carro a fim de se obter dados com maior precisão e, consequentemente, ter maior segurança nas tomadas de decisões.

Já a conectividade é utilizada para o veículo ter acesso às últimas informações e notícias sobre o trânsito, clima, mapas, condições da rodovia, e assim aproveitar esses dados para, por exemplo, antecipar freagens e evitar possíveis acidentes na estrada. Além disso, a conectividade possibilita que os VAs disponham do *GPS (Sistema de Posicionamento Global)* para localização dos mesmos.

Os algoritmos de planejamento de movimento e controle têm a finalidade de tratar e processar os dados obtidos pelos sensores e também recebidos via conexão com a rede, e, através desse processamento, calcular a melhor decisão que atuará diretamente na direção, nos freios, na rota a ser feita e na velocidade do veículo. Essa terceira tecnologia empregada nos VAs trata-se do maior desafio deste desenvolvimento, pois há uma enorme quantidade de variáveis e de dados a serem considerados e aprendidos com o tempo pelo algoritmo ao realizar a tomada de decisão.

Vale explanar também que, dentro do universo de veículos autônomos, existem diferentes níveis de autonomia formalmente considerados (WIRED, 2016):

- **Nível 0 - Sem automação**: o motorista humano é responsável pelo controle e desempenho em tempo integral de todas as tarefas de condução do veículo;
- **Nível 1 - Assistente de direção**: através de informações do ambiente de condução, há um sistema de assistência atuando sobre a direção ou aceleração/desaceleração do veículo, sendo que as demais tarefas ficam a cargo do motorista humano;
- **Nível 2 - Automação parcial**: através das informações do ambiente de condução, o sistema de assistência é capaz de controlar a direção e a aceleração/desaceleração, mas as demais tarefas continuam sob responsabilidade do motorista humano, logo, este deve monitorar e agir como um *callback*;
- **Nível 3 - Automação condicional**: todas as tarefas de condução são realizadas pelo sistema de direção automatizado, podendo o motorista humano intervir quando necessário;
- **Nível 4 - Elevada automação**: em uma área definida com restrições de clima e área, todos os aspectos da tarefa de condução são realizados pelo sistema de direção automatizado, mesmo sem intervenção humana quando o sistema requisitar ou alertar;
- **Nível 5 - Completa automação**: em todas as estradas e condições climáticas, o sistema de direção automatizado realiza todas as tarefas de direção, não havendo

necessidade de intervenção humana.

Considerando os níveis de autonomia listados anteriormente, é possível notar algumas das vantagens dos VAs, como:

- Redução de acidentes no trânsito causados por falha humana;
- Redução da emissão de gases poluentes e consumo de energia;
- Redução de trânsito e congestionamentos nas cidades;
- Promover a locomoção de pessoas que não tem habilitação ou que possuam dificuldades ou limitações para dirigir, como por exemplo, deficientes físicos.

Atualmente, já se tornou comum veículos possuírem algumas características semi-autônomas, como por exemplo, sistemas de freio automático e sensores/assistente de estacionamento. Entretanto, a realidade de veículos completamente autônomos ainda segue com diversos estudos profundos e estágios de testes para se concretizar e se estabelecer amplamente pelas ruas e estradas do mundo. Dos níveis de autonomia listados anteriormente, a maioria das empresas automobilísticas nos últimos anos tem se empenhado bastante no desenvolvimento de veículos com autonomia Nível 3 ou 4 (FAGGELLA, 2017).

Estudos para aperfeiçoar cada vez mais os sistemas inteligentes de VAs podem ser encontrados tanto em projetos internos de empresas do setor automotivo quanto em instituições acadêmicas. Uma das referências de protótipo experimental encontra-se dentro do Campus USP São Carlos, trata-se do projeto *CaRINA (Carro Robótico Inteligente para Navegação Autônoma)* do Laboratório de Robótica Móvel (LRM), pertencente ao Instituto de Ciências Matemáticas e de Computação (ICMC). Sendo que um dos protótipos de teste é um Palio Adventure 2011 exibido na Figura 2 (PRADO, 2013).

Figura 2 – Veículo experimental do projeto CaRINA.

Fonte: Universidade de São Paulo (2015).

2.1.2 Veículos Articulados

Dentre os tipos de veículos existentes, os articulados se caracterizam pela capacidade de carregar cargas maiores através das estradas e ruas. Logo, constituem uma parte da frota de automóveis extremamente importante para o transporte e logística de bens em todo o mundo. Entretanto, com o aumento da demanda global por produtos, do número de veículos nas ruas, e assim aumento na intensidade dos tráfegos, uma maneira de tentar reduzir a quantidade de veículos no trânsito trata-se de acoplar carretas aos tratores/carros. Com isso, passam a ter mais caminhões longos ao se adicionar mais ou maiores carretas.

Em geral, os veículos articulados são compostos por duas partes: um trator ou motor primário acoplado a um semi-reboque sem direção, ou seja, estes veículos são guiados apenas pelo eixo dianteiro do trator, enquanto que as demais rodas presentes na parte traseira do trator e no semi-reboque não possuem direção para coordená-las. Por isso, os veículos articulados com essa configuração são chamados de "*truck-trailers*".

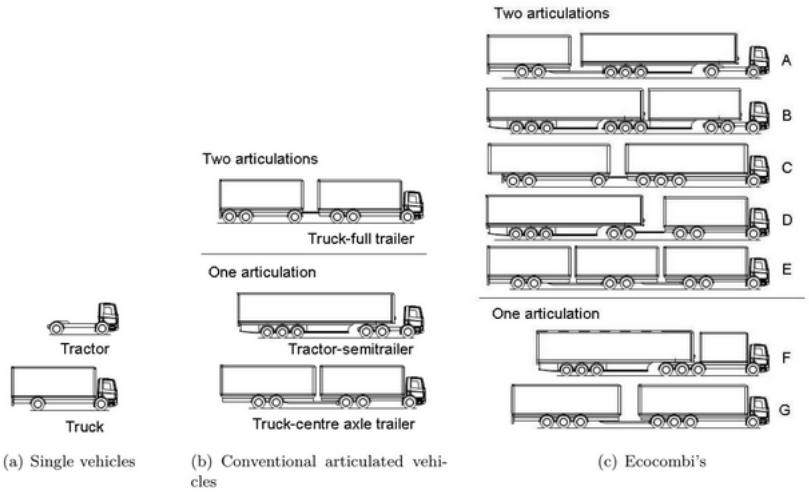
Devido a esta característica de possuir o acoplamento de um semi-reboque, os veículos articulados apresentam vantagens de flexibilidade e eficiência no transporte de cargas, porém, ao mesmo tempo possuem limitações, em virtude do seu extenso tamanho e de seu restrito controle direcional, que os tornam inadequados para determinadas operações durante sua locomoção. Isto é, passam a ter uma baixa capacidade de fazer manobras em ruas estreitas, curvas muito acentuadas, rotatórias e, inclusive, para estacionar.

Apesar de parecidos, os caminhões podem apresentar diferentes configurações em relação a sua articulação que é responsável por possibilitar a conexão entre um veículo e uma carreta ou, para o caso de mais de uma carreta acoplada, entre duas carretas. Isso se deve aos distintos propósitos para cada modelo. Assim, cada tipo de combinação desencadeia em diferentes modelagens matemáticas, cinemáticas e dinâmicas, já que suas geometrias e, também, suas variáveis de controle são peculiares entre si. A Figura 3 exibe algumas combinações de caminhão existentes no mercado, sendo que a maioria dos caminhões convencionais utilizados possui os modelos representados nas Figuras 3a e 3b.

Nota-se que o trator, o qual é o veículo sem carga acoplada, e o caminhão, o qual já possui alguma carga, da Figura 3a, são veículos individuais por não apresentarem articulação. Já a Figura 3b, mostra que existem duas configurações possíveis para veículos convencionais de uma articulação: um trator semi-trailer e um caminhão com reboque de eixo central. Na mesma Figura 3b, há um exemplo de modelo convencional para duas articulações conhecido como caminhão de trailer completo (LUIJTEN, 2010).

Em virtude de suas grandes dimensões e sistemas de controle mais complexos do que um veículo não-articulado, os estudos e estágios de testes de veículos articulados podem levar mais alguns anos para conseguir concretizar o objetivo de implementar frotas autônomas nas estradas. Uma referência de protótipo experimental encontra-se também

Figura 3 – Combinações de caminhões a) Veículos simples; b) Veículos articulados convencionais; c) Ecocombis



Fonte: Luijten (2010).

Figura 4 – Caminhão autônomo desenvolvido na USP São Carlos com Scania

Fonte: Universidade de São Paulo (2015).

no Campus USP São Carlos. Trata-se do projeto da Universidade de São Paulo firmado em parceria com a Scania em 2013, cujo resultado foi apresentado com sucesso dois anos depois, utilizando um caminhão Scania G360 6x4 (Figura 4) para o desenvolvimento (ALVES, 2015).

2.2 Modelagem Do Veículo

Tendo a compreensão dos diferentes tipos de veículos autônomos existentes e, para que o algoritmo de planejamento seja bem estruturado, é uma etapa fundamental do desenvolvimento realizar a modelagem do veículo específico que será o objeto de estudo deste trabalho, isto é, um veículo articulado. Através dessas modelagens, torna-se possível, por exemplo, extrair as informações acerca do comportamento do veículo e, com isso, poder analisar e incorporar os fatores mais relevantes da locomoção do veículo no sistema e gerar o caminho correto.

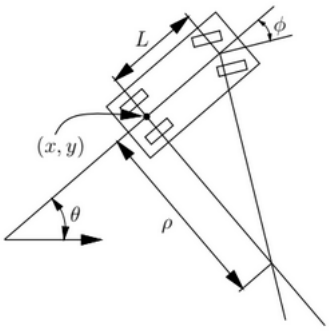
Para realizar os seus movimentos de forma autônoma, os sistemas internos dos VAs, isto é, seus algoritmos de planejamento devem levar em consideração diferentes e importantes conceitos físicos relacionados às condições do veículo como: posição, velocidade, direção, aceleração, orientação, trajetória, tempo, distância, deslocamento. Tais parâmetros constituem a chamada cinemática veicular, a qual se trata da análise dos movimentos que o veículo irá realizar e, assim, escolher a melhor forma de atuação (RODRIGUES, 2017).

2.2.1 Modelagem Cinemática De Veículo Simples

Uma maneira mais didática de se compreender os modelos cinemáticos de veículos é explanada por LaValle (2006) em seu livro, o qual apresenta que um veículo simples possui, basicamente, três graus de liberdade. Através de seu modelo cinemático, na Figura 5, considerando-o como um corpo rígido se movendo no plano com duas rodas mantidas por um eixo fixo e outras duas direcionadas pelo giro do volante, obtém-se a configuração dada por $q = (x, y, \theta)$, sendo o ponto (x, y) o centro do eixo das rodas traseiras e o ângulo θ, o ângulo de orientação do veículo em relação ao eixo de referência, que neste caso, considera-se o eixo x.

Na Figura 5, há também parâmetros como a distância L entre os eixos das rodas dianteiro e traseiro; o ângulo de esterçamento ϕ e o raio ρ de curvatura quando o veículo faz movimentos circulares.

Figura 5 – Modelo cinemático de um veículo simples



Fonte: Adaptado de LaValle (2006).

Além disso, considera-se que a velocidade do veículo é dada por v. Dessa forma, após a aplicação de técnicas trigonométricas no modelo explicado por LaValle (2006), conclui-se que a cinemática deste veículo simples é dada pelo sistema de equações das velocidades $\dot{x}$, $\dot{y}$ e $\dot{\theta}$ seguinte:

$$\dot{x} = v\, cos\, \theta$$

$$\dot{y} = v\, sen\, \theta$$

$$\dot{\theta} = \frac{v}{L} tan\, \phi$$

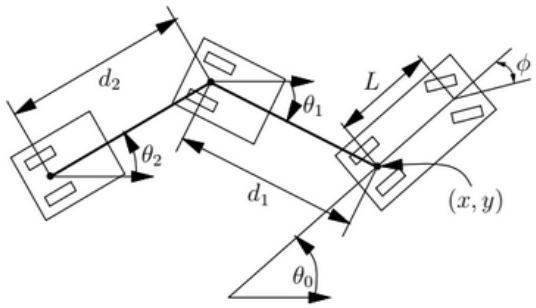
O conjunto de ações de controle consiste, então, em (v, ϕ), do qual supõe-se que $v \in \mathcal{R}$ pode assumir qualquer valor, enquanto que o ângulo de esterçamento ϕ é limitado a um intervalo que se aproxime de um esterçamento de veículo real. Devido à expressão $tan\,\phi$ presente na equação de $\dot{\theta}$, deve haver cautela na definição do intervalo de valores para o ângulo de esterçamento, sendo que limites dados como $\phi \in (-\pi/2, \pi/2)$ evitam esta problemática.

2.2.2 Modelagem Cinemática De Veículo Articulado

Quando uma ou mais carretas são acopladas a um simples veículo, são criados sistemas com maior complexidade e com sua sensibilidade diretamente relacionada com o ponto de conexão e a geometria dos corpos. Considere um veículo simples com n carretas acopladas, como mostra a Figura 6, onde cada carreta está conectada ao centro do eixo traseiro do corpo a sua frente (LAVALLE, 2006).

A diferença em comparação com o modelo cinemático do veículo simples explicado na Seção 2.2.1 encontra-se no parâmetro adicional d_i, denominado comprimento do engate (do inglês *hitch length*), que é a distância do centro do eixo principal da carreta i até o ponto de conexão com o corpo a sua frente. Surgem também mais valores de ângulo de orientação de cada carreta em relação ao mundo, dados por θ_i.

Figura 6 – Modelo cinemático de um veículo articulado puxando n-trailers



Fonte: Adaptado de LaValle (2006).

Como visto anteriormente, na Seção 2.2.1, o veículo simples apresenta variáveis de estado da forma $\mathcal{R}^2 \times \mathcal{SO}(2)$, portanto, com a inserção de mais corpos no modelo cinemático do veículo, cada uma das carretas contribui com um componente do tipo $\mathcal{SO}(2)$. Resultando, assim, em um sistema final com dimensão $n + 3$.

Aplicando as devidas análises trigonométricas e manipulações matemáticas, chega-se ao seguinte conjunto de equações que definem o modelo cinemático do veículo articulado da Figura 6:

$$\dot{x} = v\,cos\,\theta_0$$

$$\dot{y} = v\,sen\,\theta_0$$

$$\dot{\theta}_0 = \frac{v}{L} tan\,\phi$$

$$\vdots$$

$$\dot{\theta}_i = \frac{v}{d_i} \left(\prod_{j=1}^{i-1} cos(\theta_{j-1} - \theta_j) \right) sen(\theta_{i-1} - \theta_i)$$

$$\vdots$$

$$\dot{\theta}_k = \frac{v}{d_k} \left(\prod_{j=1}^{k-1} cos(\theta_{j-1} - \theta_j) \right) sen(\theta_{k-1} - \theta_k)$$

2.3 Planejamento De Caminho

Para que o veículo realize a tarefa de se deslocar de forma autônoma entre o seu ponto de partida até um destino final pelo ambiente evitando colisões, aplica-se o planejamento de caminho. Podendo esse tipo de algoritmo ser utilizado não somente no campo de veículos autônomos mas, igualmente, no campo da robótica, empregando-o nos robôs manipuladores. Sendo que, quanto maior a quantidade de graus de liberdade que o objeto de estudo autônomo possuir, maior serão a complexidade e a dificuldade para se determinar com precisão e segurança o melhor caminho a ser efetuado.

2.3.1 Definição Do Problema De Planejamento

Pode-se dizer que o termo planejamento dentro de seus significados possíveis, para o contexto deste estudo, isto é, de robótica e automobilística, refere-se a algoritmos que convertem tarefas específicas de alto nível executados por humanos em descrições de baixo nível que detalham como fazer um movimento. Para esse desafio, existem algoritmos que abrangem dois tipos de planejamento: planejamento de caminho e planejamento de trajetória (BENEVIDES, 2015).

O planejamento de trajetória objetiva determinar uma rota e calcular uma função com características temporais que satisfaça o movimento do objeto ao longo do percurso definido. Por outro lado, tem-se o foco de estudo deste trabalho: planejamento de caminho. Este tipo tem o intuito de encontrar uma rota que faça o objeto partir de uma posição inicial até atingir o destino almejado, respeitando as condições e limitações geométricas/espaciais (BENEVIDES, 2015).

É fundamental que o veículo tenha conhecimento da sua posição e do ambiente que ele está inserido para que seu sistema interno consiga determinar o seu deslocamento entre um ponto inicial e seu destino final. Desse modo, o algoritmo de planejamento de caminho tem como intuito resolver o problema de navegação do veículo, uma vez que esse tipo de algoritmo busca definir os pontos navegáveis ou não ao adicionar a ele as descrições

da configuração do ambiente e, bem como, as características cinemáticas do veículo. Respeitando, assim, todos os limites existentes no problema e, principalmente, encontrando o caminho ótimo através da sequência de estados caracterizados obrigatoriamente por não possuir colisão com obstáculos do cenário de navegação.

Portanto, para conseguir determinar a solução mais eficiente, uma das técnicas consiste na amostragem de diferentes possibilidades e compará-las para encontrar o caminho ótimo, podendo ser inseridos critérios de solução, ou seja, condicionar que o algoritmo encontre o trajeto mais rápido ou de menor gasto de energia ou de menor extensão.

A fim de compreender de forma clara os algoritmos de planejamento, existem conceitos principais e básicos que são utilizados no planejamento, sendo alguns deles (LAVALLE, 2006):

- Estado: todas as possíveis situações e configurações estão contidas em um espaço de estado, logo, um estado pode representar a posição e orientação do veículo seja no espaço de estado discreto ou contínuo. É importante, portanto, inicialmente definir a dimensão deste espaço de estado para o planejamento.

- Critérios: para executar as ações necessárias, o planejamento também pode ser submetido a diferentes critérios para gerar o resultado desejado, como por exemplo, viabilidade, a qual não foca na eficiência, e otimização, que objetiva encontrar a solução ótima.

- Plano: a sequência de ações, o comportamento e a estratégia de tomada de decisão são especificadas no plano.

Neste contexto, define-se um espaço de estado formado pelo conjunto de possíveis transformações para um determinado corpo (um veículo) como o espaço de configuração $\mathcal{C}$, possuindo a posição e orientação do corpo (LAVALLE, 2006). Por outro lado, o espaço $\mathcal{O}$ trata-se daquele que contém todos os obstáculos no espaço do mundo $\mathcal{W}$, sendo que, então, o planejamento objetiva encontrar o caminho livre de colisão entre os estados inicial e final em $\mathcal{W}$ (BARLAND, 2012).

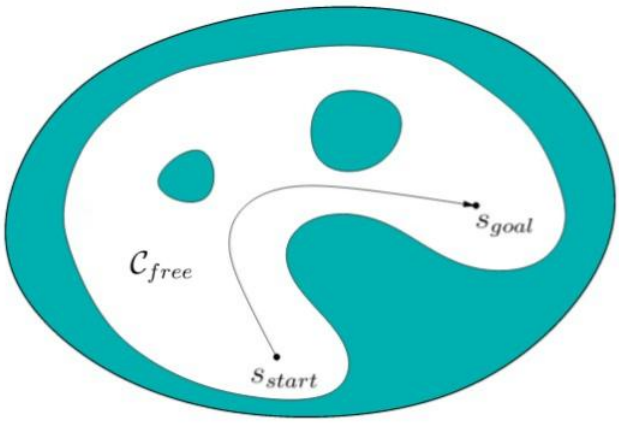
Assim, o espaço de configuração $\mathcal{C}$ do objeto de estudo está diretamente relacionado com a quantidade de graus de liberdade n que o mesmo possui, visto que as transformações realizadas são de dimensão n também. Para o caso de um corpo rígido bidimensional ($n = 2$), o espaço de configuração é o SE(2) definido por $\mathcal{C} = \mathcal{R}^2 \times \mathcal{SO}(2)$, possuindo os vetores de estado $[x, y, \theta]$. Já para o tridimensional ($n = 3$), é o SE(3), o qual, definido por $\mathcal{C} = \mathcal{R}^3 \times \mathcal{R}^3\mathcal{SO}(3)$, possui o vetor de posição $[x, y, z]$ e o de orientação dado por um quatérnio unitário $[qx, qy, qz.qw]$ (LAVALLE, 2006).

Além disso, vale explanar as definições que servem como base para as etapas de

desenvolvimento do planejamento de um veículo articulado com uma carreta (ANDREN et al., 2017):

- O estado $s = (x, y, \theta_1, \theta_2)$ representa o veículo completo por possuir a posição $(x, y) \in \mathcal{R}^2$ e a orientação em relação ao mundo $\theta_1 \in [0, 2\pi)$ do trator, e a orientação em relação ao mundo da carreta $\theta_2 \in [0, 2\pi)$;
- O espaço livre $\mathcal{C}_{free}$ representa o conjunto de todos os estados sem colisão com as áreas proibidas;
- O estado $s_{start} \in \mathcal{C}_{free}$ trata-se do estado inicial do veículo;
- O estado s_{goal} representa o objetivo desejado definido pela posição e orientação do trator e orientação da carreta.

Figura 7 – Definição básica do problema de planejamento buscando uma rota livre de colisão entre os estados inicial s_{start} e o final s_{goal}, em que a área branca corresponde ao espaço livre $\mathcal{C}_{free}$, a área azul representa os obstáculos $\mathcal{O}$ e o espaço do mundo engloba as duas áreas $\mathcal{W} = \mathcal{C}_{free} \cup \mathcal{O}$.



Fonte: Adaptado de LaValle (2006).

Desse modo, a definição básica do problema de planejamento pode ser ilustrada pela Figura 7, utilizando-se dos conceitos listados anteriormente. A maioria dos algoritmos existentes para planejamento de caminho tem em comum o uso de um grafo que conecta os possíveis estados do objeto para, assim, realizar a busca do caminho entre um estado inicial, s_{start}, e o destino desejado, s_{goal}. E depois aplica-se o modelo matemático do veículo para procurar novos estados amostrados no grafo (ANDREN et al., 2017).

2.3.2 Dubins E Reeds Shepp

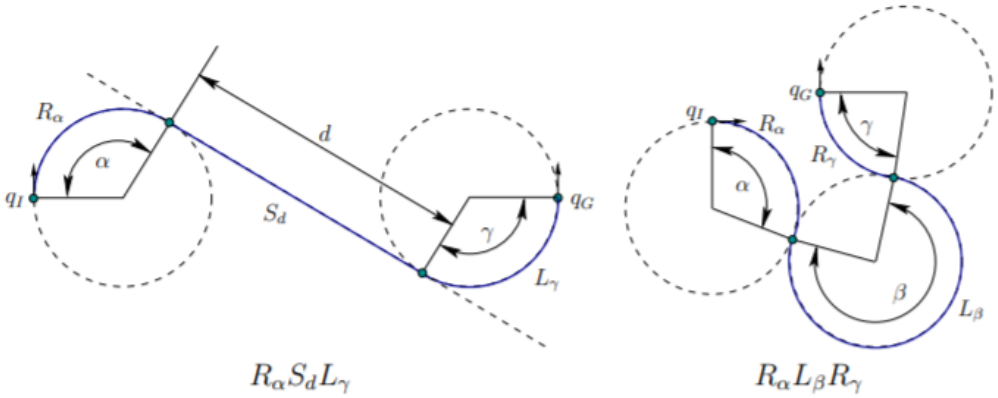
Dentro do universo de modelos de planejamento de caminho para veículos com rodas, aviões e veículos aquáticos, existem na literatura duas espécies de espaço de estado

que caracterizam de modo peculiar o caminho a ser performado pelo objeto de estudo: Dubins e Reeds Shepp.

Segundo LaValle (2006), o caminho do tipo Dubins, na geometria, refere-se à menor curva que conecta dois pontos no plano bidimensional Euclidiano x-y com uma restrição na curvatura do caminho e com tangentes inicial e final predefinidas, e ao mesmo tempo assume-se que o veículo move-se apenas para frente.

O sistema Dubins possui a restrição no ângulo de esterçamento (do inglês "*steering angle*") máximo ϕ_{max}, resultando em um raio da curva mínimo ρ_{min}. Dessa forma, um veículo com este espaço de estado possui três primitivas de movimento: virar à direita (R), seguir reto (S) e virar à esquerda (L), as quais podem ser combinadas em sequências de trios que caracterizam o menor caminho entre duas configurações (Figura 8). Esses trios compõem palavras representando as curvas Dubins: $LRL, RLR, LSL, LSR, RSL, RSR$.

Figura 8 – Curvas Dubins para as sequências RSL (à esquerda) e RLR (à direita)



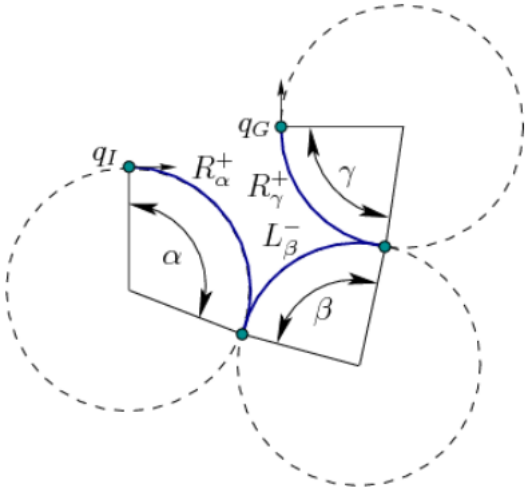
Fonte: LaValle (2006).

Caso o veículo também consiga percorrer em marcha à ré, trata-se de um caminho do tipo Reeds Shepp. Os demais critérios e restrições do Dubins aplicam-se para o Reeds Shepp. A diferença encontra-se na quantidade final de primitivas de movimento permitidas para o sistema, passando a ser 6: virar à direita movendo adiante (R^+) ou para trás (R^-); seguir reto movendo adiante (S^+) ou para trás (S^-); e virar à esquerda movendo adiante (L^+) ou para trás (L^-). Assim, o menor caminho entre duas configurações pode ser definido pela combinação da sequência de 3 a 5 dessas primitivas (Figura 9), semelhantemente às palavras para Dubins (LAVALLE, 2006).

2.3.3 Árvore Aleatória De Exploração Rápida (RRT)

O algoritmo de planejamento de movimento ou caminho caracteriza-se principalmente pela estratégia empregada, havendo na literatura uma ampla variedade de métodos de planejamento de caminho, como, por exemplo, o algoritmo de campo potencial artificial,

Figura 9 – Curvas Reeds Shepp para a sequência $R_{\alpha}^{+}L_{\beta}^{-}R_{\gamma}^{+}$



Fonte: LaValle (2006).

PRM (do inglês, *Probabilistic Roadmap Method)*, método de decomposição de células e entre outros, cujos detalhes podem ser conferidos em (LAVALLE, 2006).

Para espaços de configuração de grandes dimensões, uma estratégia eficiente utilizada consiste no chamado planejamento de caminho baseado em amostragem, no qual o espaço de configuração livre de colisão é amostrado e, posteriormente, é feita a conexão entre dois pontos que constroem um segmento livre de colisão. Dessa forma, há a vantagem do tempo de resolução desse método não depender exponencialmente da dimensão do espaço $\mathcal{C}$.

Nesta classe de técnicas de amostragem, encontra-se o algoritmo RRT (do inglês, *Rapidly-exploring Random Tree*), capaz de produzir amostras do espaço $\mathcal{C}$ e as armazenar em estruturas de dados representando-as como nós em árvores, formando, dessa maneira, caminhos através da conexão entre esses nós (PRADO, 2013). Além de possuir características semelhantes aos demais métodos existentes de planejamento aleatório, LaValle (1998) destaca a estrutura de dados aleatória RRT e sua eficiência, particularmente, por ter sido projetada para sistemas não-holonômicos e com elevados graus de liberdade. Em comparação com o PRM, o RRT apresenta a vantagem de ser um algoritmo mais simples, resultando em respostas mais rápidas, uma vez que não é necessário realizar as dezenas de milhares de conexões entre pares de configurações. Além disso, o RRT escolhido para este trabalho é amplamente empregado em projetos de estudo sobre planejamento de caminho para veículos, devido a maior eficiência em sistemas não-holonômicos e dinâmicos mais complexos, comuns nas áreas de robótica e similares.

O algoritmo RRT constrói uma árvore T de possíveis caminhos, objetivando encontrar uma rota contínua partindo de um estado inicial $x_{init} \in \mathcal{C}_{free}$ até atingir o estado final x_{goal}. De acordo com LaValle (2006), no algoritmo RRT, a cada iteração realizada,

amostra-se uniformemente um estado aleatório x_{rand} no espaço livre de configuração X_{free}. Em seguida, aplica-se um método de busca por amostragem para encontrar uma nova configuração x_{near} mais próxima de x_{rand}, considerando uma métrica ρ, geralmente atribuída à distância euclidiana em relação ao estado x_{rand}.

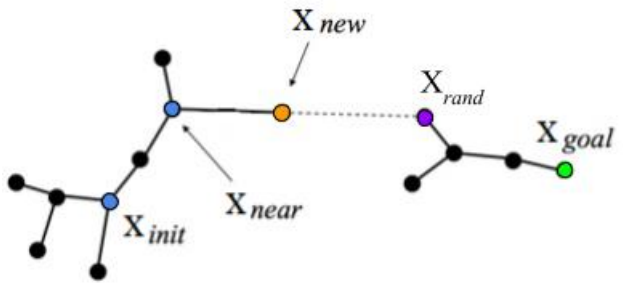
A árvore, portanto, é formada por ramos, os quais são compostos através da união de dois estados válidos que formam a área, que se estendem a diversos pontos de destino gerados aleatoriamente. Para o determinado estado x_{rand}, amostra-se um vetor de entradas aleatórias u, pertencente ao conjunto de entradas possíveis U, sendo que cada uma delas é avaliada no processo de expansão, isto é, de movimento de x_{near} dentro de um intervalo de tempo Δt fixo. Geram-se, assim, novos estados a partir de x_{near}, dentre os quais, é selecionado aquele livre de colisão e que apresente o menor custo à x_{rand} de acordo com ρ, chamado de x_{new}. Este estado e o novo ramo (x_{near}, x_{new}) são adicionados, respectivamente, aos vértices K existentes e aos ramos da árvore T. Esse processo é repetido enquanto não for encontrado um estado x_{new} que seja suficientemente próximo ao estado desejado x_{goal} (SERRANTOLA, 2018). O pseudo-algoritmo básico do RRT pode ser visto na Figura 10 e a demonstração gráfica do processo incremental de busca por amostragem do espaço X_{free} é mostrada na Figura 11.

Figura 10 – Algoritmo básico RRT

GENERATE_RRT($x_{init}, K, \Delta t$)
1 $\mathcal{T}$.init(x_{init});
2 **for** $k = 1$ **to** K **do**
3 $x_{rand} \leftarrow$ RANDOM_STATE();
4 $x_{near} \leftarrow$ NEAREST_NEIGHBOR($x_{rand}, \mathcal{T}$);
5 $u \leftarrow$ SELECT_INPUT(x_{rand}, x_{near});
6 $x_{new} \leftarrow$ NEW_STATE($x_{near}, u, \Delta t$);
7 $\mathcal{T}$.add_vertex(x_{new});
8 $\mathcal{T}$.add_edge(x_{near}, x_{new}, u);
9 Return $\mathcal{T}$

Fonte: LaValle (1998).

Figura 11 – Processo de busca incremental executado pelo RRT



Fonte: Adaptado de Bozcuoglu (2020).

Com isso, o resultado é uma árvore densa, como ilustra a Figura 12, com capacidade de explorar pontos próximos à totalidade do espaço de trabalho livre de colisão. Nota-se

que quanto maior o número de iterações ou tentativas, maior é o espaço total explorado e, consequentemente, maiores são as chances do algoritmo encontrar o caminho ótimo. Logo, uma das vantagens do RRT é a sua capacidade de reduzir o tempo de execução através dessa estratégia de exploração aleatória do espaço (BARLAND, 2012).

Figura 12 – Capacidade de exploração do espaço pelo RRT com o aumento de iterações.

Fonte: LaValle (2006).

A ampla aplicabilidade em diferentes problemas do algoritmo RRT deve-se às suas propriedades principais (LAVALLE, 1998):

- A árvore do RRT tem grande tendência a se expandir em direção às áreas não exploradas;
- Apresenta um comportamento consistente;
- O algoritmo é probabilisticamente completo, sob condições gerais;
- Consiste em algoritmo simples, contribuindo para as análises;
- Mesmo para uma quantidade mínima de ramos da árvore, os estados permanecem conectados;
- Pode ser adaptado e incorporado de diferentes maneiras para diversos problemas de planejamento;

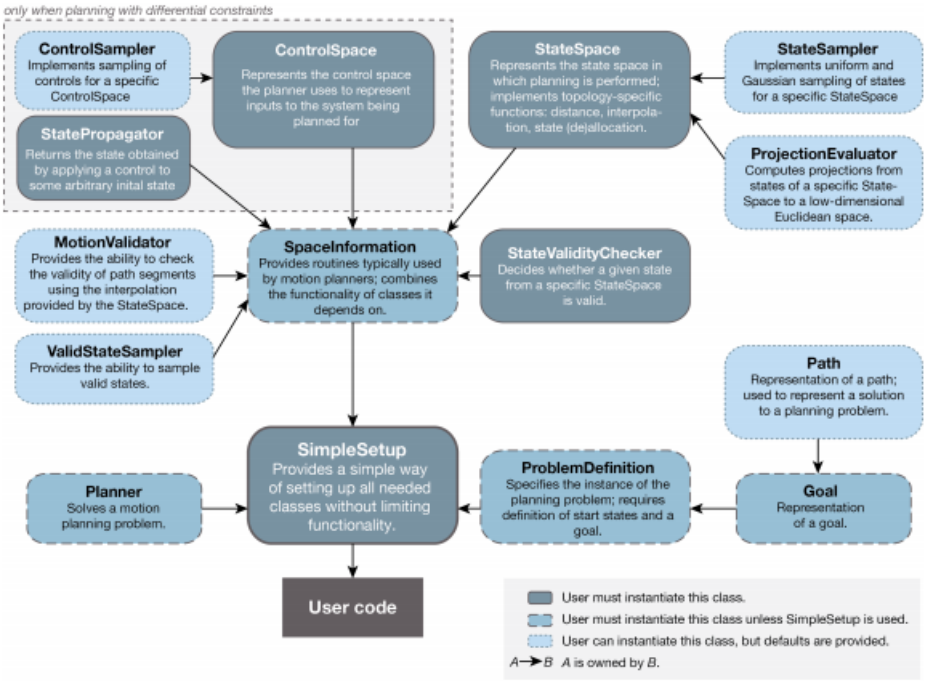
2.3.4 Planejamento Com OMPL

A OMPL, *Open Motion Planning Library*, desenvolvida pelo Kavraki Lab da Rice University, consiste em uma biblioteca com diversos algoritmos de planejamento de movimento baseados em amostragem e com amplas funcionalidades para dar suporte aos usuários nos problemas de planejamento de caminho. Além da terminologia geral citada na Seção 2.3.1, a OMPL possui outros conceitos adicionais, como mostra resumidamente a Figura 13, dos quais vários blocos podem ser personalizados pelo usuário. Há a flexibilidade

de criar novos componentes a partir daqueles disponibilizados pela OMPL, caso não haja a estrutura necessária para a particular aplicação.

Dentro do ambiente da biblioteca, a mesma não abrange códigos relacionados à detecção de colisão e à visualização. Deste modo, há uma grande liberdade de integração com outras ferramentas disponíveis para visualizações 3D e para realizar a detecção de colisão. Por outro lado, a biblioteca fornece funções e classes representativas de alguns dos conceitos principais de planejamento, como espaços de estados e controle, verificação de estados válidos, amostragem, diferentes planejadores e definição do objetivo (BARLAND, 2012). Além disso, esta biblioteca possui interface com linguagens de programação C++ e Python.

Figura 13 – Interface de aplicação de programação da OMPL (API)

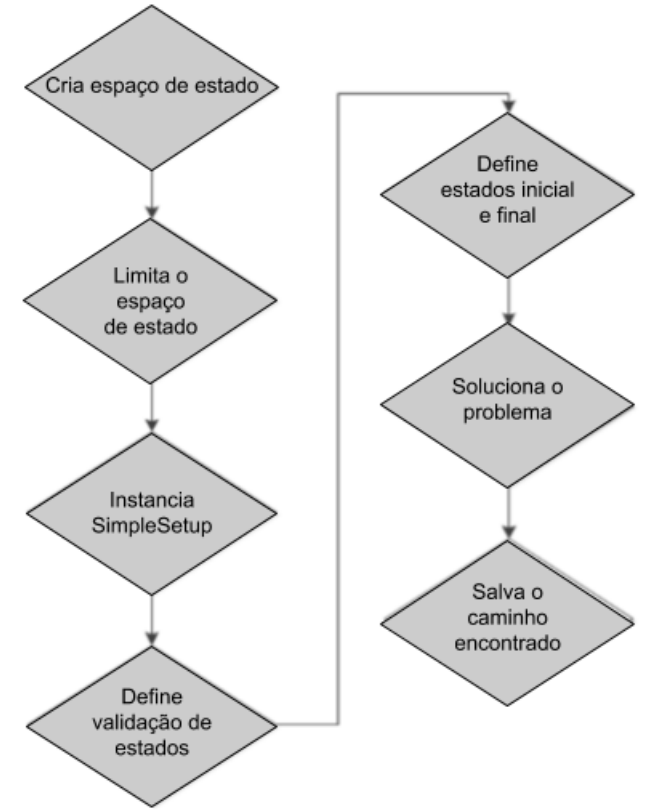


Fonte: Kavraki Lab (2020).

Devido a esta variedade de funcionalidades disponíveis e dedicadas para problemas de planejamento de caminho e devido à flexibilidade de adaptação do sistema para um caso de estudo específico, este projeto, portanto, foi desenvolvido utilizando a OMPL. Desse modo, para elaborar o algoritmo de planejamento, a princípio, seguiu-se um fluxograma básico com os passos necessários para estruturação do problema de planejamento de caminho, mostrado na Figura 14.

Como o fluxograma mostra, o primeiro passo a ser definido no algoritmo consiste na declaração do espaço de estado no qual o usuário deseja realizar o planejamento, bem como os limites desse espaço, e, com isso, define-se o espaço de informação que armazena todas as informações sobre o espaço onde o planejamento vai ser aplicado. Porém, o usuário tem a opção de empregar a classe `SimpleSetup` que ajuda a simplificar este processo

Figura 14 – Fluxograma básico para problema de planejamento de caminho



Fonte: Adaptado de Barland (2012).

de configuração e até o resultado final do algoritmo, uma vez que ele é capaz de definir parâmetros padrões para as classes obrigatórias do planejamento, caso o usuário não o faça. Logo, para este projeto, foi utilizada essa classe a fim de tentar simplificar a solução final do algoritmo objetivando aprimorar o caminho encontrado.

Uma etapa fundamental para conseguir determinar o caminho é a validação de estados, sendo que, no caso da OMPL, se nenhum método for definido para esse tipo de verificação, todos os estados serão considerados como válidos. No entanto, para este projeto busca-se definir que o estado é válido se o veículo articulado não teve colisão geométrica com ele mesmo (entre veículo motor e carreta) e com obstáculos do ambiente (a explanação sobre a detecção de colisão é feita na Seção 2.3.5), e se está dentro dos limites do espaço de estado preestabelecidos.

O problema a ser resolvido deve ser definido ao informar os estados inicial e final e escolher o tipo de planejador para, finalmente, resolver o problema desejado e armazenar o caminho encontrado em um arquivo ou exibir no terminal.

A OMPL apresenta algoritmos de planejamento que podem ser classificados de duas maneiras diferentes: planejadores geométricos e planejadores baseados em controle. Os primeiros basicamente consideram que qualquer rota viável tem capacidade de ser

uma trajetória dinâmica viável, uma vez que tais planejadores possuem apenas restrições geométricas e cinemáticas no sistema, podendo exemplificá-los com PRM, RRT e KPIECE (do inglês, *Kinematic Planning by Interior-Exterior Cell Exploration*). Por outro lado, os planejadores baseados em controle têm a característica de realizar a propagação do estado, visando manipular os efeitos dos controles definidos nos estados do sistema de estudo. Assim, estes baseados em controle apresentam em sua configuração restrições diferenciais, sendo um exemplo também o RRT adaptado (SERRANTOLA, 2018).

2.3.5 Detecção De Colisão

Um tópico extremamente relevante para o planejamento de caminho trata-se de delimitar o espaço de livre colisão dentro do espaço de trabalho do objeto de estudo. O objetivo de atingir o destino desejado é bem sucedido se o caminho encontrado é composto por segmentos entre estados discretos livres de violação. Ou seja, é preciso satisfazer duas condições: o estado do veículo deve obedecer $s \in \mathcal{C}_{free}$ e também garantir que, para o estado s, as demais variáveis do modelo matemático do veículo foram satisfeitas (LAVALLE, 2006).

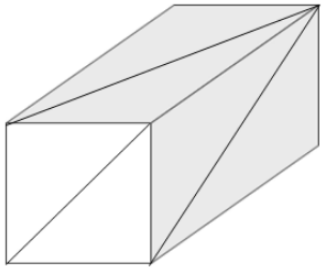
Logo, para a primeira condição citada, é necessária a implementação das relações espaciais ou de proximidade entre dois objetos geométricos, sendo que, para tal, comumente emprega-se um conjunto de consultas de proximidade (LARSEN et al., 2000):

- Detecção de colisão: considerando dois ou mais objetos geométricos, é verificado se ocorreu contato geométrico entre eles;
- Distância de separação: para dois objetos desconexos, calcula-se a distância mínima Euclideana entre eles;
- Distância aproximada: considerando um valor de tolerância do erro, calcula-se a distância aproximada de separação dentro da tolerância predefinida.

Na prática, citam-se duas bibliotecas de programação em C++ que têm a estrutura necessária para implementar a funcionalidade de detecção de colisão no algoritmo de planejamento e que são compatíveis com a OMPL. São elas: PQP.h e FCL.h (GAMMA,).

A Biblioteca de Consulta de Proximidade (do inglês *Proximity Query Package*), conhecida como PQP.h, performa os três tipos de consultas citados anteriormente em pares de modelos geométricos compostos por triângulos que representam o objeto de estudo, como consta o exemplo na Figura 15. Logo, é possível construir os chamados `PQP_models`, que são criados a partir da definição das coordenadas x, y e z de cada triângulo que compõe o objeto, adicionando, assim, a característica de dimensões e geometrias reais no algoritmo de planejamento (GAMMA, 1999).

Figura 15 – Objeto representado por conjunto de triângulos pela biblioteca PQP.h



Fonte: Barland (2012).

Assim, a colisão é detectada quando dois ou mais triângulos se sobrepõem, sendo que, para a PQP.h identificar a localização dos `PQP_models` e conseguir checar ocorrência de colisão, é necessário que o usuário informe também a matriz de rotação e o vetor de translação referentes a cada modelo no espaço tridimensional (BARLAND, 2012).

A FCL.h (do inglês *Flexible Collision Library*) também consegue realizar os três tipos consultas de proximidade entre pares de modelos geométricos formados por triângulos. Mas, se comparada à PQP.h, a FCL.h inclui também outros tipos de consultas e possui diferentes formatos de objetos, como caixa, esfera e cone, o que acaba facilitando o processo de configuração da geometria do objeto de estudo. A mesma necessita igualmente da definição da matriz de rotação e do vetor de translação de cada modelo (GITHUB, 2020).

Destaca-se a funcionalidade da FCL.h de possuir um algoritmo *broadphase* que pode ser usado para detecção de colisão e para checagem de distância de separação entre dois grupos de modelos. Através deste algoritmo, é possível evitar um processo de grande complexidade nesta etapa do planejamento, ao simplificar o modo de operação para verificar a colisão entre dois ou mais objetos em um mesmo ambiente de navegação (GITHUB, 2020).

2.3.5.1 Transformações 3D

Dentro da aplicação das funcionalidades providas pelas bibliotecas de detecção de colisão, torna-se necessário definir determinados parâmetros relacionados ao movimento do veículo para que a detecção seja executada correta e precisamente. Conceitos estes que envolvem transformações geométricas 3D de um corpo rígido, visto que, para este trabalho, assumiu-se a importância de fazer a inserção das dimensões tridimensionais do veículo articulado para um planejamento mais próximo da realidade (LAVALLE, 2006).

A translação 3D por x_t, y_t, $z_t \in \mathcal{R}$ resulta basicamente em:

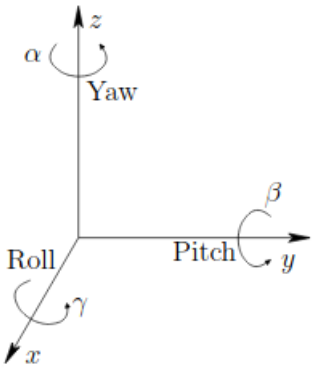
$$(x, y, z) \mapsto (x + x_t, y + y_t, z + z_t)$$

Já a rotação 3D pode ser feita em relação aos três eixos ortogonais x, y e z,

possuindo terminologias como rolagem (do inglês *roll*), arfagem (do inglês *pitch*) e guinada (do inglês *yaw*), respectivamente (Figura 16). Neste trabalho, considera-se que a rotação tridimensional é realizada apenas em relação ao eixo z, tendo, portanto, uma matriz de rotação, em sentido anti-horário, com ângulo α, em radianos, dada por (LAVALLE, 2006):

$$R_z(\alpha) = \begin{pmatrix} cos\,\alpha & -sen\,\alpha & 0 \\ sen\,\alpha & cos\,\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Figura 16 – Rotações tridimensionais *roll*, *pitch* e *yaw*



Fonte: LaValle (2006).

3 METODOLOGIA

Baseando-se nas pesquisas bibliográficas, neste capítulo são mostrados os métodos aplicados na execução de cada etapa deste projeto, abordando desde a definição do modelo cinemático do veículo articulado até as bibliotecas de programação empregadas para o desenvolvimento do algoritmo final. Ademais, são abordadas e explanadas as decisões técnicas tomadas durante o desenvolvimento do projeto.

3.1 Modelo Geométrico E Cinemático Do Veículo Articulado

Neste trabalho, foi considerado o modelo de veículo com apenas um trailer, tendo como articulação simples um engate de comprimento d_1, que conecta os centros dos eixos do trator e da carreta. Considera-se, então, o modelo cinemático explanado na seção anterior, vide Figura 17, cujas equações são dadas por:

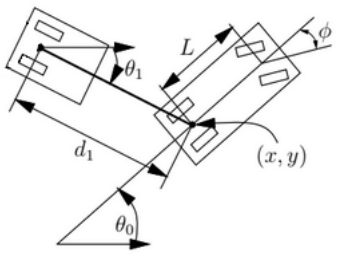
$$\dot{x} = v\,cos\,\theta_0$$

$$\dot{y} = v\,sen\,\theta_0$$

$$\dot{\theta}_0 = \frac{v}{L}\,tan\,\phi$$

$$\dot{\theta}_1 = \frac{v}{d_1} sen(\theta_0 - \theta_1)$$

Figura 17 – Modelo do veículo articulado com uma carreta



Fonte: Adaptado de LaValle (2006).

3.2 Planejamento Para Corpo Rígido

Para implementar um algoritmo específico para o modelo articulado escolhido, as demonstrações e os tutoriais disponíveis no próprio portal da Kavraki Lab (2020) forneceram o embasamento técnico acerca da sintaxe e das funções básicas que geralmente são usados nos algoritmos envolvendo esta biblioteca. O conhecimento sobre a elaboração das etapas principais ocorreu através de um dos códigos demonstrativos (KAVRAKI LAB,

2010b) que contém um planejamento básico para um corpo rígido 3D, isto é, considerando o corpo como um objeto pontual.

O código se trata de um planejamento para um corpo rígido em 3D constituindo-se, basicamente, de uma função responsável por validar os estados, chamada `isStateValid()`, e outra para configuração do planejamento, denominada `plan()`. Há uma rotina inicial `main()`, a partir da qual o código se inicia ao convocar as próximas funções de planejamento.

A rotina principal `plan()`, geralmente, é uma das primeiras funções a se invocar no processo, pois é responsável por inicializar e definir as variáveis do sistema de planejamento do corpo rígido. Isto é, tanto parâmetros relacionados ao corpo (como limites e dimensão do espaço de estados que representará o corpo) quanto parâmetros do desempenho do algoritmo de planejamento (como planejador, tempo de resolução máximo, precisão, etc), além de também definir o estado inicial e o objetivo final do planejamento.

E uma rotina intermediária de validação de estados `isStateValid`, a qual é invocada durante o processo de execução do planejamento de caminho, sendo nela definidas as restrições, condições e relações específicas para o seu tipo de problema a ser resolvido. Ou seja, esta fase do código determina, durante a busca pela trajetória/caminho final, se os estados do corpo gerados pelo algoritmo a cada iteração são válidos ou não para ajudar a compor o objetivo final. Além disso, tal rotina é definida como uma função `booleana`, retornando, assim, 0 ou 1.

O pseudo código exposto no Algoritmo 1) retrata as etapas-chave deste processo de implementação para o algoritmo de planejamento empregando a OMPL.

Algorithm 1 Planejamento com etapas-chave

```
Function main:
    plan();
    return 0;
End Function

Function plan:
    stateSpace ← Create state space;
    ss ← SimpleSetup(stateSpace);
    startState ← Create start state;
    goalState ← Create goal state;
    si ← SpaceInformation(stateSpace);
    isValid ← isStateValid(si, stateSpace);
    planner ← SelectPlanner();
    solved ← solve(maximumTime);
    if solved then
        getSolutionPath()
    else
        print (No solution found)
    end

End Function

Function isStateValid(stateSpace):
    /* Check validity of state using values os position and rotaion from
       stateSpace                                                        */
    condition ← Create condition to validate state;
     if condition then
        isValid ← true;
    else
        isValid ← false;
    end
     return isValid;
End Function
```

3.3 Planejamento Em Espaços De Estados Dubins E Reeds Shepp

A teoria e os conceitos envolvidos nos espaços de estados Dubins e Reeds Shepp, discorridos previamente na Seção 2.3.2, constam também nas funções disponibilizadas pela biblioteca OMPL, possibilitando aplicá-los na estratégia de planejamento. Incrementando a abordagem da elaboração do algoritmo, configurou-se o ambiente físico de navegação do corpo rígido, isto é, simulando a presença de obstáculos diversos e personalizando livremente este cenário.

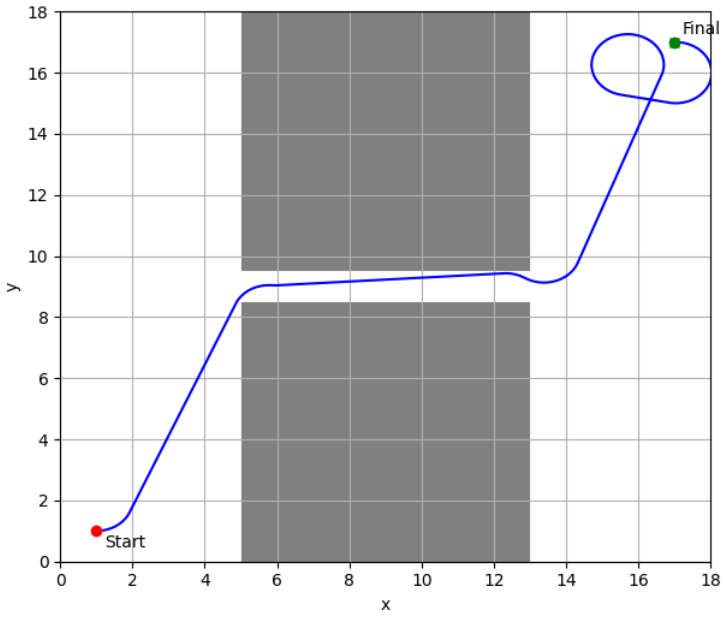
A princípio, tem-se como referência um dos códigos demonstrativos disponíveis no portal da OMPL (KAVRAKI LAB, 2010a), focando em implementar o planejamento empregando `DubinsStateSpace` e `ReedsSheppStateSpace` para solucionar dois tipos de problemas no plano bidimensional, cujas diferenças são os cenários onde o objeto está

inserido.

O primeiro problema consiste em partir do ponto (1, 1) com orientação 0 para chegar ao objetivo final (17, 17) com orientação $-\pi$ rad, sendo, então, um cenário de dois grandes espaços abertos conectados por uma passagem estreita. Já o segundo problema consiste em partir do ponto (0.5, 0.5) com orientação 0.5π rad, objetivando atingir o destino (5.5, 0.5) com orientação 0.5π rad, mas, dessa vez, o cenário se trata apenas de um longo corredor estreito.

Os dois primeiros caminhos obtidos para solucionar o primeiro problema empregam um planejador padrão definido pelo próprio `SimpleSetup`, chamado KPIECE (cuja descrição detalhada pode ser conferida em (ŞUCAN; KAVRAKI, 2009)), o qual se assemelha ao RRT por também se tratar de um algoritmo baseado em amostragem (SUCAN; KAVRAKI, 2012), sendo a Figura 18 correspondente ao caminho com espaço Dubins e a Figura 19, ao caminho com espaço Reeds Shepp.

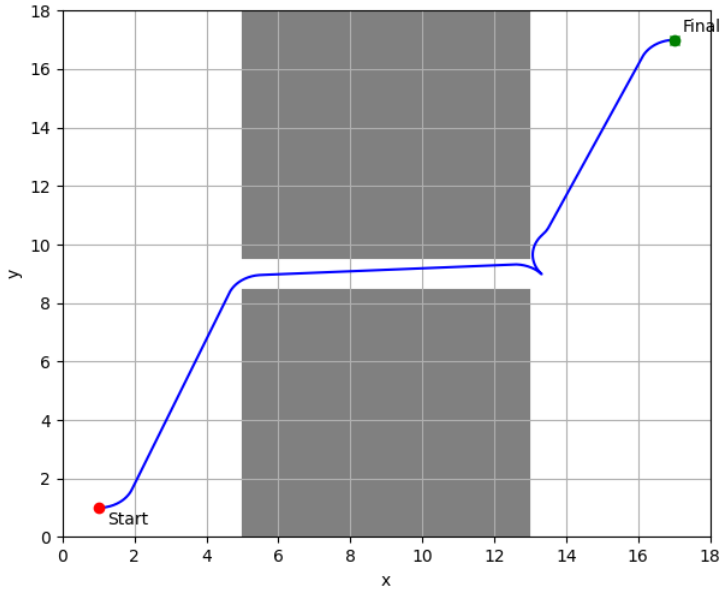
Figura 18 – Caminho obtido no primeiro problema para algoritmo com KPIECE no espaço Dubins



Fonte: Autoria própria.

Pelas Figuras 18 e 19, com planejador KPIECE, nota-se visual e claramente a diferença entre os caminhos, sendo que para Dubins foram feitas mais manobras e curvas do que para o Reeds Shepp. Tendo também a possibilidade de andar em marcha a ré, isto é, a presença de um grau de liberdade a mais no espaço de estados do tipo Reeds Shepp (Figuras 21 e 22) foi um fator que influenciou diretamente no desempenho do algoritmo, uma vez que o RRT não foi capaz de solucionar o problema para o objeto que só possui capacidade de fazer movimentos para frente, como ilustra o "X" da cor verde não alcançado na Figura 20.

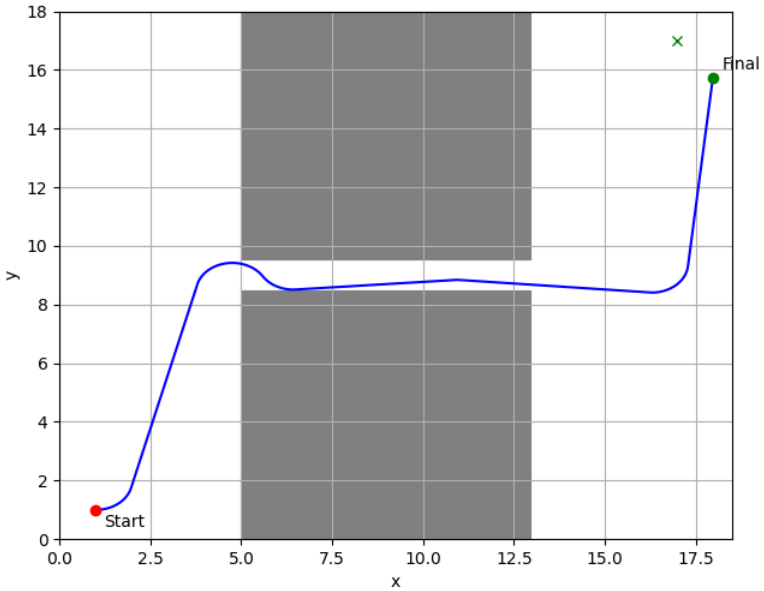
Figura 19 – Caminho obtido no segundo problema para algoritmo com KPIECE no espaço Reeds Shepp



Fonte: Autoria própria.

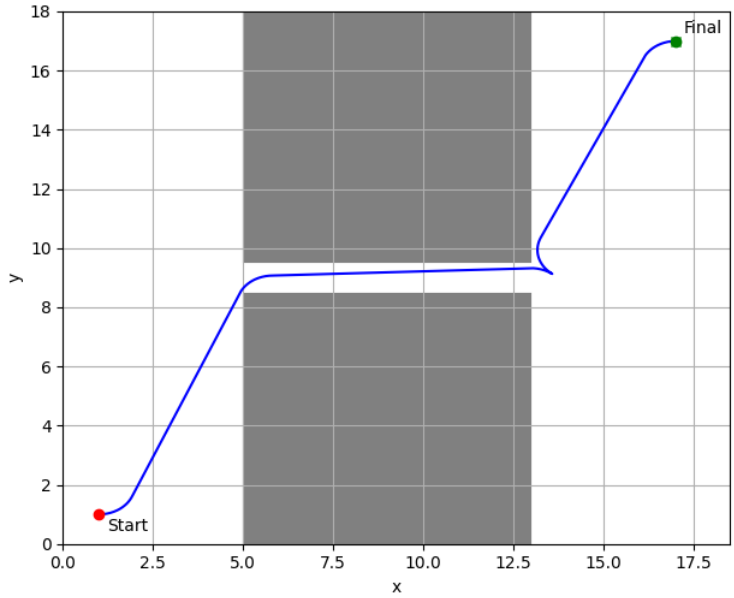
As simulações para os dois tipos de cenários demonstram maior tempo de conclusão do planejamento para os casos que empregam o espaço Dubins, devido às limitações de movimento para resolução de problemas. A característica do movimento para frente e para trás do Reeds Shepp, por meio dos exemplos, é assim validada para ser um parâmetro importante a ser implementado no sistema do veículo articulado.

Figura 20 – Caminho obtido para o primeiro problema usando algoritmo RRT e espaço Dubins



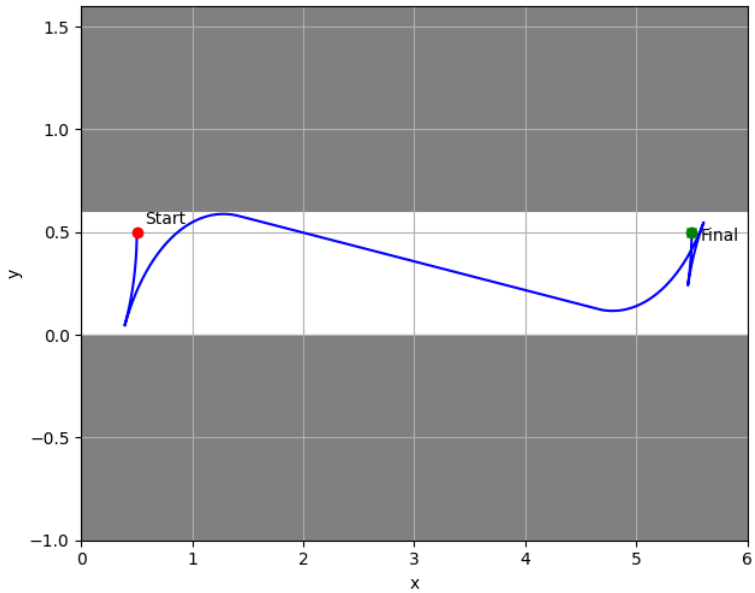
Fonte: Autoria própria.

Figura 21 – Caminho obtido para o primeiro problema usando algoritmo RRT e espaço Reeds Shepp



Fonte: Autoria própria.

Figura 22 – Caminho obtido para o segundo problema usando algoritmo RRT e espaço Reeds Shepp



Fonte: Autoria própria.

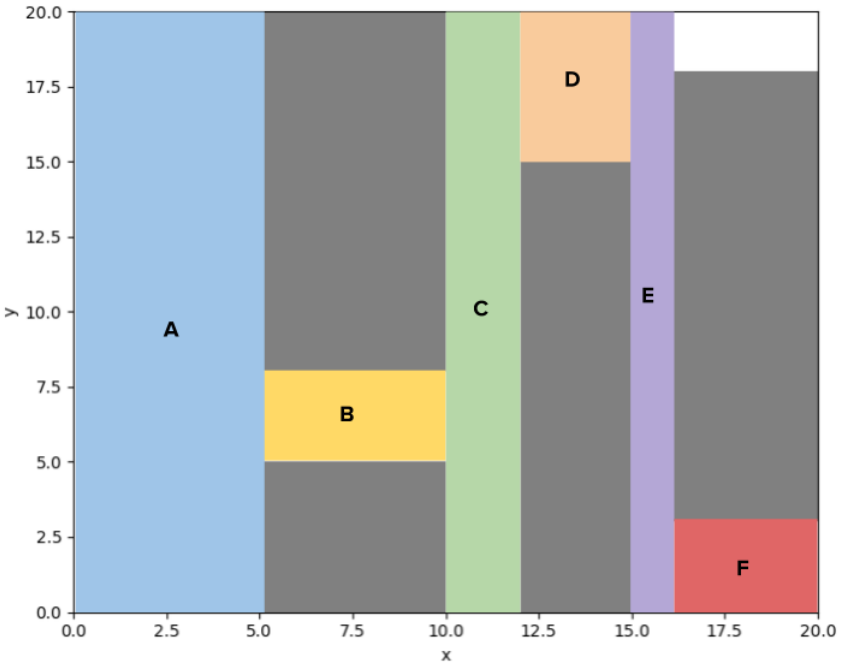
3.3.1 Definindo Obstáculos Personalizados

Considerando as saídas obtidas até então, seria interessante inserir obstáculos de modo personalizado no ambiente de planejamento visto que o código demonstrativo da OMPL (KAVRAKI LAB, 2010a) já exemplificava como definir as áreas de colisões, isto é, a presença de obstáculos no cenário através da imposição de restrições nos valores de

posição x e y do objeto, as quais são respeitadas no processo de planejamento uma vez instanciadas na função de validação dos estados `isStateValid()`, definindo o cenário das Figuras 24 e 25 em que os obstáculos são representados pelos retângulos cinzas e as regiões navegáveis (sem colisão) são delimitados através de um conjunto de expressões condicionais nomeadas de A a F (vide Figura 23):

- Região A - dada por $x < 5$
- Região B - dada por $x > 5$ *e* $x < 10$ *e* $y > 5$ *e* $y < 8$
- Região C - dada por $x > 10$ *e* $x < 12$
- Região D - dada por $x > 12$ *e* $x < 15$ *e* $y > 15$ *e* $y < 20$
- Região E - dada por $x > 15$ *e* $x < 16$
- Região F - dada por $x > 16$ *e* $x < 20$ *e* $y < 3$

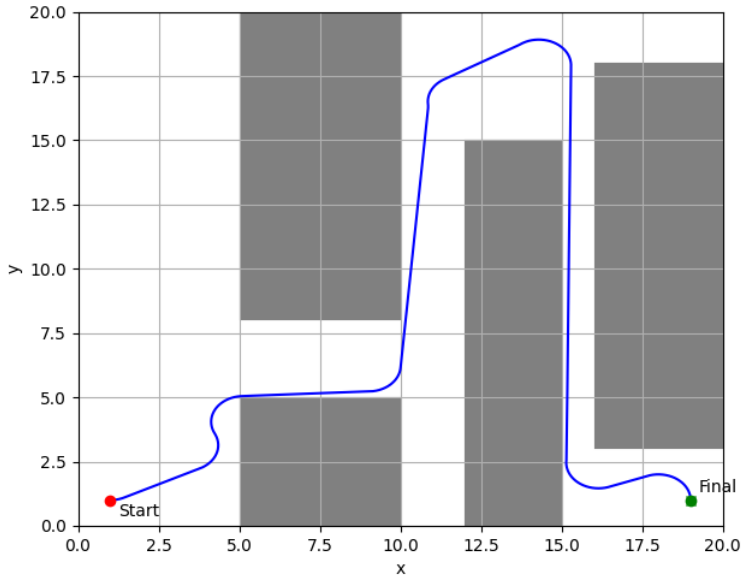
Figura 23 – Cenário customizado através das áreas livres de colisão (nomeadas de A a F) definidas por meio de expressões condicionais



Fonte: Autoria própria.

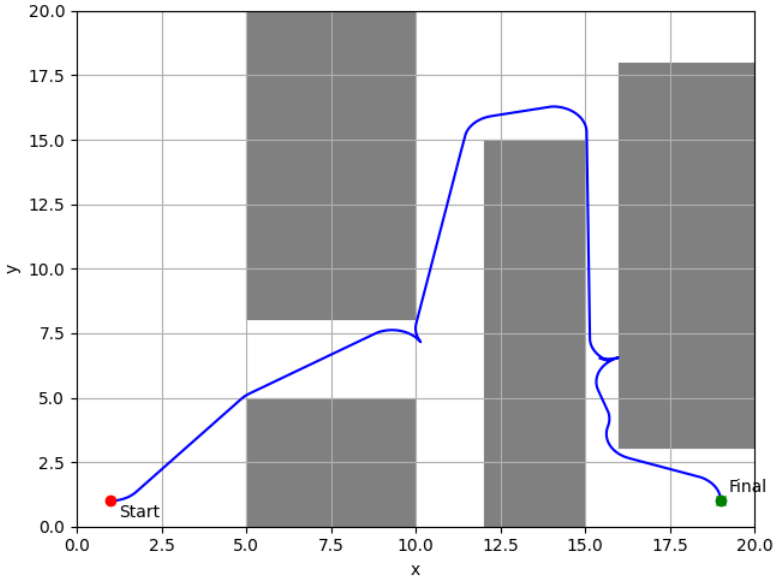
O aumento no grau de complexidade do problema, vide Figuras 24 e 25, permite validar a eficiência do uso do algoritmo RRT para solucionar até mesmo este cenário customizado, seja para o espaço Dubins ou para Reeds Shepp.

Figura 24 – Caminho para cenário customizado usando algoritmo RRT e espaço Dubins



Fonte: Autoria própria.

Figura 25 – Caminho para cenário customizado usando algoritmo RRT e espaço Reeds Shepp



Fonte: Autoria própria.

3.4 Planejamento Para Veículo Articulado Usando FCL.h

Para resolver o problema de planejamento de movimento de um veículo articulado, no algoritmo implementado, a configuração instanciada na rotina `plan()` foi personalizada para espaços de estado do tipo Reeds Shepp, uma vez que, se o veículo não possui deslocamento no eixo z, esta alteração para espaços do tipo SE(2) é adequada. Portanto, o veículo motor (*truck*) é representado pelo espaço de estado Dubins ou Reeds Shepp. E, para inserir a configuração de um veículo articulado com uma carreta ($k = 1$), observou-

se que, pela modelagem cinemática do mesmo da Seção 2.2.2, a carreta inclui, para o sistema como um todo, uma variável: a sua velocidade angular dada por $\dot{\theta}_1$. Além disso, a classe `SimpleSetup` continuou sendo empregada na implementação do algoritmo de planejamento.

3.4.1 Rotina De Planejamento

Deste modo, nota-se que a posição x-y da carreta não influencia no modelo cinemático do veículo articulado. Sendo, assim, conveniente adicionar ao espaço de estado do veículo apenas mais um componente do tipo $\mathcal{R}^1$. Para isso, foi necessário criar um espaço de estado composto, chamado `CompoundState`, que é resultante da soma do espaço Dubins ou Reeds Shepp com o espaço $\mathcal{R}^1$ (`RealVectorStateSpace(1)`) , obtendo no fim um espaço de dimensão 3. Este `CompoundState` é uma das flexibilidades presentes na biblioteca OMPL para criar um ambiente de planejamento qualquer, permitindo definir, individualmente, os limites (`bounds`) e valores iniciais (`start`) e finais (`goal`) de cada espaço de estado que o compõe.

Analisando as equações cinemáticas da Seção 2.2.2, concluiu-se que as variáveis de controle do sistema se resumem em: velocidade v do *truck* e ângulo da direção/esterçamento ϕ. Tais controles são aplicados aos estados através do uso de um planejador baseado em controle, isto é, RRT adaptado para esta categoria. Logo, implementou-se na rotina `plan()` um espaço de controle, `ompl::control::RealVectorStateSpace`, sobre o espaço de estado composto do veículo articulado, para criar as duas variáveis de controle do problema em estudo. Estas também podem ter seus limites definidos como desejado. Vale ressaltar que, para o caso do espaço Reeds Shepp, devem ser inseridos limites de velocidade que abrangem valores negativos, visto que este tipo pode percorrer em marcha à ré.

Juntamente com o espaço de controle criado, foi necessário descrever uma função externa denominada `propagate()`, que tem o intuito de realizar a aplicação dos controles por um período de tempo, chamado `duration`, no estado atual `state` do veículo e, com isso, gerar próximo estado `result`. Destaca-se que é nesta função de propagação onde foram inseridas as equações cinemáticas do veículo articulado com uma carreta, explanadas na Seção 2.2.2, mas utilizou-se a aproximação discreta da função diferencial, considerando, deste modo, um intervalo de tempo (`duration`) pequeno. Ou seja, as equações aplicadas no algoritmo foram descritas da seguinte maneira:

$$
\begin{aligned}
x_0 &= x_{0\,inicial} + v * \texttt{duration} * cos\,(\theta_0) \\
y_0 &= y_{0\,inicial} + v * \texttt{duration} * sen\,(\theta_0) \\
\theta_0 &= \theta_{0\,inicial} + \frac{v}{L} * \texttt{duration} * tan\,(\phi) \\
\theta_1 &= \theta_{1\,inicial} + \frac{v}{d_1} * \texttt{duration} * sen\,(\theta_0 - \theta_1)
\end{aligned}
$$

sendo d_1 a distância entre os centros do veículo motor e da carreta, (x_0, y_0, θ_0) referente ao veículo motor, θ_1 referente à orientação da carreta, L é o comprimento do veículo motor, e `duration` é o tempo de aplicação em segundos.

Ao utilizar a função `propagate()`, surgiram duas novas possibilidades de personalização do ambiente de planejamento: definir os valores mínimo e máximo do `duration` e definir o `stepSize`. Quando os controles são aplicados aos estados iniciais, isso ocorre por um tempo de duração que é um múltiplo inteiro do `stepSize`, dentro dos limites especificados para `duration`.

Outra instância opcional para adicionar à estratégia de planejamento consiste na definição de um objetivo de otimização, sendo, neste projeto, um objetivo interessante o menor comprimento do caminho. Em seguida, foi selecionado o planejador para o algoritmo, sendo que o mesmo é aplicado sobre o espaço de controle para conseguir determinar os controles apropriados, os quais atuando pelas equações, possibilitem o objeto atingir o destino desejado. Finalmente, o tempo máximo de solução para o algoritmo é informado na função `solve()`.

3.4.2 Rotina De Validação De Estados Com FCL

Na rotina de validação de estados `isStateValid()`, foram extraídos individualmente os espaços de estados do veículo motor e da carreta do espaço composto (`CompoundState`) para utilizar tais valores de posição e orientação na etapa de construção dos modelos geométricos da biblioteca FCL.h. Para criar os objetos de colisão (`CollisionObjectd`) da FCL, foram instanciadas, primeiramente, as matrizes de rotação 3D (`Matrix3d`) do veículo motor e da carreta e as transformações 3D (`Transform3d`) de cada objeto a ser considerado no cenário de planejamento (veículo motor, carreta e obstáculos), as quais incluem a matriz de rotação (pela função membro `linear()` da classe `Transform3d`) e o vetor de translação (pela função membro `translation()` da classe `Transform3d`) do respectivo objeto.

Ressalta-se que as matrizes de rotação 3D são calculadas a partir do respectivo ângulo de orientação θ do objeto através uma função `rotationMatrixTheta()` criada separadamente, sendo tais rotações feitas ao redor do eixo cartesiano z, empregando o equacionamento mostrado na Seção 2.3.5.1.

O vetor translação foi preenchido com o valor do centro de cada objeto. Particularmente, para o veículo motor o centro (x_0, y_0) foi definido com os valores de x e y extraídos do espaço de estados Reeds Shepp, enquanto que o centro da carreta foi inserido como equações, uma vez que sua posição (x_1, y_1) depende de sua orientação atual e da posição do veículo motor. As equações usadas para os cálculos foram:

$$x_1 = x_0 - cos\,(\theta_1) * d_1$$

$$y_1 = y_0 - sen\,(\theta_1) * d_1$$

Em seguida, para este projeto, considerou-se conveniente construir a geometria de todos os objetos do cenário de planejamento com a geometria de uma caixa (`Boxd`), podendo facilmente definir as suas dimensões de largura (eixo y), comprimento (eixo x) e altura (eixo z). Portanto, dadas as geometrias (`Boxd`) e as transformações (`Transform3d`) de cada objeto, foi feita a combinação entre seus respectivos pares para obter a instância do objeto de colisão (`CollisionObjectd`). Nos dois cenários simulados neste trabalho, foram criados, dessa maneira, um total de 3 a 5 objetos de colisão no algoritmo, variando apenas a quantidade de obstáculos presentes no ambiente.

Inicializados os objetos, as consultas de proximidade entre eles foi efetuada em duas partes: primeiramente, foram instanciadas a estrutura de dados de pedido de consulta de colisão (do inglês "*collision query request data structure*") e a de resultado da consulta de colisão (do inglês "*collision query result data structure*"); e depois, realizou-se o teste de colisão ao chamar a função de consulta de colisão `collide()`, a qual recebe como parâmetros os dois objetos a serem analisados se ocorreu sobreposição e as estruturas de dados de pedido e de resultado da consulta.

Para o problema de veículo articulado, torna-se necessário que a detecção de colisão seja feita entre todos os obstáculos do ambiente e o veículo articulado, e também, entre o veículo motor e sua carreta. Logo, devido a este nível de complexidade maior, há uma quantidade significativa de combinações a serem consideradas na checagem de colisão. Para facilitar este processo, empregou-se o algoritmo de `broadphase` da FCL, em que foi possível formar dois grupos de objetos: o de obstáculos e o do veículo articulado, e cada um dos grupos foi atribuído a um gerenciador de colisão (do inglês `BroadphaseCollisionManager`). Além disso, foram definidas as estruturas de dados intermediárias que armazenam informações geradas durante o processo de computação `broadphase` para cada função de colisão `collide()` chamada.

Portanto, foram feitas duas chamadas de função de consulta de colisão: uma para verificar colisão entre o grupo de obstáculos e o grupo do veículo articulado, e outra, para verificar entre os objetos que compõem o grupo do veículo articulado. A partir da estrutura de dados obtida ao final de cada checagem, pôde-se extrair a resposta booleana se ocorreu colisão ou não.

Para evitar que o veículo articulado apresente uma configuração inviável, como o *jackknifing*, foi imposta uma restrição nos valores dos ângulos de orientação do veículo motor e da carreta através da condição de que o módulo da diferença entre os ângulos deve ser menor ou igual a 60°. A última linha da rotina do `isStateValid()` foi composta por todas as condições a serem respeitadas pelo algoritmo de planejamento para considerar um estado válido (isto é, igual a 1), sendo elas:

- Resultados booleanos das duas consultas de colisão `collide()` devem ser iguais a 0, indicando que não teve colisão;
- Limites estabelecidos na rotina `plan()` devem ser respeitados;
- O módulo da diferença entre os ângulos θ_0 e θ_1 deve ser menor ou igual a 60°.

Outro fator importante adotado na estratégia de planejamento baseou-se na inserção de um objetivo de otimização do problema, sendo, neste experimento, o tamanho final do caminho encontrado. Existe a desvantagem, no entanto, deste tipo de objetivo de otimização em ter a tendência a encontrar caminhos muito próximos dos obstáculos para conseguir reduzir a sua extensão.

4 RESULTADOS E DISCUSSÃO

Para a fase de testes, objetivando realizar simulações de planejamento de caminho para situações mais realistas e aplicáveis a problemas do cotidiano, utilizou-se como referência dimensional o veículo da Scania G 360 LA6x2 R885, modelo semelhante ao caminhão autônomo desenvolvido pela USPSC da Figura 4, retirando de sua ficha técnica Lectura Specs (2020) informações de suas dimensões. Com isso, as dimensões se resumem na Tabela 1, sendo o comprimento correspondente ao valor L empregado nas equações cinemáticas do veículo.

Tabela 1 – Dimensões Scania G 360 LA6x2 R885.

Parâmetro	**Tamanho [m]**
Comprimento (L)	7.05
Largura	2.60
Altura	3.27

Fonte: Lectura Specs (2020)

Enquanto que a carreta foi pensada para comportar pacotes de encomenda, realizando o serviço de uma transportadora. Com isso, escolheu-se um comprimento de 10 metros e com a mesma largura do veículo motor (2.6 metros). Além disso, para facilitar o experimento e as análises, a configuração do veículo mostrado na Figura 17 foi levemente adaptada considerando as duas seguintes condições:

- O engate possui um comprimento $d_1 = 1000$ cm (*hitch length*) e conecta o centro do veículo motor com o centro da carreta;
- O comprimento L presente no modelo cinemático da Seção 2.2.2 corresponde ao comprimento total do veículo motor, neste caso, do Scania G 360 LA6x2 R885.

Os testes se aplicaram em dois cenários diferentes, porém, ambos retratando situações rotineiras destes veículos em seus trajetos e ressalta-se também que o veículo é representado pelo espaço de estados Reeds Shepp, que foi verificado inicialmente na Seção 3.3, permitindo manobras para frente e para trás, e o planejamento utiliza o algoritmo RRT em ambos experimentos realizados.

Para realizar a validação do algoritmo de planejamento implementado, os resultados foram gerados através da compilação em ambiente Visual Studio Code, usando a linguagem de programação C++, em um computador cuja configuração é Intel(R) Core(TM) i5 de 1.60GHz e com 6GB de memória RAM. Ademais, as estratégias e as metodologias de implementação do algoritmo de planejamento seguem exatamente as descrições contidas na Seção 3.4.

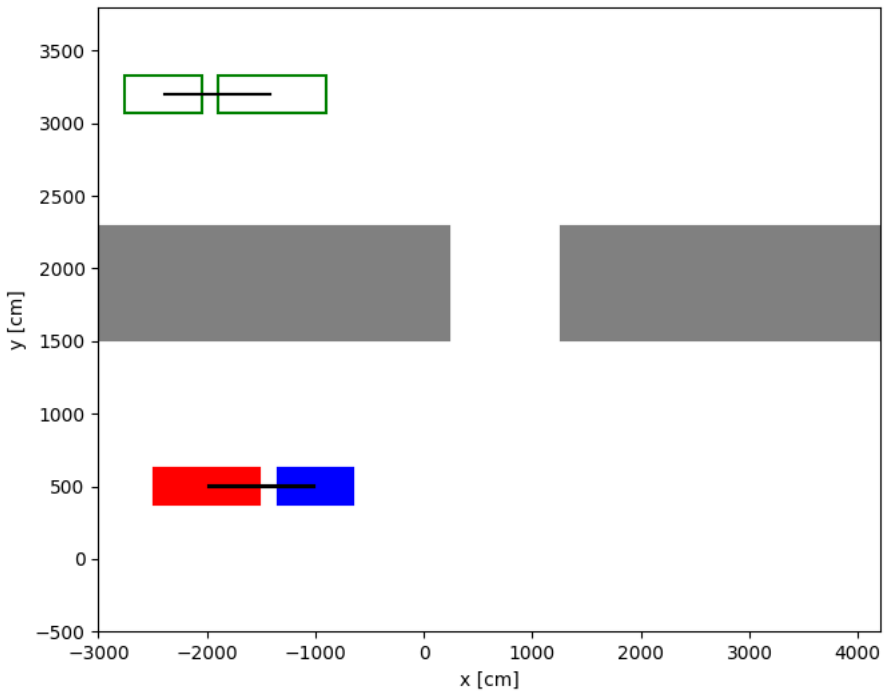
O primeiro experimento, representado pela Figura 26, compõe um problema comum de percorrer por uma passagem estreita e estacionar do outro lado do ambiente, sendo os obstáculos representados pelos retângulos cinzas e o veículo articulado ilustrado pelo veículo motor, em azul, e a carreta, em vermelho. Ressalta-se que os valores do plano bidimensional estão em centímetros, visando, assim, ter mais precisão nos valores aplicados e gerados. A Tabela 2 contém os valores dos pontos inicial e final desejado do veículo referente ao problema deste primeiro cenário, sendo o ponto inicial ilustrado na Figura 26 pelo veículo com formas preenchidas, e o estado objetivado, pelo veículo com apenas o contorno das formas na cor verde. Ressalta-se que os valores x_0 e y_0 correspondem ao centro do veículo motor (*truck*), o ângulo θ_0 corresponde à orientação do *truck* e θ_1, à orientação do *trailer*.

Tabela 2 – Início e objetivo do problema do cenário com passagem estreita

	x_0 **[cm]**	y_0 **[cm]**	θ_0 **[rad]**	θ_1 **[rad]**
Início	-1000	500	0	0
Objetivo	-2400	3200	π	π

Fonte: Autoria própria.

Figura 26 – Cenário para o experimento com passagem estreita



Fonte: Autoria própria.

Já para o segundo experimento, exibido na Figura 27, foi proposta uma complexidade maior devido ao problema de estacionar o veículo articulado em uma vaga determinada. A representação geométrica na imagem dos estados inicial e final desejado é

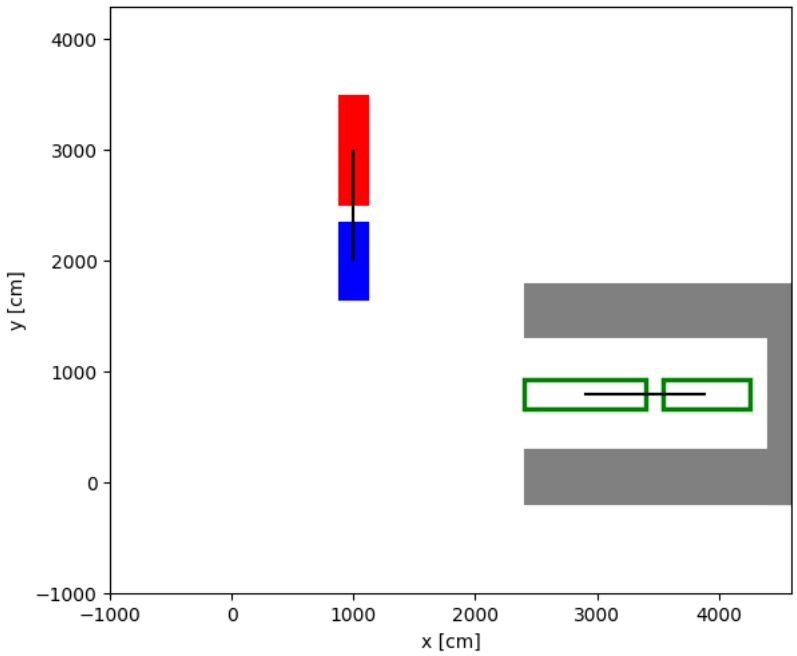
a mesma para o primeiro experimento (Figura 26). Os obstáculos se concentram de modo a formar uma região de destino bem restrita para o veículo adentrar, simulando, assim, um problema de estacionamento de um veículo articulado para fazer carregamento ou descarregamento da carga em um local específico. Os estados inicial e final desejado para este segundo problema estão dispostos na Tabela 3.

Tabela 3 – Início e objetivo do problema do cenário de estacionamento

	x_0 **[cm]**	y_0 **[cm]**	θ_0 **[rad]**	θ_1 **[rad]**
Início	1000	2000	$-\pi/2$	$-\pi/2$
Objetivo	3900	800	π	π

Fonte: Autoria própria.

Figura 27 – Cenário para o experimento com cenário de estacionamento



Fonte: Autoria própria.

Levando em consideração o fato de que o planejador RRT que está sendo empregado neste trabalho baseia-se em uma técnica de amostragem aleatória para conseguir explorar o máximo possível de estados do ambiente de navegação, a etapa de testes possui também resultados referentes às estatísticas obtidas a partir das 100 simulações do mesmo experimento pelo processo de *benchmark*, a fim de coletar um conjunto de resultados maior para gerar análises mais consistentes e precisas. Com isso, analisou-se mais amplamente o desempenho do algoritmo para um determinado problema por meio de parâmetros-chave, como erro, desvio padrão, tempo gasto, quantidade de estados gerados, distância em relação à posição final desejada e extensão da rota.

O algoritmo implementado para o planejamento desta fase de testes encontra-se no Apêndice A, o qual aborda mais especificamente o problema do cenário de estacionamento.

4.1 Experimento Para Cenário Com Passagem Estreita

A área de trabalho no plano bidimensional foi limitada a $x \in [-25, 20.5]$ m e $y \in [-40, 40]$ m, de modo a impor que o veículo circulasse obrigatoriamente pela passagem estreita para chegar no seu destino final. Da mesma forma, limitou-se o ângulo de esterçamento para $\phi \in [-\pi/3, \pi/3]$ rad para evitar o problema de *jackknifing* e a velocidade também foi restringida para, no máximo, 7 m/s, que equivale a aproximadamente 25 km/h, para frente (velocidade positiva) e para trás (velocidade negativa).

Foi possível verificar a relação direta do valor de stepSize na propagação das variáveis de controle no sistema dinâmico do veículo, visto que, para valores maiores que 1, a propagação se dá por períodos de tempo muito grandes, gerando um caminho com manobras bruscas e inviáveis para o veículo real. Já a duração mínima e máxima dos controles está diretamente relacionada com o tempo total necessário para o algoritmo conseguir atingir o destino final e, assim, fornecer uma solução viável. Logo, através de algumas simulações, se aplicado um stepSize menor que 1 e uma duração máxima de controle menor que 4 ou 5 segundos, o tempo de solução precisou ser estendido para mais de 4.5 minutos, visto que a propagação se deu de maneira contínua, porém, a passos muito curtos, o que não é uma característica tão relevante para este projeto.

A partir de uma série de testes iniciais, os parâmetros de stepSize e de mínima e máxima duração (em segundos) de aplicação dos controles no sistema mais adequados para esta análise foram, respectivamente, 0.1 e [1, 10], uma vez que o passo de propagação permitiu obter um caminho contínuo e suave (sem movimentos bruscos), bem como, foi possível que o algoritmo encontrasse uma solução para o problema de planejamento em, no máximo, 30 segundos. Trata-se de uma característica muito importante para a etapa de *benchmark* deste algoritmo, tendo em vista a grande quantidade de testes a serem repetidos nessa parte da análise.

Considerando a aplicação de algoritmo de amostragem aleatória em um espaço de controle com restrições dadas por equações diferenciais, para analisar melhor os resultados gerados no processo de *benchmark*, adotaram-se valores de tolerância para a distância entre a posição final do veículo motor da solução gerada e a posição de destino desejada em 2.5 metros e para a diferença entre os ângulos de orientação θ_0 e θ_1, em 3°.

As 100 simulações de *benchmark* efetuadas estão resumidas como valores estatísticos na Tabela 4. Dessas simulações efetuadas, 77% tiveram resultados dentro dos valores de tolerância considerados, porém, não ocorreu solução exata. Dessa forma, a posição final (x_0, y_0) do truck obtida em cada simulação tem um erro (distância) euclidiano calculado

em relação ao destino final objetivado $(-2400, 3200)$ cm, bem como os erros de orientação final θ_0 (truck) e θ_1 (trailer) em relação à desejada orientação π rad são calculados por meio da equação da diferença.

Os valores de erro final são expostos através de média, desvio padrão, mínimo e máximo na Tabela 5, a qual apresenta os cálculos para todas as 100 simulações de modo a avaliar a performance geral do processo de *benchmark*, e na Tabela 6, que apresenta os cálculos apenas para as 77 simulações encontradas dentro da margem de tolerância.

Sendo o caminho ilustrado na Figura 28 uma das soluções encontradas com os melhores indicadores obtidos, isto é, possui um dos trajetos mais curtos, com uma posição final à cerca de 5 cm de distância do destino e com ângulos de orientação cuja diferença em relação ao valor desejado menor que 1° para θ_0 e menor que 6.5° para θ_1.

Tabela 4 – Resultados estatísticos gerais obtidos com base nas 100 simulações do experimento para cenário com passagem estreita

Parâmetro	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**
Tempo [s]	30.000584	0.000300	30.0002	30.0016
Total de estados gerados	41357.06	4000.75	34198	48850
Estados do caminho	505.52	127.17	236	825
Tamanho do caminho [m]	172.82	48.83	64.45	280.42

Fonte: Autoria própria.

Tabela 5 – Erros de posição (x_0, y_0) final e de orientações θ_0 e θ_1 finais obtidas com base em todas as 100 simulações do experimento para cenário com passagem estreita

Parâmetro	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**
Erro de orientação θ_0 [°]	40.59	55.61	0.61	178.30
Erro de orientação θ_1 [°]	22.26	37.29	0.17	125.29
Erro de posição x_0 [m]	0.47	1.02	0	5.70
Erro de posição y_0 [m]	3.04	6.34	0	19.40
Erro de posição x_0 e y_0 [m]	3.11	6.41	0	20.23

Fonte: Autoria própria.

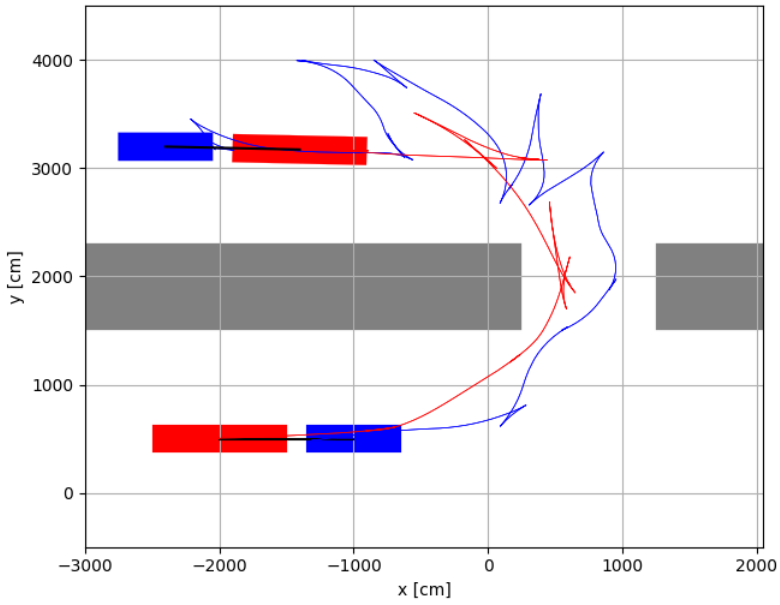
Algumas posições do veículo (com contorno cinza) ao longo do trajeto foram registradas na Figura 29. Além disso, o desenvolvimento do movimento está ilustrado nas Figuras 30, a qual foca apenas no *truck* e no *trailer*, e 31, a qual insere também a articulação (em preto) juntamente, possibilitando verificar que a distância entre o *truck* e o *trailer* se mantém igual a d_1.

Tabela 6 – Erros de posição (x_0, y_0) final e de orientações θ_0 e θ_1 finais obtidas com base nas 77 simulações dentro da margem de tolerância escolhida para a distância entre a posição final do veículo motor da solução gerada e a posição de destino desejada em 2.5 metros e para a diferença entre os ângulos de orientação θ_0 e θ_1, em 3°, do experimento para cenário com passagem estreita

Parâmetro	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**
Erro de orientação θ_0 [°]	21.51	20.19	0.61	79.28
Erro de orientação θ_1 [°]	7.55	5.49	0.17	22.96
Erro de posição x_0 [m]	0.14	0.25	0.00	1.21
Erro de posição y_0 [m]	0.39	0.60	0.00	2.46
Erro de posição x_0 e y_0 [m]	0.43	0.64	0.00	2.46

Fonte: Autoria própria.

Figura 28 – Uma das soluções geradas com melhor desempenho no planejamento para o cenário com passagem estreita

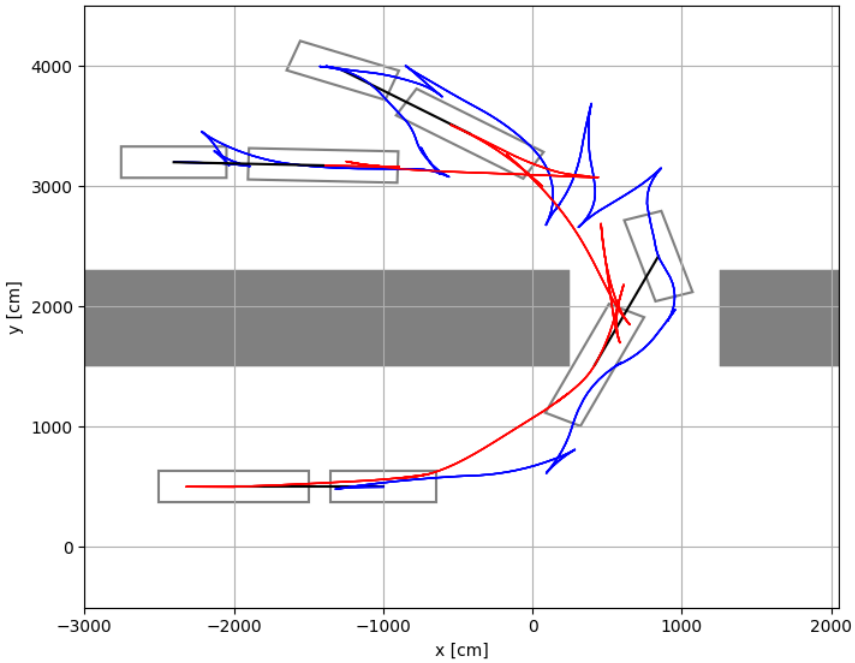


Fonte: Autoria própria.

4.2 Experimento Para Cenário Com Estacionamento

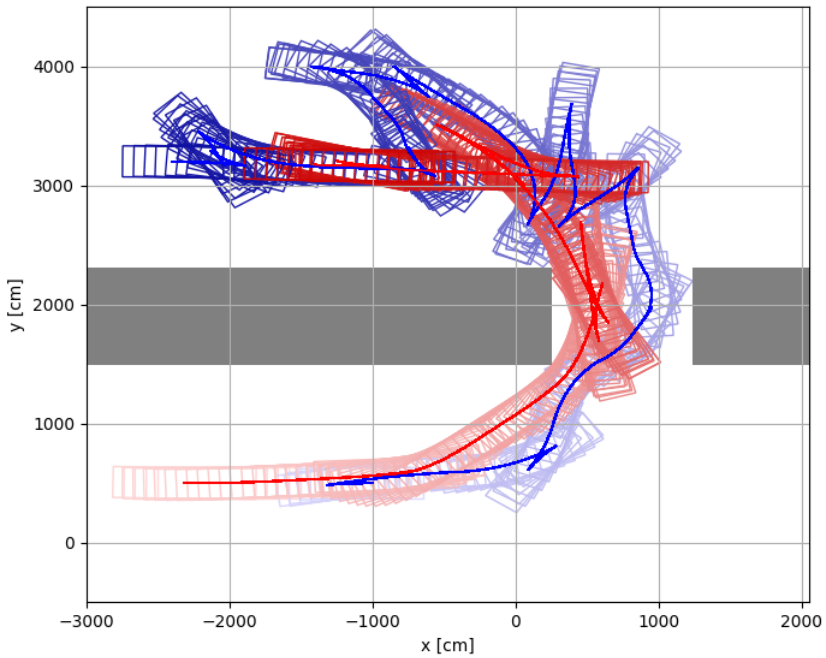
Tratando-se de um problema mais complexo para um veículo articulado, a área de trabalho neste experimento foi ampliada tal que $x \in [-100, 44]$ m e $y \in [-100, 100]$ m para que o objeto de estudo pudesse se deslocar e encontrar um caminho com mais rapidez e sem entraves maiores do que o desafio de conseguir adentrar na vaga de estacionamento com sucesso.

Figura 29 – Posições pontuais do veículo (contorno cinza) em uma das soluções geradas com melhor desempenho no planejamento para o cenário com passagem estreita



Fonte: Autoria própria.

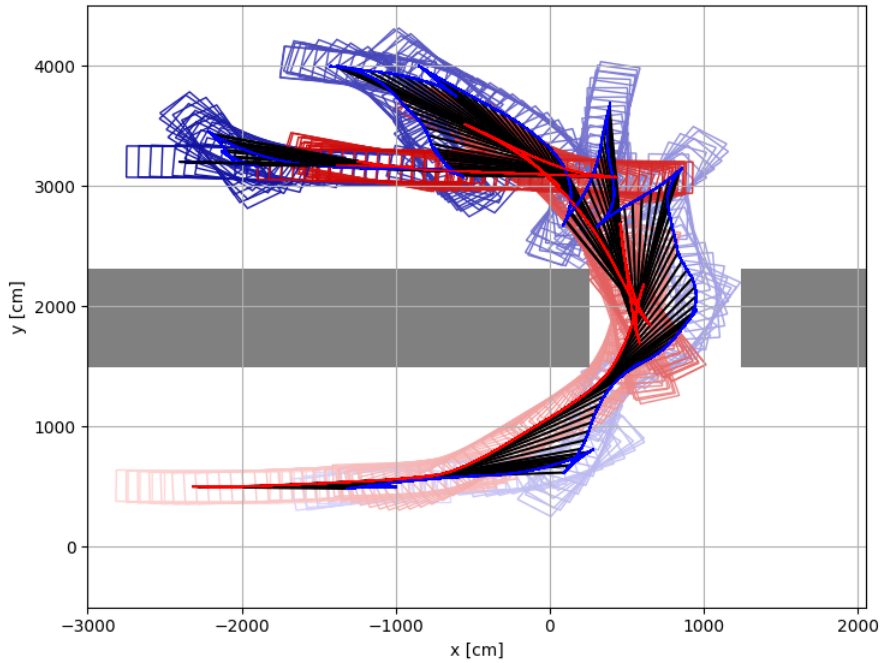
Figura 30 – Movimento do *truck* (azul) e *trailer*(vermelho) de uma das soluções geradas com melhor desempenho no planejamento para o cenário com passagem estreita



Fonte: Autoria própria.

Em relação ao experimento anterior, foram estabelecidos os mesmos valores de

Figura 31 – Movimento com a articulação (preto) de uma das soluções geradas com melhor desempenho no planejamento para o cenário com passagem estreita



Fonte: Autoria própria.

configuração de ângulo de esterçamento, velocidade, stepSize, mínima e máxima duração de aplicação dos controles, e tempo máximo de resolução para este segundo problema. Visto que o desempenho de tais fatores no resultado gerado foi praticamente o mesmo obtido nas validações realizadas anteriormente. Os resultados gerados a partir das 100 simulações deste problema novamente não registraram solução exata, com isso, foram calculados os erros euclidianos (distância) da posição final (x_0, y_0) do *truck* em relação ao destino $(3900, 800)$ cm e também a diferença entre a orientação final θ_0 truck e $\theta_1$trailer e seus respectivos destinos π rad. Tais cálculos incluindo todos os resultados gerados nas 100 simulações estão resumidos através de média, desvio padrão, mínimo e máximo na Tabela 8.

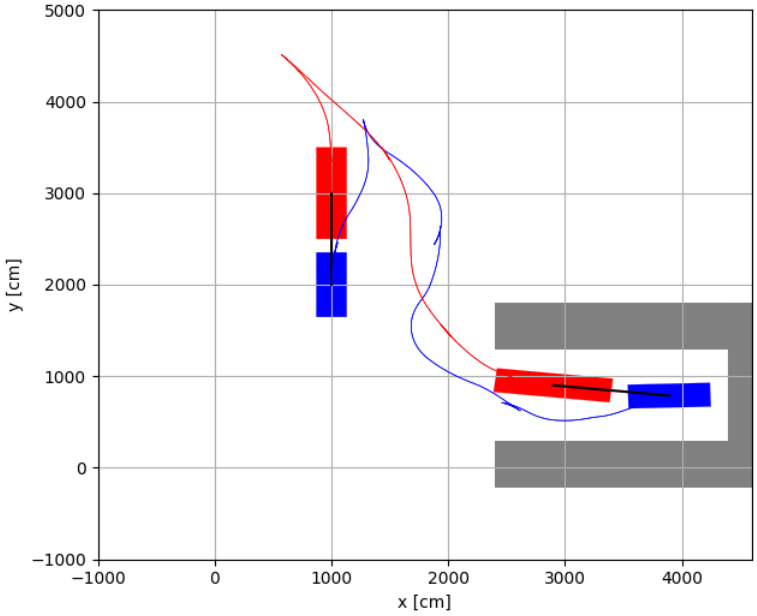
Mas, tendo em vista que o problema deste experimento possui um grau de dificuldade mais elevado, os valores de tolerância adotados foram discretamente maiores, sendo que, para a distância entre a posição final do veículo motor da solução gerada e a posição de destino desejada, considerou-se 2.5 metros e, para a diferença entre os ângulos de orientação θ_0 e θ_1, adotou-se 5°.

Com isso, 68 das 100 simulações possuíram resultados finais dentro dos limites tolerados, estatística esta que possibilita relacionar a complexidade do cenário de estacionamento diretamente com a dificuldade em atingir o estado de destino com o menor erro possível, uma vez que, no experimento para cenário com uma curta passagem estreita, obtiveram-se quase 10% a mais de testes bem sucedidos dentro de margens de tolerância

ainda menores. Assim, os valores de média, desvio padrão, mínimo e máximo das 68 simulações que se encontram dentro da tolerância constam na Tabela 9.

A Figura 32 contém um dos caminhos encontrados com os melhores indicadores registrados, isto é, possui um dos trajetos mais curtos (cerca de 81.6 metros), com uma posição final à 13.6 cm de distância do destino e com ângulos de orientação cujas diferenças em relação ao valor desejado são menores que 1.8° para θ_0 e menores que 6.5° para θ_1. É possível observar algumas posições do veículo (com contorno cinza) durante o percurso na Figura 33, e o desenvolvimento do movimento do mesmo na Figura 34, a qual exibe apenas o *truck* e o *trailer*, e Figura 35, que inclui a articulação (em preto) junto com o veículo.

Figura 32 – Uma das soluções geradas com melhor desempenho no planejamento para o cenário com estacionamento



Fonte: Autoria própria.

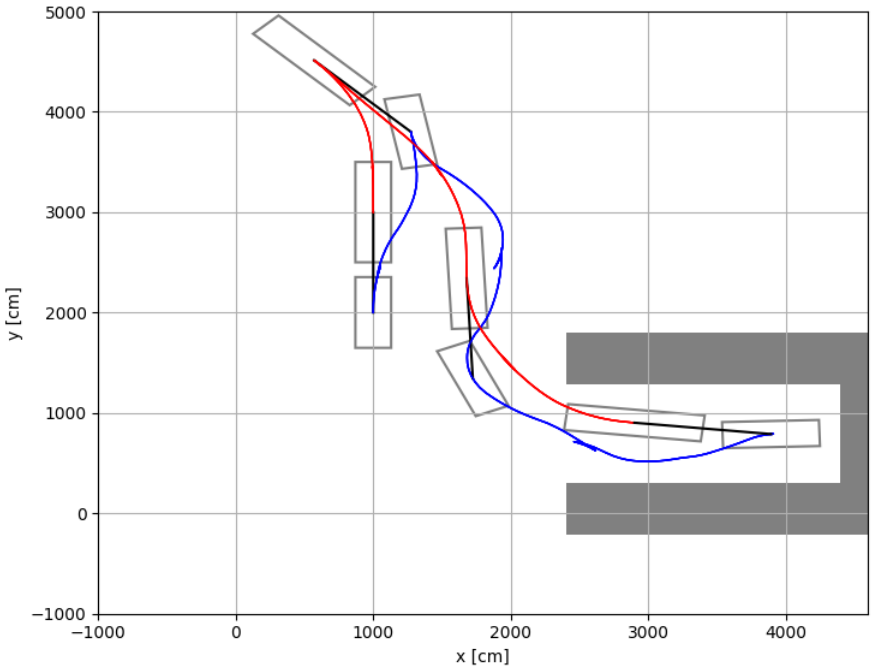
Os registros de *benchmark* coletados resultaram nos dados estatísticos de média, desvio padrão, valor máximo e valor mínimo, exibidos na Tabela 7.

Tabela 7 – Resultados estatísticos obtidos com base em tod 100 simulações do experimento para cenário com estacionamento

Parâmetro	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**
Tempo [s]	30.000545	0.000354	30.0001	30.0025
Total de estados gerados	38295.52	1910.22	34029	42043
Estados do caminho	303.09	130.08	121	771
Tamanho do caminho [m]	89.34	33.76	38.96	259.98

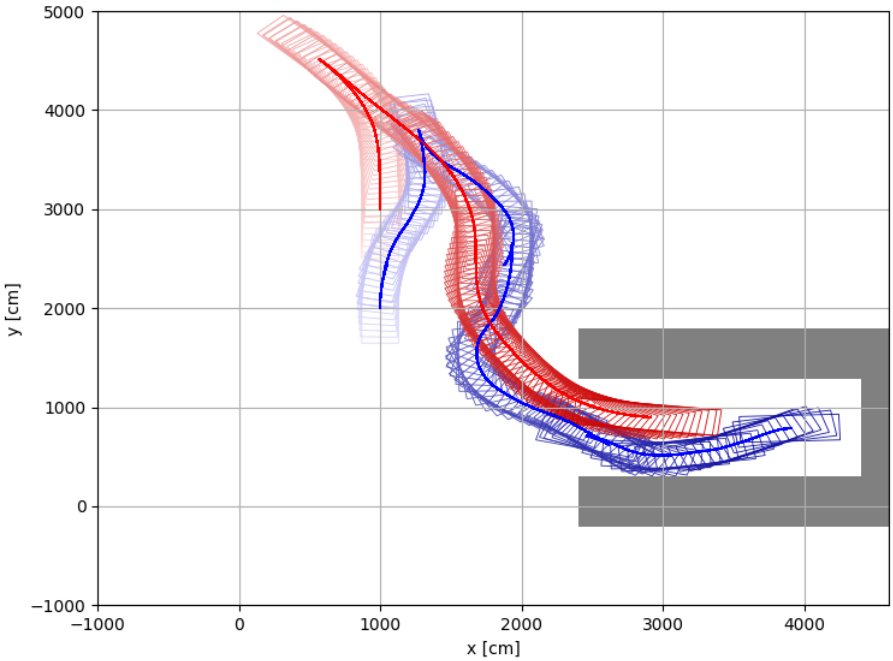
Fonte: Autoria própria.

Figura 33 – Posições pontuais do veículo (contorno cinza) em uma das soluções geradas com melhor desempenho no planejamento para o cenário com estacionamento



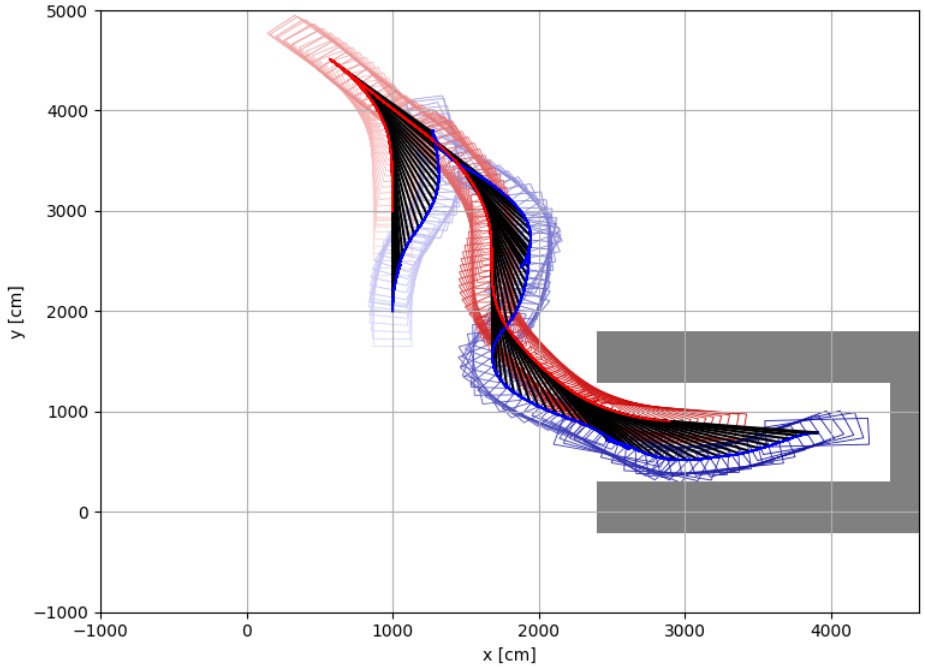
Fonte: Autoria própria.

Figura 34 – Movimento do *truck* (azul) e *trailer*(vermelho) de uma das soluções geradas com melhor desempenho no planejamento para o cenário com estacionamento



Fonte: Autoria própria.

Figura 35 – Movimento com a articulação (preto) de uma das soluções geradas com melhor desempenho no planejamento para o cenário com estacionamento



Fonte: Autoria própria.

Tabela 8 – Erros de posição (x_0, y_0) e de orientações θ_0 e θ_1 finais obtidas com base em todas as 100 simulações do experimento para cenário com estacionamento

Parâmetro	Média	Desvio Padrão	Mínimo	Máximo
Erro de orientação θ_0 [°]	20.361	17.978	0.006	58.772
Erro de orientação θ_1 [°]	12.559	14.614	0.105	61.358
Erro de posição x_0 [m]	0.80	2.48	0	12.50
Erro de posição y_0 [m]	3.14	4.60	0	13.32
Erro de posição x_0 e y_0 [m]	3.71	4.90	0	13.36

Fonte: Autoria própria.

4.3 Discussões Dos Resultados Dos Experimentos

A partir das estatísticas apresentadas nas Tabelas 4 e 7, o tempo de processamento foi igual ao máximo imposto de 30 segundos, demostrando a eficiência na análise do algoritmo de planejamento com RRT para ambos cenários com dificuldades distintas.

Apesar de mais complexo, o experimento com cenário de estacionamento possui um trajeto menor para percorrer até o destino se comparado com o primeiro experimento, com isso, pode-se afirmar que é adequado o número de estados que compõem o caminho

Tabela 9 – Erros de posição (x_0, y_0) final e de orientações θ_0 e θ_1 finais obtidas com base nas 68 simulações dentro da margem de tolerância escolhida para a distância entre a posição final do veículo motor da solução gerada e a posição de destino desejada em 2.5 metros e para a diferença entre os ângulos de orientação θ_0 e θ_1, em 5°, do experimento para cenário com estacionamento

Parâmetro	**Média**	**Desvio Padrão**	**Mínimo**	**Máximo**
Erro de orientação θ_0 [°]	26.52	16.80	1.52	58.77
Erro de orientação θ_1 [°]	5.41	3.96	0.10	15.03
Erro de posição x_0 [m]	0.24	0.29	0.00	1.14
Erro de posição y_0 [m]	0.49	0.55	0.00	2.20
Erro de posição x_0 e y_0 [m]	0.58	0.60	0.00	2.21

Fonte: Autoria própria.

apresentar medidas de média, desvio padrão, valor máximo menores. Entretanto, ao mesmo tempo que possui uma área de trabalho bem mais ampla, a quantidade total de estados gerados no segundo experimento não ultrapassou os respectivos valores do primeiro, indicando a atuação do objetivo de otimização focado no tamanho final do caminho. A adição desta funcionalidade contribuiu para que o algoritmo evitasse realizar rotas muito longas, com trechos inviáveis e pouco interessantes para este tipo de aplicação real.

Para os dois experimentos, foram obtidos caminhos com comprimentos pequenos, os quais, a princípio, poderiam ser soluções ideais para o caso de empregar a otimização do tamanho da rota, mas nem sempre implicam nesta conclusão, pois a pequena extensão da rota geralmente encontrou-se em soluções incompletas, isto é, com valores elevados de erro. Isso se deve ao fato de o desempenho da solução gerada se classificar com base no equilíbrio geral dos valores finais registrados para posição (x_0, y_0) e orientação θ_0 e θ_1. Logo, a análise acerca do comprimento da rota contribui para otimizar a solução final, entretanto, o foco principal é conseguir atingir o estado final com o menor erro possível em orientação e posição.

Além disso, os valores médios de erro da posição (x_0, y_0) considerando as 100 simulações de ambos cenários ficaram entre 3.11 e 3.71 metros, resultados estes um pouco acima do valor de tolerância, mas destaca-se a ocorrência de 52% das soluções com erro de posição (x_0, y_0) praticamente nulo (menor que 1 cm) para o problema de estacionamento, e 70% para o problema de passagem estreita. Por outro lado, ao observar as Tabelas 6 e 9, nota-se que os resultados dentro da margem de tolerância apresentaram ótimo desempenho em relação aos erros de posição do *truck*, pois estes tiveram médias e desvios padrões menores que 65 cm, valores máximos até 2.46 metros e mínimos iguais a 0.

O erro da orientação θ_0, considerando todas as 100 simulações, demonstrou valores médios elevados (entre 20.36° e 40.59° considerando ambos cenários), tendo 20% de resultados abaixo de 1° para o problema de estacionamento e apenas 4%, para o de passagem estreita. Por outro lado, o erro de θ_1 registrou valor médio entre 12.56° e 22.26°, mas com uma taxa de 7% a 10% de soluções com erro menor que 1°. Baseando nas Tabelas 6 e 9, pode-se perceber que, mesmo para os resultados dentro da tolerância em ambos cenários, o erro médio da orientação θ_0 ficou acima de 21°, que é um valor consideravelmente elevado para este tipo de aplicação. Entretanto, o erro médio de orientação θ_1 ficou abaixo de 8°, o qual corresponde a um bom resultado final.

Dessa forma, no geral, os testes realizados validaram a capacidade do algoritmo RRT para determinar soluções viáveis para problemas contendo ambientes bem limitados espacialmente. As variações de valores obtidos em dois cenários completamente distintos ressaltam o nível de complexidade elevado para conseguir fazer um planejamento equilibrado, isto é, com pequenos erros finais e com eficiência em relação ao tempo de processamento e ao comprimento do trajeto final a ser percorrido.

As imagens geradas dos movimentos finais de cada experimento (Figuras 30 e 34) permitem validar também o processo de detecção de colisão bem sucedido, visto que o veículo se locomoveu apenas nas regiões sem colisão com os obstáculos.

5 CONCLUSÃO

Esta monografia abordou o processo de implementação de um algoritmo de planejamento de caminho para um caminhão articulado com uma carreta acoplada (*truck-trailer*), representado pelo espaço de estados Reeds Shepp, empregando na estratégia de planejamento o algoritmo RRT, reconhecido na literatura para este tipo de aplicação.

O veículo articulado simulado neste projeto baseou-se em um modelo cinemático simples (Seção 6), e foi dimensionado adequadamente para representar um caminhão real (Tabela 1) em uma aplicação prática do mesmo em cenários comuns a este tipo de veículo. O fato deste objeto de estudo possuir mais graus de liberdade do que um veículo comum implica em um planejamento mais complexo, o qual foi contornado ao empregar o algoritmo baseado em amostragem RRT juntamente com o espaço de estados Reeds Shepp que inclui movimentos para frente e para trás, contribuindo consideravelmente na busca pela solução devido a esta flexibilidade. Vale lembrar que o espaço Dubins é também empregado para determinar trajetos ótimos, mas que, pelas simulações realizadas na Seção 3.3, apresentou impasses como maior tempo de processamento, maiores erros e menores chances de encontrar uma solução.

A inserção de uma biblioteca designada para detectar colisão entre objetos geométricos (FCL) possibilitou resolver a etapa do planejamento de considerar também as dimensões do veículo e dos obstáculos, e não mais considerar o veículo como um corpo rígido pontual (como visto na Seção 3.2). Assim, por visualizações do movimento do veículo como um todo, foi possível validar este importante efeito aplicado ao veículo durante seu trajeto.

As equações cinemáticas que representam o sistema do veículo articulado foram inseridas e consideradas através do uso do espaço de controle da biblioteca OMPL, que justamente possui uma função responsável por propagar as variáveis de controle (velocidade e ângulo de esterçamento) do sistema na posição e orientação do veículo por um intervalo de tempo *duration*. E instanciando um objetivo de otimização baseado no comprimento do caminho proporcionou rotas econômicas, direcionadas à região de destino, evitando percorrer trechos desnecessários.

Os cenários criados para simular situações reais com um caminhão articulado em tamanho real permitiram observar o desempenho do algoritmo a partir dos valores de tempo de processamento, erro e quantidade de estados produzidos em 100 iterações para cada cenário. Demonstrando, assim, a eficiência do método RRT para solucionar distintos problemas de planejamento de caminho.

Para trabalhos futuros, são sugeridos:

- Estender o estudo deste planejamento de caminho em sistemas dedicados à construção de veículos/robôs, como ROS (do inglês *Robot Operating System*), para validar o algoritmo em testes experimentais;
- Efetuar testes de planejamento para outros tipos de algoritmos baseados em amostragem e abordar mais métodos de otimização do caminho encontrado;
- Estudar a métrica ρ apresentada na definição do método RRT para o algoritmo implementado, uma vez que o RRT é aplicado sobre o espaço de controle;
- Estender o estudo para a aplicação do método RRT bidirecional, tratando-se de um método mais rápido e otimizado;
- Aprofundar o estudo acerca da ponderação dos objetivos finais (posição e orientação) na performance do algoritmo para determinar a solução. E adaptar o algoritmo para que a carreta, além da sua orientação, tenha a sua posição (x,y) inserida na estratégia de planejamento, visando atingir com melhor precisão a configuração final do veículo articulado.

REFERÊNCIAS

ALVES, L. USP e Scania desenvolvem caminhão autônomo totalmente produzido no Brasil. **Universidade de São Paulo**, 2015. Disponível em: <https://www5.usp.br/95355/usp-e-scania-desenvolvem-caminhao-autonomo-totalmente-produzido-no-brasil/>. Acesso em: 01 mar. 2020.

ANDREN, N. et al. **Predictive Control for Autonomous Articulated Vehicles**. 2017. 67 f. Bachelor Thesis — University of Gothenburg, Göteborg, Sweden, 2017. Disponível em: <https://gupea.ub.gu.se/bitstream/2077/53343/1/gupea_2077_53343_1.pdf>.

BARLAND, M. **Motion Planning Framework for Industrial Manipulators using the Open Motion Planning Library (OMPL)**. 2012. 62 f. Dissertação (Master of Science in Engineering Cybernetics) — Norwegian University of Science and Technology, Trondheim, 2012. Disponível em: <https://core.ac.uk/download/pdf/52106758.pdf>.

BENEVIDES, J. R. S. **Planejamento de Rota para Manipulador Planar de Base Livre Flutuante utilizando o Algoritmo RRT**. 2015. Dissertação (Dissertação (Mestrado em Sistemas Dinâmicos)) — Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2015. Doi:10.11606/D.18.2015.tde-20052015-085004.

BOUTELDJA, M.; CEREZO, V. Jackknifing warning for articulated vehicles based on a detection and prediction system. **3rd International Conference on Road Safety and Simulation**, Indianapolis, p. 14–16, set. 2011. Disponível em: <http://onlinepubs.trb.org/onlinepubs/conferences/2011/RSS/3/Bouteldja,M.pdf>. Acesso em: 17 maio 2020.

BOZCUOGLU, A. K. **Rapidly-Exploring Random Trees**. [s.n.], 2020. Disponível em: <http://kovan.ceng.metu.edu.tr/~asil/old/_1./hw4.html>. Acesso em: 21 abr. 2020.

SUCAN, I. A.; KAVRAKI, L. E. A sampling-based tree planner for systems with complex dynamics. **IEEE Transactions on Robotics**, v. 28, n. 1, p. 116–131, February 2012.

ELHASSAN, A. **Autonomous driving system for reversing an articulated vehicle**. 2015. 82 f. Dissertação (Master of Science Thesis) — KTH ROYAL INSTITUTE OF TECHNOLOGY, Stockholm, 2015. Disponível em: <https://www.kth.se/polopoly_fs/1.602293.1550156311!/AmroThesis.pdf>.

FAGGELLA, D. **Self-driving car timeline for 11 top automakers**. **Venture Beat**, 2017. Disponível em: <https://venturebeat.com/2017/06/04/self-driving-car-timeline-for-11-top-automakers/>. Acesso em: 01 mar. 2020.

GAMMA. **Collision Detection and Proximity Query Packages**. <http://gamma.cs.unc.edu/research/collision/packages.html>. Acesso em: 15 jun. 2019.

_____. **PQP - A Proximity Query Package**. 1999. <http://gamma.cs.unc.edu/SSV>. Acesso em: 15 jun. 2019.

GITHUB. **Flexible Collision Library**. 2020. <https://github.com/flexible-collision-library/fcl>. Acesso em: 20 jun. 2020.

KAVRAKI LAB. **GeometricCarPlanning.cpp**. 2010. Disponível em: <https://ompl.kavrakilab.org/GeometricCarPlanning_8cpp_source.html>. Acesso em: 12 abr. 2020.

_____. **RigidBodyPlanning.cpp**. 2010. Disponível em: <https://ompl.kavrakilab.org/RigidBodyPlanning_8cpp_source.html>. Acesso em: 12 abr. 2020.

_____. **The Open Motion Planning Library**. 2020. Disponível em: <https://ompl.kavrakilab.org/>. Acesso em: 10 abr. 2020.

KUFFNER, J. J.; LAVALLE, S. M. Rrt-connect: An efficient approach to single-query path planning. In: **Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)**. [S.l.: s.n.], 2000. v. 2, p. 995–1001 vol.2.

LARSEN, E. et al. **Fast Proximity Queries with Swept Sphere Volumes**. Chapel Hill, f 32, 2000. Disponível em: <http://gamma.cs.unc.edu/SSV/ssv.pdf>.

LAVALLE, S. M. **Rapidly-Exploring Random Trees: A New Tool for Path Planning**. [S.l.], 1998.

_____. **Planning Algorithms**. [S.l.]: Cambridge University Press, 2006. 842 f.

LECTURA SPECS. **Scania G 360 B6x4HZ Specifications Technical Data (2016-2019).** 2020. Disponível em: <https://www.lectura-specs.com/en/model/transportation/trucks-rigid-chassis-scania/g-360-b6x4hz-11725842>. Acesso em: 20 abr. 2020.

LJUNGQVIST, O. On motion planning and control for truck and trailer systems. **Linköping University Electronic Press**, Linköping, p. 78, 2019. Doi: 10.3384/lic-diva-153892.

LUIJTEN, M. F. J. **Lateral Dynamic Behaviour of Articulated Commercial Vehicles**. 2010. Dissertação (Master's thesis) — Eindhoven University of Technology, Eindhoven, 2010. Acesso em: 14 mar. 2020.

PISSARDINI, R.; WEI, D.; JR., E. F. VeÍculos autÔnomos: Conceitos, histÓrico e estado-da-arte. In: . São Paulo: [s.n.], 2013.

PRADO, M. G. **Planejamento de trajetória para estacionamento de veículos autônomos**. 2013. Dissertação (Dissertação (Mestrado em Ciências de Computação e Matemática Computacional)) — Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2013. Doi:10.11606/D.55.2013.tde-13052013-101339. Acesso em: 23 mar. 2020.

RODRIGUES, L. C. **FUNDAMENTOS, TECNOLOGIAS E APLICAÇÕES DE VEÍCULOS AUTÔNOMOS**. 2017. 83 f. Trabalho de Conclusão do Curso (Bacharelado Engenharia Eletrônica) — Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2017.

SERRANTOLA, W. G. **Planejamento de rota e trajetória para manipulador planar de base livre flutuante com dois braços**. 2018. Dissertação (Dissertação (Mestrado em Sistemas Dinâmicos)) — Escola de Engenharia de São Carlos, Universidade

de São Paulo, São Carlos, 2018. Doi:10.11606/D.18.2018.tde-06112018-103624. Acesso em: 22 mar. 2020.

ŞUCAN, I. A.; KAVRAKI, L. E. Kinodynamic motion planning by interior-exterior cell exploration. In: **Springer Tracts in Advanced Robotics**. Springer Berlin Heidelberg, 2009. p. 449–464. Disponível em: <https://doi.org/10.1007%2F978-3-642-00312-7_28>.

UNIVERSIDADE DE SÃO PAULO. **Arquivo dos álbuns**. São Carlos, 2015. Disponível em: <https://get.google.com/albumarchive/113784208838909725493>. Acesso em: 16 fev. 2020.

WIRED. **A brief story of Autonomous Vehicles Technology**. 2016. Disponível em: <https://www.wired.com/brandlab/2016/03/a-brief-history-of-autonomous-vehicle-technology/>. Acesso em: 01 mar. 2020.

WORLD HEALTH ORGANIZATION. **Road traffic injuries**. 2020. <https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries>. Acesso em: 10 jun. 2020.

ZIPS, P.; BÖCK, M.; KUGI, A. An optimisation-based path planner for truck-trailer systems with driving direction changes. In: **Proc. IEEE Int. Conf. on Robotics and Automation**. Seattle, WA, USA: [s.n.], 2015. p. 630–636.

A APÊNDICE(S)

Listing A.1 – Algoritmo de planejamento empregado no experimento de estacionamento

```
1 #include <ompl/base/SpaceInformation.h>
2 #include <ompl/geometric/SimpleSetup.h>
3 #include <ompl/config.h>
4 #include <ompl/geometric/PathGeometric.h>
5 #include <ompl/base/spaces/DubinsStateSpace.h>
6 #include <ompl/base/spaces/ReedsSheppStateSpace.h>
7 #include <ompl/base/ScopedState.h>
8 #include <ompl/base/spaces/SO2StateSpace.h>
9 #include <ompl/base/spaces/SE2StateSpace.h>
10 #include <ompl/control/spaces/RealVectorControlSpace.h>
11 #include <ompl/control/SpaceInformation.h>
12 #include <ompl/control/planners/rrt/RRT.h>
13 #include <ompl/control/SimpleSetup.h>
14 #include <ompl/base/objectives/PathLengthOptimizationObjective.h
      >
15 #include <ompl/control/PathControl.h>
16 #include <iostream>
17 #include <valarray>
18 #include <limits>
19 #include <boost/math/constants/constants.hpp>
20 #include <fcl/fcl.h>
21 #include <math.h>
22 #include <fcl/broadphase/broadphase_collision_manager.h>
23 #include <fcl/broadphase/broadphase_collision_manager-inl.h>
24 #include <fcl/broadphase/broadphase_dynamic_AABB_tree.h>
25 #include <fcl/broadphase/broadphase_dynamic_AABB_tree-inl.h>
26 #include <fcl/broadphase/broadphase_dynamic_AABB_tree_array.h>
27 #include <fcl/broadphase/default_broadphase_callbacks.h>
28 #include <fstream>
29 #include <boost/program_options.hpp>
30 #include <ompl/base/StateSpace.h>
31 #include <ompl/base/OptimizationObjective.h>
32
33 namespace ob = ompl::base;
```

```
34 namespace og = ompl::geometric;
35 namespace oc = ompl::control;
36 namespace po = boost::program_options;
37 using namespace fcl;
38
39 void propagate(const oc::SpaceInformation *si, const ob::State *
     start,
40 const oc::Control *control, const double duration, ob::State *
     result)
41 {
42       // CONTROL
43     const double* ctrl =
44     control->as<oc::RealVectorControlSpace::ControlType>()->
        values;
45
46     // START STATE COMPOUND
47     const auto *se2_start =
48     start->as<ob::CompoundState>()->as<ob::SE2StateSpace::
        StateType>(0);
49     const auto *se2_pos_start =
50     se2_start->as<ob::RealVectorStateSpace::StateType>(0);
51     const auto *se2_rot_start =
52     se2_start->as<ob::SO2StateSpace::StateType>(1);
53     const auto *r1_start =
54     start->as<ob::CompoundState>()->as<ob::RealVectorStateSpace
        ::StateType>(1);
55
56     // RESULT STATE COMPOUND
57     auto *se2_result = result->as<ob::CompoundState>()->as<ob::
        SE2StateSpace::StateType>(0);
58     auto *se2_result_pos = se2_result->as<ob::
        RealVectorStateSpace::StateType>(0);
59     auto *se2_result_rot = se2_result->as<ob::SO2StateSpace::
        StateType>(1);
60     auto *r1_result = result->as<ob::CompoundState>()->as<ob::
        RealVectorStateSpace::StateType>(1);
61
62     const double L = 705, d1 = 1000;
63     // PROPAGATION FOR TRUCK
```

```
64      se2_result_pos->values[0] = se2_pos_start->values[0] + ctrl
            [0]*duration*cos(se2_rot_start->value);
65      se2_result_pos->values[1] = se2_pos_start->values[1] + ctrl
            [0]*duration*sin(se2_rot_start->value);
66      se2_result_rot->value = se2_rot_start->value + (ctrl[0]*
            duration*tan(ctrl[1]))/L;
67      // PROPAGATION FOR TRAILER
68      r1_result->values[0] = r1_start->values[0] + (ctrl[0]*
            duration*sin(se2_rot_start->value - r1_start->values[0]))
            /d1;
69 }
70
71 void rotationmatrixtheta(double theta, Matrix3d& R)
72 {
73     R << cos(theta), -sin(theta), 0,
74          sin(theta), cos(theta), 0,
75          0, 0, 1;
76 }
77
78 bool isStateValidFCL(const oc::SpaceInformation *si, const ob::
      State *state)
79 {
80      const auto *truck_space = state->as<ob::CompoundState>()->as
            <ob::SE2StateSpace::StateType>(0);
81      const auto *trailer_space = state->as<ob::CompoundState>()->
            as<ob::RealVectorStateSpace::StateType>(1);
82      //EXTRACTING TRUCK POSITION XY AND ORIENTATION YAW:
83      const auto *truck_space_pos = truck_space->as<ob::
            RealVectorStateSpace::StateType>(0);
84      const auto *truck_space_rot = truck_space->as<ob::
            SO2StateSpace::StateType>(1);
85      //EXTRACTING TRAILER ORIENTATION YAW
86      const auto *trailer_space_rot = trailer_space->as<ob::
            RealVectorStateSpace::StateType>();
87
88      double x1 = truck_space_pos->values[0];  // x position of
            the truck
89      double y1 = truck_space_pos->values[1];  // y position of
            the trailer
```

```
90      double z = 0; // Considering that the truck trailer only
            walks in the plan XY
91      double theta_truck = truck_space_rot->value;     // theta
            orientation of the truck
92      double theta_trailer = trailer_space_rot->values[0]; //
            theta orientation of the trailer
93      double d1 = 1000; // distance between the connection points
            truck-trailer
94
95      Matrix3d R_truck, R_trailer; // MATRIX ROTATION FOR THE
            TRUCK AND TRAILER
96      Transform3d tf_obstacle1, tf_obstacle2, tf_obstacle3,
            tf_truck, tf_trailer;
97      rotationmatrixtheta(theta_trailer, R_trailer); // CONVERTING
             THE YAW FROM TRUCK AND TRAILER TO THE MATRIX ROTATION 3D
98      rotationmatrixtheta(theta_truck, R_truck); // THE MATRIX
            USED IS FOR ROTATION AROUND THE Z AXIS
99
100     std::shared_ptr<Boxd> obstacle_geom1(new Boxd(2400, 500,
            500)); // obstacle defined as a cube
101     tf_obstacle1.setIdentity();
102     //tf_obstacle.linear() = R_obstacle;
103     tf_obstacle1.translation() = Vector3d(3600, 50, 0.);     //
            obstacle position
104     CollisionObjectd co_obstacle1(obstacle_geom1, tf_obstacle1);
105
106     std::shared_ptr<Boxd> obstacle_geom2(new Boxd(2400, 500,
            500)); // obstacle defined as a cube
107     tf_obstacle2.setIdentity();
108     //tf_obstacle.linear() = R_obstacle;
109     tf_obstacle2.translation() = Vector3d(3600,1550,0.);     //
            obstacle position
110     CollisionObjectd co_obstacle2(obstacle_geom2, tf_obstacle2);
111
112     std::shared_ptr<Boxd> obstacle_geom3(new Boxd(200, 1800,
            500)); // obstacle defined as a cube
113     tf_obstacle3.setIdentity();
114     //tf_obstacle.linear() = R_obstacle;
115     tf_obstacle3.translation() = Vector3d(4500, 900, 0.);     //
```

```
        obstacle position
116     CollisionObjectd co_obstacle3(obstacle_geom3, tf_obstacle3);
117
118     std::shared_ptr<Boxd> truck_geom(new Boxd(705, 260, 327));
        // car defined as a cube
119     tf_truck.setIdentity();
120     tf_truck.linear() = R_truck;
121     tf_truck.translation() = Vector3d(x1,y1,z);     // truck
        position
122     CollisionObjectd co_truck(truck_geom, tf_truck);
123
124     std::shared_ptr<Boxd> trailer_geom(new Boxd(1000, 260, 327))
        ; // trailer defined as a cube
125     tf_trailer.setIdentity();
126     tf_trailer.linear() = R_trailer;
127     tf_trailer.translation() = Vector3d((x1 - d1*cos(
        theta_trailer)),(y1 - d1*sin(theta_trailer)),z);     //
        trailer position
128     CollisionObjectd co_trailer(trailer_geom, tf_trailer);
129
130     std::vector<CollisionObjectd*> group_obstacles = {&
        co_obstacle1, &co_obstacle2, &co_obstacle3};
131     std::vector<CollisionObjectd*> group_car = {&co_truck, &
        co_trailer};
132     BroadPhaseCollisionManager<double>* manager_obstacles_group
        = new DynamicAABBTreeCollisionManager<double>();
133     BroadPhaseCollisionManager<double>* manager_car_group = new
        DynamicAABBTreeCollisionManager<double>();
134     manager_obstacles_group->registerObjects(group_obstacles);
135     manager_car_group->registerObjects(group_car);
136     DefaultCollisionData<double> collision_data_carObst;
137     DefaultCollisionData<double> collision_data_carself;
138     manager_obstacles_group->setup();
139     manager_car_group->setup();
140
141     // COLLISION QUERY BETWEEN THE 2 GROUPS
142     manager_obstacles_group->collide(manager_car_group,&
        collision_data_carObst, DefaultCollisionFunction);
143     bool res_collision_carObst = collision_data_carObst.result.
```

```
        isCollision();
144     // SELF COLLISION QUERY IN THE CAR GROUP
145     manager_car_group->collide(&collision_data_carself,
        DefaultCollisionFunction);
146     bool res_collision_self = collision_data_carself.result.
        isCollision();
147
148     double theta_dif = std::abs(theta_truck - theta_trailer);
149
150     return si->satisfiesBounds(state) && (!res_collision_self)
        && (!res_collision_carObst) && (theta_dif<=(boost::math::
        constants::pi<double>()/3));
151 }
152
153 void planWithSimpleSetup()
154 {
155     auto espacose2(std::make_shared<ob::ReedsSheppStateSpace>())
        ; // TRUCK STATE SPACE (SE2)
156     auto espacor1(std::make_shared<ob::RealVectorStateSpace>(1))
        ; // TRAILER STATE SPACE
157     ob::StateSpacePtr spacefull = espacose2 + espacor1;
158
159     ob::RealVectorBounds espacose2bounds(2);
160     espacose2bounds.setLow(0, -10000); // min do x do truck
161     espacose2bounds.setHigh(0, 4400); // max do x do truck
162     espacose2bounds.setLow(1, -10000); // min do y do truck
163     espacose2bounds.setHigh(1, 10000); // max do y do truck
164
165     ob::RealVectorBounds espacor1bounds(1);
166     espacor1bounds.setLow(-boost::math::constants::pi<double>())
        ; // min theta do trailer
167     espacor1bounds.setHigh(boost::math::constants::pi<double>())
        ; // max theta do trailer
168
169     espacose2->setBounds(espacose2bounds);
170     espacor1->setBounds(espacor1bounds);
171
172     auto cspace(std::make_shared<oc::RealVectorControlSpace>(
        spacefull, 2)); // control space of 2 controls (truck
```

```
        velocity and steering angle)
173     ob::RealVectorBounds cbounds(2);
174     cbounds.setLow(0, -700.); // min velocity
175     cbounds.setHigh(0, 700.); // max velocity
176     cbounds.setLow(1, -boost::math::constants::pi<double>()/3);
        // min steering angle
177     cbounds.setHigh(1, boost::math::constants::pi<double>()/3);
        // max steering angle
178     cspace->as<oc::RealVectorControlSpace>()->setBounds(cbounds)
        ;
179
180     oc::SimpleSetup ss(cspace);
181     const oc::SpaceInformation *si = ss.getSpaceInformation().
        get();
182
183     ss.setStatePropagator([si](const ob::State *state, const oc
        ::Control* control, const double duration, ob::State *
        result)
184     { propagate(si, state, control, duration, result); });
185
186     ss.setStateValidityChecker([si](const ob::State *state) {
        return isStateValidFCL(si, state); });
187
188     ob::ScopedState<> start(spacefull);
189     start[0] = 1000; start[1] = 2000; // x0 e y0
190     start[2] = -boost::math::constants::pi<double>()/2; // theta
         do truck
191     start[3] = -boost::math::constants::pi<double>()/2; // theta
         do trailer
192
193     ob::ScopedState<> goal(spacefull);
194     goal->as<ob::CompoundState>()->as<ob::SE2StateSpace::
        StateType>(0)->setX(3900.);
195     goal->as<ob::CompoundState>()->as<ob::SE2StateSpace::
        StateType>(0)->setY(800.);
196     goal->as<ob::CompoundState>()->as<ob::SE2StateSpace::
        StateType>(0)->setYaw(0*boost::math::constants::pi<double
        >());
197     goal->as<ob::CompoundState>()->as<ob::RealVectorStateSpace::
```

```
        StateType>(1)->values[0] = 0*boost::math::constants::pi<
        double>();
198
199     ss.getSpaceInformation()->setPropagationStepSize(0.1);
200     ss.getSpaceInformation()->setMinMaxControlDuration(1, 10);
201     ss.setStartAndGoalStates(start, goal);
202
203     ss.getProblemDefinition()->setOptimizationObjective(std::
        make_shared<ob::PathLengthOptimizationObjective>(ss.
        getSpaceInformation()));
204
205     ss.setPlanner(std::make_shared<oc::RRT>(ss.
        getSpaceInformation()));
206     ss.setup();
207     ss.print();
208     ob::PlannerStatus solved = ss.solve(30.0);
209     if (solved)
210     {
211         std::cout << "--- Found solution: ----" << std::endl;
212         og::PathGeometric path1 = ss.getSolutionPath().
            asGeometric();
213         std::ofstream myfile;
214         myfile.open("saida_head.txt");
215         myfile << ss.getLastPlannerStatus() << std::endl;
216         myfile << ss.haveExactSolutionPath() << std::endl;
217         myfile << ss.haveSolutionPath() << std::endl;
218         myfile << ss.getLastPlanComputationTime() << std::endl;
219         myfile << path1.getStateCount() << std::endl;
220         myfile << path1.length() << std::endl;
221         myfile << path1.clearance() << std::endl;
222         myfile.close();
223
224         myfile.open("saida.txt");
225         path1.interpolate(7000);
226         path1.printAsMatrix(myfile);
227         myfile.close();
228     }
229     else
230         std::cout << "---No solution found---" << std::endl;
```

```
231 }
232
233 int main(int /*argc*/, char ** /*argv*/)
234 {
235       std::cout << "OMPL version: " << OMPL_VERSION << std::endl;
236       std::cout << std::endl << std::endl;
237       planWithSimpleSetup();
238       return 0;
239 }
```