UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

RÔMULO RAMOS RADAELLI

PLANEJAMENTO DE MOVIMENTO PARA VEÍCULOS CONVENCIONAIS USANDO RAPIDLY-EXPLORING RANDOM TREE

VITÓRIA

2013

RÔMULO RAMOS RADAELLI

PLANEJAMENTO DE MOVIMENTO PARA VEÍCULOS CONVENCIONAIS USANDO RAPIDLY-EXPLORING RANDOM TREE

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

VITÓRIA

2013

Dados Internacionais de Catalogação-na-publicação (CIP)
(Biblioteca Setorial Tecnológica,
Universidade Federal do Espírito Santo, ES, Brasil)

R124p

Radaelli, Rômulo Ramos, 1990-
Planejamento de movimento para veículos convencionais usando Rapidly-Exploring Random Tree / Rômulo Ramos Radaelli. – 2013.
82 f. : il.

Orientador: Claudine Badue.
Coorientador: Alberto Ferreira de Souza.
Dissertação (Mestrado em Informática) – Universidade Federal do Espírito Santo, Centro Tecnológico.

1. Robótica. 2. Veículos autônomos. 3. Navegação de robôs móveis. I. Badue, Claudine. II. Souza, Alberto Ferreira de. III. Universidade Federal do Espírito Santo. Centro Tecnológico. IV. Título.

CDU: 004

RÔMULO RAMOS RADAELLI

PLANEJAMENTO DE MOVIMENTO PARA VEÍCULOS CONVENCIONAIS USANDO RAPIDLY-EXPLORING RANDOM TREE

COMISSÃO EXAMINADORA

Profa. Dra. Claudine Badue

Universidade Federal do Espírito Santo

Orientadora

Prof. Dr. Alberto Ferreira de Souza

Universidade Federal do Espírito Santo

Coorientador

Prof. Dr. Edilson de Aguiar

Universidade Federal do Espírito Santo

Prof. Dr. Denis Fernando Wolf

Universidade de São Paulo

Vitória, 26 de Agosto de 2013.

RESUMO

Este trabalho tem o objetivo de apresentar um planejador de movimento para veículos convencionais baseado no algoritmo Rapidly-Exploring Random Tree (RRT). O RRT foi escolhido por ser capaz de lidar com o modelo de movimentação de um veículo convencional, que está sujeito a restrições cinemáticas e dinâmicas, as quais tornam o problema de planejamento de movimento mais difícil. As restrições cinemáticas de um veículo convencional impedem que ele gire ao redor do próprio eixo ou se movimente lateralmente, e as restrições dinâmicas limitam sua velocidade e aceleração.

O planejador de movimento desenvolvido neste trabalho é capaz de realizar planejamentos para pista e em áreas livres. No planejamento para pista, o objetivo é manter o carro em sua faixa, sem invadir a contra mão. No planejamento em áreas livres, o objetivo é gerar trajetórias capazes de estacionar o veículo.

O desempenho do planejador de movimento foi avaliado em um arcabouço de simulação de veículos autônomos em uma plataforma robótica chamada IARA, baseada no automóvel de passeio Ford Escape Hybrid adaptado com sensores e mecanismos para controlar o acelerador, freio, posição do volante, etc., por meio de computadores instalados no carro. Os resultados experimentais demonstram que o planejador de movimento é capaz de produzir trajetórias livres de colisão em tempo real. Além disso, as trajetórias geradas pelo planejador de movimento podem gastar uma quantidade de energia equivalente àquelas geradas por um motorista.

ABSTRACT

This work presents a motion planning for car-like robots based on the Rapidly-Exploring Random Tree (RRT) algorithm. RRT was chosen because it is capable of handling with the motion model of a car-like vehicle that is subject to kinematic and dynamic constraints, which make the motion planning problem harder. Kinematic constraints of a car-like robot prevent it from turning around its own axis or moving laterally, and dynamic constraints limit its speed and acceleration.

The motion planner developed in this work is able to plan on the road and in open areas. When planning on the road, the goal is to keep the car in its lane, without invading the counter hand. When planning in open areas, the goal is to generate trajectories able to park the car.

The performance of the motion planner was evaluated in a simulation framework for autonomous vehicles and in a robotic platform called IARA. IARA is built upon the Ford Escape Hybrid equipped with sensors and modified with mechanisms for controlling the throttle, brake, steering wheel position, etc., through computers installed in the car. Experimental results demonstrate that the motion planner is capable of planning collision-free trajectories in real time. Moreover, the trajectories generated by the motion planner can spend an amount of energy equivalent to those generated by a driver.

LISTA DE FIGURAS

Sumário

1 INTRODUÇÃO

A robótica está passando por uma grande transformação de escopo e dimensão. Saindo de um contexto dominantemente industrial, o foco da robótica está rapidamente se expandindo para os desafios de ambientes não estruturados. Interagindo, auxiliando e explorando com humanos, os robôs emergentes estarão cada vez mais em contato com as pessoas e suas vidas [1].

Um exemplo dessa interação com humanos são os veículos autônomos. No futuro, eles poderão prover acessibilidade para deficientes, reduzir o tempo e o custo para transportar cargas e oferecer conforto para pessoas que não podem ou simplesmente não querem dirigir. Espera-se também que com o uso de veículos autônomos, o trânsito se torne mais seguro, pois um robô não sofre de problemas como cansaço, nervosismo, embriaguez, pressa, estresse, etc. Segundo [2], sensoriamento e controle automatizado constituem em importantes tecnologias dos futuros veículos que podem salvar milhares de vidas que são perdidas em acidentes de trânsito.

Além disso, esta tecnologia tem aplicação militar. Robôs autônomos podem ser utilizados em missões perigosas evitando perdas de vidas. Pensando nisso, a Defense Advanced Research Projects Agency (DARPA) organizou uma série de competições para acelerar o desenvolvimento de tecnologia para veículos autônomos terrestres [1]. Três competições foram realizadas:

1. DARPA Grand Challenge 2004: nesta competição, os carros tinham que percorrer 240 km autonomamente. O evento teve 106 inscritos, mas apenas 15 times competiram no evento final. O melhor veículo conseguiu completar cinco por cento do percurso antes de falhar [1].
2. DARPA Grand Challenge 2005: nesta competição, os carros tinham que percorrer 212 km de estrada de chão autonomamente. O evento teve 197 inscrições e cada um dos 23 times que chegaram à final passaram por uma série de desafios complexos, que comprovaram que seus carros eram melhores que os carros finalistas da corrida de 2004. Cinco times conseguiram completar o percurso [1].

3. DARPA Urban Challenge 2007: esta competição envolvia um percurso de 96 km em área urbana, que tinha que ser completado em menos de 6 horas. As regras incluíam obedecer todos os sinais de trânsito e negociar com outros carros para entrar em rotatórias [2]. Seis equipes completaram o percurso.

Depois das competições organizadas pela DARPA, progressos na tecnologia de veículos autônomos continuaram acontecendo por outras iniciativas, como:

- Google *driveless car*: veículo autônomo baseado no Toyota Prius desenvolvido pela empresa Google. Foi o primeiro carro autônomo licenciado em Nevada [3]. Google anunciou que o veículo completou 500000 km em percursos autonomamente sem acidentes [4].
- *MadeInGermany*: veículo autônomo baseado no Volkswagen Passat desenvolvido pelo laboratório AutoNOMOS [5]. Foi certificado para dirigir autonomamente no estado de Berlim [6].
- Recentemente, montadoras de veículos também mostraram resultados em suas pesquisas de veículos autônomos. A Mercedes-Benz apresentou uma versão autônoma do sedã S 500 e demonstrou a tecnologia em um percurso urbano [7]. Volvo apresentou um carro que pode estacionar e dirigir autonomamente. O veículo está equipado com um sistema para detectar animais, pedestres, faixas de rodovias e barreiras, assim como um sistema para comunicação entre carros [8]. A Nissan também está pesquisando a tecnologia de veículos autônomos e pretende introduzi-la em 2020 [9].

Para que um carro seja autônomo é preciso que ele execute diversas tarefas, tais como observar o ambiente, localizar-se e ter uma representação interna do mundo. Executando estas tarefas, o robô sabe onde está e o que tem ao seu redor, mas não possui a habilidade para interagir com o ambiente ao seu redor. Para que esta interação aconteça, é preciso que um planejador de movimento entre em ação.

Um planejador de movimento tem o objetivo de descobrir quais são os movimentos necessários para que um robô saia do seu estado inicial e chegue a um estado desejado de forma segura e autônoma. Este problema pode ser resolvido de várias maneiras. Uma delas é dividi-lo em partes. O planejamento de movimento encontra um caminho em que o robô não bateria, sem considerar todas as restrições de

movimentação. Nesse caso, o planejamento pode ser resolvido discretizando o espaço de busca e aplicando algoritmos de otimização, como A* [10] ou D* [11]. A trajetória obtida pelo planejador pode ser impossível de ser executada por um robô, pois, em sua construção, nem todas as restrições de movimentação foram consideradas. Por isso, normalmente a trajetória é suavizada e outro algoritmo é aplicado para descobrir os comandos de velocidade e ângulo de direção do volante que devem ser enviados ao veículo.

Outra forma de lidar com o problema de planejamento de movimento é resolver tudo em um só passo. Para que isso seja possível, um modelo de movimento mais aproximado do veículo é utilizado durante o planejamento. Deste modo, as trajetórias encontradas podem ser executadas pelo carro sem necessidade de pós-processamentos. Com o uso de um modelo de movimentação mais preciso, o planejamento considera um espaço de busca maior, tornando-se mais difícil o uso de algoritmos clássicos de otimização. Nesse caso, tem sido comum a utilização de algoritmos baseados em amostragem, como Probabilistic Roadmaps (PRM) [12] ou Rapidly-Exploring Random Tree (RRT) [13]. Estes algoritmos trabalham melhor com espaços de busca maiores. Neste trabalho, o problema de planejamento de movimento é resolvido utilizando o algoritmo RRT.

A ideia básica do RRT é crescer uma árvore de forma progressiva de um estado inicial, aplicando controles de entrada em intervalos de tempo pequenos para alcançar novos estados aleatórios. Cada vértice na árvore representa um estado, e cada arco representa um comando que foi aplicado para alcançar o novo estado a partir do estado anterior. Quando o vértice alcança a região do destino, uma trajetória do estado inicial até o estado destino é representada pela árvore [14].

Neste trabalho, foram feitas diversas modificações no algoritmo RRT original, que tornaram possível gerar trajetórias em tempo real compatíveis com o modelo de movimentação de um veículo convencional e com o ambiente ao seu redor. Por exemplo, se o veículo estiver em uma rua, as trajetórias geradas devem mantê-lo em sua faixa e corrigir as incertezas relativas ao controle e sensoriamento de forma a evitar colisão e invasão de contramão.

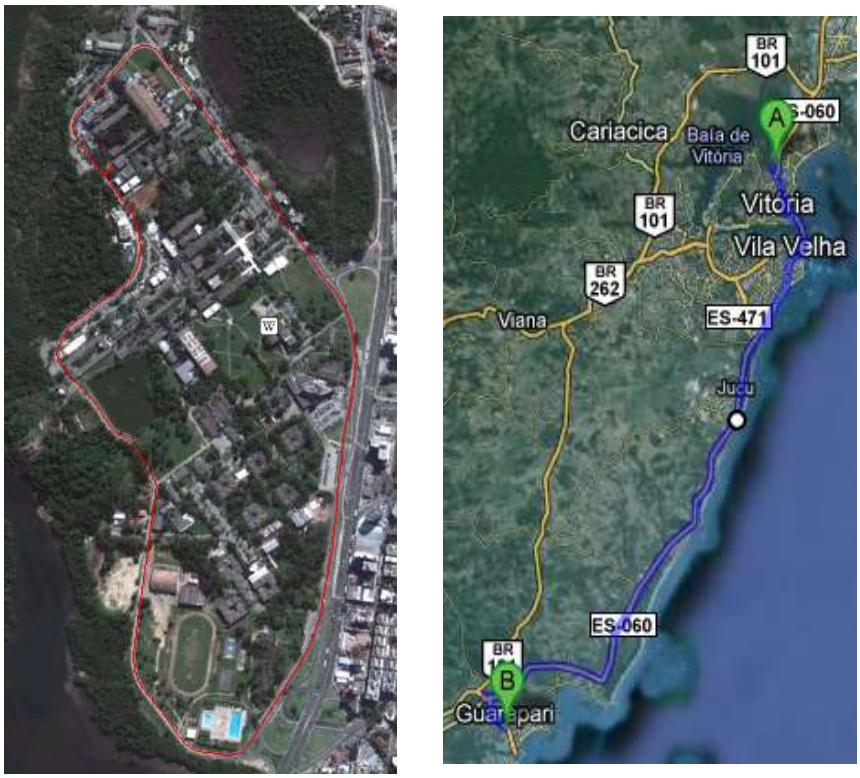
O desempenho do planejador de movimento foi avaliado em um arcabouço de simulação de veículos autônomos e na plataforma robótica *Intelligent Autonomous Robot Automobile* (IARA). IARA foi construída baseada no automóvel de passeio Ford Escape Hybrid adaptado com sensores e mecanismos para controlar o acelerador, freio, posição do volante, etc., por meio de computadores instalados em seu porta-malas. Resultados experimentais demonstram que o planejador de movimento é capaz de produzir trajetórias livres de colisão em tempo real. Além disso, as trajetórias produzidas pelo planejador de movimento podem gerar gastos de energia equivalentes àqueles geradas pelas trajetórias produzidas por motoristas.

1.1 Motivação

Este trabalho foi desenvolvido dentro de um projeto do Laboratório de Computação de Alto Desempenho (LCAD) da UFES. São dois os principais objetivos desse projeto: realizar a Volta da UFES e a Ida a Guarapari.

O campus principal da UFES (campus de Goiabeiras) possui um anel viário que o circunda que possui 3.570 metros (Figura 1-A). Na Volta da UFES, o objetivo é desenvolver pesquisas que viabilizem a transformação de um automóvel de passeio em um veículo capaz de realizar a volta pela universidade autonomamente. No caminho da Volta da UFES, as restrições impostas pelas leis de trânsito podem ser, em larga medida, desconsideradas, já que será possível realizá-la em um domingo ou feriado, situações em que o trânsito no anel viário é praticamente nulo. A distância da UFES à cidade de Guarapari, por outro lado, é de 58,5 Km (Figura 1-B). Na Ida a Guarapari o objetivo é desenvolver pesquisas que viabilizem o aperfeiçoamento do veículo autônomo de modo a torná-lo capaz de realizar a Ida a Guarapari autonomamente. No caminho da Ida a Guarapari todas as leis de trânsito se aplicam e terão que ser consideradas pelos algoritmos que comandarão e controlarão o veículo autônomo.

A motivação deste trabalho é então desenvolver um planejador de movimento para veículos convencionais, cumprindo assim uma das etapas do projeto.



(A) (B)

Figura 1 - Caminho da Volta da UFES (A) e Ida a Guarapari (B)

1.2 Objetivo

O objetivo deste trabalho é desenvolver um planejador de movimento para veículos convencionais que possui restrições cinemáticas e dinâmicas. Este planejador será utilizado em dois ambientes: rua e área livre. Na rua, ele deve ser capaz de manter o veículo na pista correta e evitar colisão com obstáculos móveis e dinâmicos. Em áreas livres, ele deve ser capaz de gerar trajetórias em que o veículo possa realizar manobras de estacionamento.

1.3 Contribuições

A principal contribuição do trabalho foi o desenvolvimento de um planejador de movimento baseado no RRT para um veículo com restrições cinemáticas e dinâmicas. O planejador foi testado extensivamente em simulações. Por diversas

vezes também foi testado em um Ford Escape Hybrid demonstrando bons resultados.

Para que o planejador funcionasse no veículo de forma adequada (gerasse trajetórias com comportamento consistente com o ambiente e em tempo aceitável), uma combinação de modificações foi proposta para o RRT:

- Uso da heurística RC-RRT
- Mapa de custo de Obstáculo
- Mapa de custo da Pista
- Goal Bias na Pista
- Mapa de Custo de Gradiente
- Uso do Reed-Sheep

Até onde pudemos examinar na literatura, esta combinação de técnicas empregada para resolver o problema de planejamento de movimento é única e os resultados obtidos são promissores.

1.4 Organização do Texto

Esta dissertação está dividida da seguinte forma:

Capítulo 2 – Problema do Planejamento de Movimento: define o problema de planejamento de movimento e apresenta as informações necessárias para resolvê-lo.

Capítulo 3 – Planejamento de Movimento para Veículos Convencionais Usando RRT: apresenta a solução proposta neste trabalho para planejamento de movimento para veículos convencionais usando RRT.

Capítulo 4 – Metodologia: apresenta o hardware, software e a métrica utilizada para avaliar o planejador de movimento desenvolvido neste trabalho.

Capítulo 5 – Experimentos e Resultados: apresenta os experimentos e os resultados da avaliação da abordagem proposta neste trabalho.

Capítulo 6 – Discussão: apresenta trabalhos correlatos e faz uma análise crítica do trabalho realizado.

Capítulo 7 – Conclusão e Trabalhos Futuros: apresenta as conclusões e direções para trabalhos futuros.

2 PROBLEMA DO PLANEJAMENTO DE MOVIMENTO

Este capítulo apresenta o problema de planejamento de movimento. Nele o problema será definido e as informações necessárias para desenvolvê-lo serão apresentadas.

2.1 Definição do Problema

O objetivo do planejamento de movimento é especificar uma tarefa em uma linguagem de alto nível e o robô autonomamente compilar a tarefa especificada em um conjunto de primitivas de movimentação de baixo nível a fim de completá-la [15]. Normalmente, a tarefa é encontrar uma trajetória para um robô de uma configuração para outra enquanto desvia de obstáculos, seja o robô um braço robótico, robô móvel ou até mesmo um piano que pode se movimentar livremente. No entanto, a necessidade de planejadores de movimento tem se estendido para outros campos além da robótica, como animação computadorizada (por exemplo, gerar movimento para avatares), *computer-aided design* (por exemplo, testar se um produto pode ser fabricado) e biologia (por exemplo, ajudar na análise de movimentação de moléculas) [15].

Neste trabalho, planejadores de movimento serão classificados da seguinte forma:

- **Planejamento de Movimento Holonômico:** considera que o robô não possui nenhuma restrição de movimentação; apenas a geometria do robô é analisada. Por exemplo, o problema de mover um piano livremente ou planejamento de um robô diferencial podem ser tratados em um planejador holonômico.
- **Planejamento de Movimento Não Holonômico:** considera restrições de movimentação não holonômicas; o próximo estado do veículo é determinado por uma função de movimentação. Por exemplo, um carro não pode se movimentar lateralmente, pois possui restrições Ackerman.
- **Planejamento de Movimento *Kinodynamic*** [16]**:** considera simultaneamente restrições cinemáticas e dinâmicas do robô, como

aceleração, velocidade e forças. O próximo estado do veículo é determinado por uma função que considera as restrições não holonômicas e também as acelerações, velocidades e limites que o robô sofre. Por exemplo, um veículo não se movimenta lateralmente, não para instantaneamente enquanto anda a 100 km/h e está sujeito a limites de aceleração na mudança de velocidade e ângulo de direção.

Neste trabalho, o planejamento de movimento será utilizado no contexto de veículos convencionais. Por esse motivo, o planejamento empregado será o *kinodynamic*. Para realizá-lo, algumas informações devem ser conhecidas:

- Mapa que represente o ambiente ao redor do robô;
- Estado do robô, que inclui sua localização no mapa (dada pelas coordenadas x, y e pela orientação), velocidade e ângulo da roda dianteira;
- Estado destino, dado por uma posição ou grupo de posições que indiquem onde o robô deve ir;
- Função que simule a movimentação do robô.

Ao término do planejamento de movimento, uma trajetória é gerada, composta por uma lista de comandos e estados que informam como o robô chega ao estado destino.

2.2 Formulação do Problema

Este trabalho utilizará a seguinte formulação do problema [14]:

1. Espaço de estados: espaço topológico, X.
2. Limite de Valores: $x_{init} \in X$ e $X_{goal} \subset X$
3. Detecção de Colisão: uma função, $D: X \rightarrow \{true, false\}$, que determina se as restrições de movimentação e colisão do estado x foram satisfeitas. Uma restrição de colisão é satisfeita quando o estado x está em uma região livre de obstáculos.
4. Entradas: um conjunto, U, que especifica uma série de controles ou ações que podem afetar um estado.

5. Simulação incremental: dado o estado atual, $x(t)$, e entradas aplicadas em um dado intervalo de tempo, $\{u(t')|t \leq t' \leq t + \Delta t\}$, computar $x(t + \Delta t)$.
6. Métrica: Uma função que retorne um valor real, $p: X \times X \rightarrow [0, \infty)$, com respeito à distância entre dois pontos em X.

2.3 Mapa

Um dos itens necessários para realizar o planejamento de movimento é uma representação do mundo. Esta representação é utilizada pelo algoritmo para descobrir onde o robô pode trafegar com segurança e o que é obstáculo. Neste trabalho, o mundo é representado a partir de um mapa de ocupação de *grid* bidimensional [17].

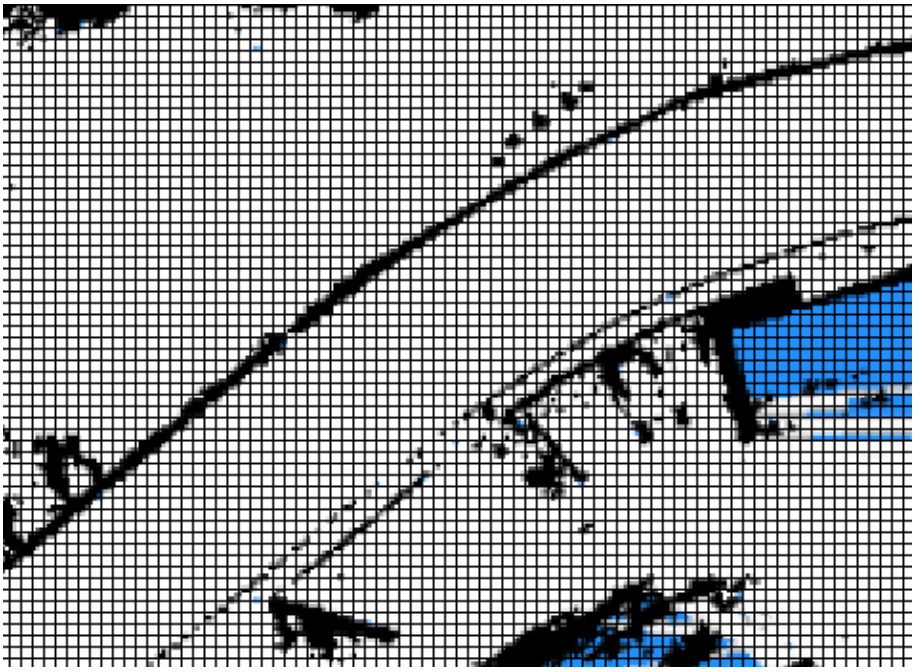


Figura 2 - Mapa de ocupação g*rid*

Cada célula do *grid* representa um espaço fixo no mundo real; o tamanho desse espaço é proporcional à resolução do mapa. Por exemplo, em um mapa com resolução de 0,2 m, cada célula representa 0,2 m² do mundo real. Cada posição do mapa pode conter duas informações:

1. Probabilidade de obstáculo: valor que varia entre 0 e 1; quanto mais próximo de 1 maior a probabilidade de existir um obstáculo;
2. Local desconhecido: caso não exista nenhuma observação de sensores sobre a célula, seu valor é -1.

A Figura 2 mostra um mapa de grid. Células em branco representam células com valores próximos a 0, células pretas representam valores próximos a 1 e células azuis representam células com valores igual a -1.

Para descobrir se um robô está em uma região livre, cada valor de célula abaixo da posição do robô é verificado. Se alguma dessas células possuírem valor acima de um determinado limiar, o robô está em uma posição não trafegável; caso contrário, está em uma posição trafegável.

2.4 Modelo do Robô

Uma parte importante para o planejamento de movimento é saber como o robô sai de sua posição atual e vai para uma posição vizinha. Em casos holonômicos, qualquer posição vizinha ao robô é acessível, mas, no caso não holonômico, o robô se movimenta de acordo com uma função de movimentação e, por isso, nem todas as posições vizinhas são acessíveis.

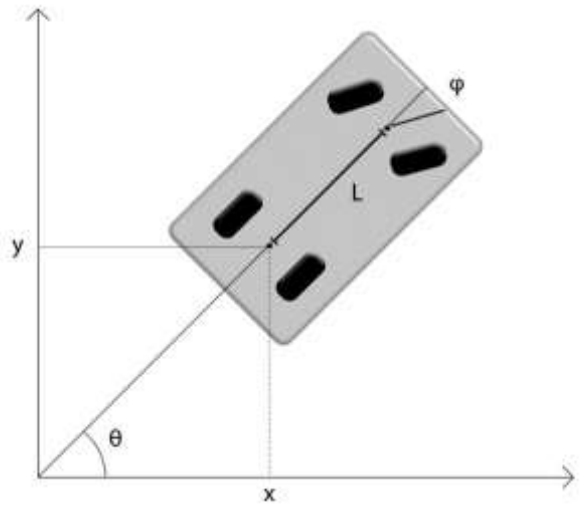


Figura 3 - Modelo cinemático simples de um veículo Ackerman

A Figura 3 ilustra um modelo cinemático simples para representação de movimento Ackerman, este modelo é muito utilizado [18] [19] [20] [21]. Nela, a posição do veículo é dada pelas coordenadas x e y localizadas entre as rodas traseiras do veículo e a orientação é dada pelo ângulo θ. O ângulo da roda da frente é dado por φ, que é a média do ângulo das rodas dianteiras direita e esquerda. A velocidade do veículo é denotada por v.

O vetor de estado do veículo é dado por:

$$\boldsymbol{x} = [x, y, \theta, v, \varphi],$$

e o vetor de controle por:

$$\boldsymbol{u} = [v, \varphi, t].$$

O modelo cinemático de movimentação do veículo é dado por:

$$x_{t+1} = x_t + \Delta t v_t \cos\theta_t,$$

$$y_{t+1} = y_t + \Delta t v_t \sin\theta_t \text{ e}$$

$$\theta_{t+1} = \theta_t + \Delta t v_t \frac{\tan\varphi_t}{L},$$

onde L é a distância das rodas dianteiras para as rodas traseiras e Δt é a discretização do tempo.

O desenvolvimento de um modelo de predição de movimento de veículos com fidelidade alta é importante para gerar predições de movimento e trajetórias precisas [22]. Para que o planejador de movimento tenha bons resultados, é importante a utilização de um modelo fiel ao do veículo. Por isso, este trabalho utiliza o modelo cinemático simples adicionado das seguintes restrições:

- Aceleração máxima;
- Desaceleração máxima;
- Taxa de curvatura máxima.

Um modelo ainda mais complexo pode ser utilizado adicionando outras restrições, como por exemplo, a latência de envio de comandos e o limite da velocidade em função do ângulo da roda atual [22].

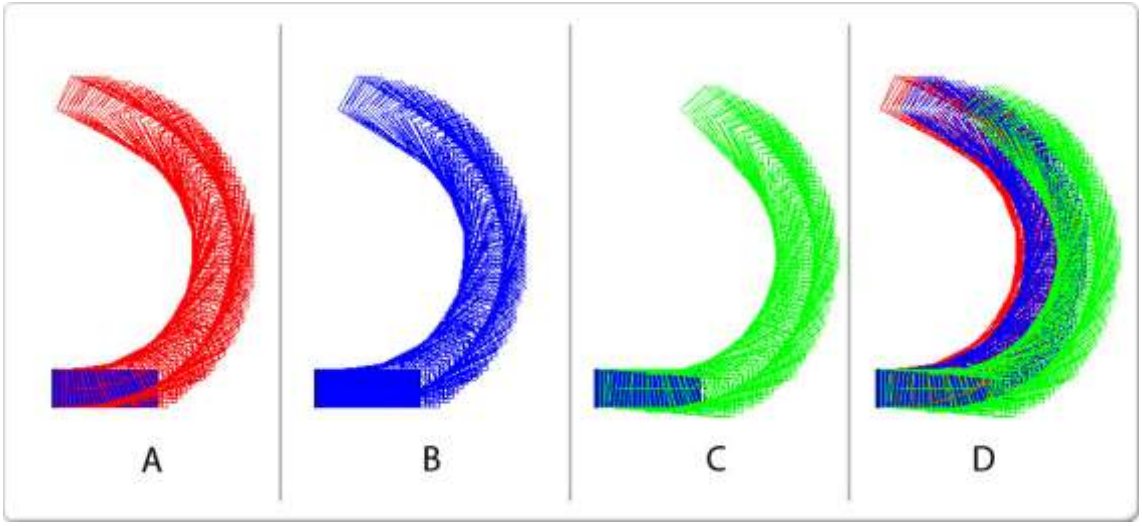


Figura 4 - Comparação modelagem do veículo

A Figura 4 ilustra a diferença de uma modelagem de veículo mais simples de outra que considera aceleração. O robô é representado pelo retângulo azul e o comando dado para gerar as trajetórias foi $\boldsymbol{u} = [1, 25, 15]$. A figura está dividida da seguinte forma:

A. Modelo do veículo simples foi utilizado com estado inicial $\boldsymbol{x} = [0, 0, 0, 0, 0]$.
B. Modelo do veículo que considera acelerações foi utilizado com estado inicial $\boldsymbol{x} = [0, 0, 0, 1, 0]$.
C. Modelo do veículo que considera acelerações foi utilizado com estado inicial do robô $\boldsymbol{x} = [0, 0, 0, 1, -25]$.
D. Sobreposição das trajetórias A, B e C.

Na Figura 4-D, é possível notar a diferença entre as trajetórias. O modelo de movimentação do A não considera aceleração. Por isso, o veículo consegue virar o volante instantaneamente para 25 graus e começa a fazer a curva imediatamente. Na trajetória B, é possível notar uma curva mais aberta. Isso acontece porque o modelo de movimentação utilizado não permite que o veículo vire 25 graus instantaneamente. Antes disso, ele passa por ângulos de volante intermediários (de 0 grau a 25 graus). Na trajetória C, o efeito da aceleração é notado com mais facilidade. O veículo inicialmente está com o ângulo do volante a -25. Por isso, a trajetória é iniciada com um ângulo de direção oposta ao comando e aos poucos converge para o ângulo desejado.

A Figura 4 exemplifica como o modelo de movimentação é importante para o planejamento de movimento. Por exemplo, considere um veículo em uma pista simples prestes a fazer uma curva fechada a esquerda. Se um planejador utilizando um modelo pouco fiel gerar o caminho da Figura 4-A, mas o veículo estiver com o volante girado para o ângulo oposto ao comando, ele executaria um caminho parecido com o da Figura 4-C, invadindo assim a contra mão e correndo o risco de colidir com outro veículo.

2.5 Rapidly-exploring Random Tree

Esta seção tem como objetivo apresentar o algoritmo básico Rapidly-Exploring Random Tree (RRT). Este algoritmo foi apresentado pela primeira vez em [13]. RRT é uma estrutura de dados aleatória projetada para uma ampla gama de problemas de planejamento de caminho. RRT foi especificamente projetado para suportar restrições não holonômicas (incluindo dinâmicas) e alto grau de liberdade.

A ideia básica do RRT é crescer uma árvore de forma progressiva de um estado inicial, aplicando controles de entrada em intervalos de tempo pequenos para alcançar novos estados aleatórios. Cada vértice na árvore representa um estado, e cada arco representa um comando que foi aplicado para alcançar o novo estado a partir do estado anterior. Quando o vértice alcança a região do destino, uma trajetória do estado inicial até o estado destino é representada pela árvore [14].

A Figura 5 ilustra a construção de um RRT. A árvore começa com apenas um vértice inicial, x_{init}. A cada iteração, um vértice x_{rand} é gerado de forma aleatória e um vértice próximo a x_{rand}, x_{near}, é selecionado na árvore. Um comando que leva x_{near} em direção a x_{rand} é selecionado. Aplicando esse comando a x_{near} por um intervalo de tempo o vértice x_{new} é gerado e então x_{new} e o arco que liga x_{near} a x_{new} são adicionados a árvore.

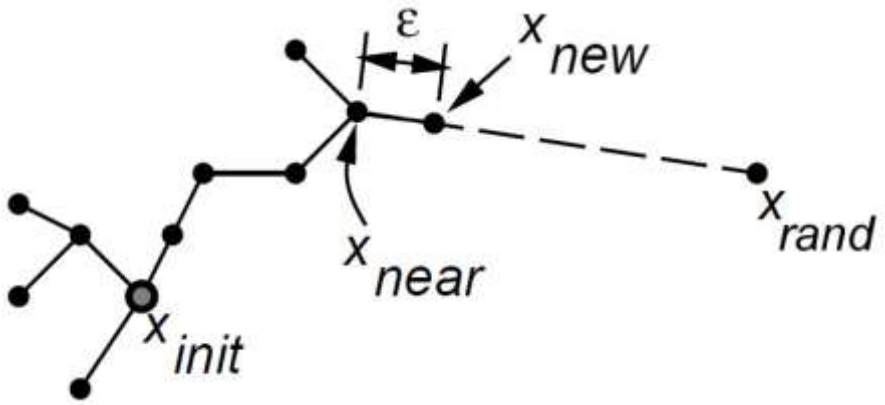


Figura 5 - Construção de um RRT [14]

Planejamento de movimento geralmente é visto como uma busca em um espaço de estados, X, por um caminho contínuo do estado inicial, x_{ini} até uma região de destino $X_{goal} \subset X$ ou um estado específico de destino x_{goal}. Para planejamento de movimento holonômico, $X = C$, que é o espaço de configurações de um corpo rígido em um mundo 2D ou 3D. Para um planejamento de movimento *kinodynamic*, o estado X codifica configuração e velocidades.

É assumido que uma região de obstáculos fixos, $X_{obs} \subset X$, deve ser evitada, e que uma representação explícita de X_{obs} não está disponível. Porém, é possível checar se um dado estado está contido em X_{obs}. Estados em X_{obs} podem corresponder a velocidades além do limite, configurações em que o robô colidiria com um obstáculo, ou outras interpretações dependendo da aplicação. O RRT é construído de forma que todos seus vértices são estados em X_{free}, o complemento de X_{obs}. Além disso, todo arco do RRT corresponde a um caminho que está inteiramente em X_{free}.

Uma equação de transição de estado na forma de $\dot{\mathrm{x}} = \mathrm{f}(\mathrm{x}, \mathrm{u})$ é definida para expressar as restrições *kinodynamic* ou não holonômicas. O vetor u é selecionado de um conjunto, U, de entradas. O vetor $\dot{\mathrm{x}}$ é o estado derivativo com relação ao tempo. Essa representação de controle teórica é poderosa o suficiente para codificar virtualmente qualquer tipo de modelo cinemático ou dinâmico. Integrando f em um intervalo de tempo fixo, Δt, o próximo estado, x_{new} pode ser determinado para um dado estado inicial, x, e entradas $u \in U$. Usando Euler, $x_{new} \approx x + f(x, u)\Delta t$.

Para um planejamento holonômico, depois de integrar a função de simulação de movimento f em Δt, um novo estado pode ser obtido que move o sistema em qualquer direção relativa a x. Em um problema não holonômico, o próximo estado é restrito de acordo com a escolha da função f.

```
BUILD_RRT(x_init)
{
1.    T.init(x_init);
2.
3.    for (k = 0; k < K; k++)
4.    {
5.          x_rand = RANDOM_STATE();
6.          x_near = NEAREST_NEIGHBOR(x_rand, T);
7.          EXTEND(x_near, x_rand);
8.    }
}
```

Figura 6 - Pseudocódigo BUILD_RRT

A Figura 6 ilustra o algoritmo para construção de um RRT básico. A função BUILD_RRT recebe de entrada um estado inicial, $x_{int} \in X_{free}$. Uma iteração simples é feita onde cada passo tenta estender o RRT adicionando um novo vértice na direção de um estado gerado aleatoriamente. A iteração do *loop* acontece da seguinte forma. Primeiro, um estado aleatório é sorteado pelo RANDOM_STATE. NEAREST_NEIGHBOR então seleciona o vértice mais próximo da árvore em relação ao vértice aleatório. O vértice mais próximo é escolhido de acordo com uma métrica, p. A escolha da métrica deve ser feita de acordo com o problema. Uma métrica tipicamente utilizada é a distância euclidiana entre dois pontos.

```
EXTEND(x_near, x_rand)
{
1.    if (NEW_STATE(x_near, x_rand, x_new, u_new))
2.    {
3.          T.add_vertex(x_new);
4.          T.add_edge(x_near, x_new, u_new);
5.
6.          if (x_new == X_goal)
7.                return REACHED;
8.          else
9.                return ADVANCED;
10.   }
11.   return TRAPPED;
}
```

Figura 7 - Pseudocódigo EXTEND

A Figura 7 mostra a função EXTEND que recebe o vértice mais próximo e o vértice aleatório e chama NEW_STATE. Esta função tenta encontrar um movimento de x_{near} na direção de x_{rand} aplicando um comando $u \in U$ com certo incremento de tempo, Δt. Esse comando pode ser escolhido tentando todos os possíveis comandos e selecionando o que gera o novo estado mais próximo do estado aleatório, x_{rand}. NEW_STATE também utiliza a função de detecção de colisão para determinar se o novo estado (e todos os estados intermediários) satisfaz todas as restrições (de movimentação e de colisão). Se NEW_STATE obtém sucesso, um vértice que representa o novo estado x_{new} e um arco que representa o comando u_{new} que leva x_{near} a x_{new} são adicionados à árvore. EXTEND pode reportar quatro situações:

1. GOAL_REACHED: o novo vértice x_{new} alcançou X_{goal} ou em planejamentos não holonômicos está perto suficiente de X_{goal}.
2. ADVANCED: um novo vértice x_{new} é adicionado ao RRT.
3. TRAPPED: NEW_STATE falhou em produzir um estado que esteja em X_{free}.
4. REACHED: o novo vértice x_{new} alcançou x_{rand}.

Quando a função EXTEND retorna REACHED, significa que a árvore já se expandiu até X_{goal} e uma trajetória foi encontrada. Entretanto, BUILD_RRT pode continuar até o fim do loop com o objetivo de encontrar soluções melhores.

2.5.1 Propriedades do RRT

As principais propriedades do RRT são:

- A expansão do RRT tem uma forte tendência em direção a regiões não exploradas.
- A distribuição de vértices no RRT se aproxima da distribuição das amostragens.
- O RRT é probabilisticamente completo, ou seja, a probabilidade de encontrar uma solução, caso ela exista, se torna 1 quando o tempo de execução vai para infinito.

- RRT pode ser considerado como um módulo de planejamento de caminho que pode ser incorporado em uma vasta gama de aplicações de sistemas de planejamento.

2.5.2 Heurísticas Básicas do RRT

Em [14] uma análise foi feita sobre diversos tipos de RRTs, apresentando os pontos que necessitam de mais atenção.

O RRT básico pode ser usado isoladamente como planejador, porque seus vértices eventualmente cobrirão o espaço de x_{free} e chegarão próximo a um estado destino x_{goal} específico. O problema é que sem nenhum *bias* em direção ao destino, a convergência pode ser lenta.

Um planejador melhorado, chamado **RRT-GoalBias**, pode ser obtido substituindo RANDOM_STATE por uma função que retorna o estado destino com uma dada probabilidade; caso contrário, retorna um estado aleatório. Mesmo com uma probabilidade muito baixa, RRT-GoalBias geralmente converge para o destino muito mais rápido que um RRT básico. Se um *bias* muito alto for introduzido, o planejador pode ficar preso em mínimos locais.

Uma versão melhorada, chamada **RRT-GoalZoom**, substitui RANDOM_STATE por uma função que decide entre uma amostra aleatória de uma região ao redor do destino ou do espaço de estado completo. O tamanho da região em volta do destino é controlado pela distância do vértice mais próximo do destino em cada iteração. O efeito é que a concentração das amostras gradualmente é aumentada em volta do destino à medida que a árvore se aproxima. Esse planejador tem um bom desempenho na prática. Entretanto, é possível que haja degradações devido a mínimo local.

Outra questão importante é o tamanho dos passos usados para construir um RRT. Esse tamanho poderia ser escolhido dinamicamente durante a execução. Se o estado está longe de uma colisão, então passos maiores poderiam ser utilizados. Ao invés de estender RRT em passos incrementais, usando estados aleatórios

diferentes a cada passo, EXTEND pode ser usado iterativamente até que o estado destino, estado aleatório ou um obstáculo seja alcançado.

A Figura 8 mostra a função **CONNECT** que pode substituir EXTEND, fazendo com que o RRT cresça rapidamente, caso permitido pelas restrições de colisão e de movimentação. Uma das principais vantagens da função CONNECT é que um caminho longo pode ser construído com apenas uma chamada do algoritmo NEAREST_NEIGHBOR. Entretanto, se um vizinho mais próximo eficiente for utilizado, então pode fazer mais sentido utilizar o EXTEND.

```
CONNECT(x_near, x_rand)
{
1.     do
2.     {
3.           status, x_near = EXTEND(x_near, x_rand);
4.     } while(status == ADVANCED);
5.     return status;
}
```

Figura 8 - Função CONNECT

3 PLANEJAMENTO DE MOVIMENTO PARA VEÍCULOS CONVENCIONAIS USANDO RRT

Este capítulo apresenta a solução proposta neste trabalho para planejamento de movimento de veículos convencionais que possuem restrição de movimento (utiliza modelo de direção Ackerman) e restrições dinâmicas, o que leva a um planejamento de movimento *kinodynamic*. Neste tipo de planejamento, o espaço de busca é grande, o que é um problema para algoritmos de planejamento tradicionais. Por isso, o RRT foi utilizado como base no planejador de movimento. Ele funciona bem com problemas com muitas dimensões e possui uma boa estrutura para lidar com modelos de movimento com restrições *kinodynamic*. Para isso, basta implementar $f(x,u)$ com base no modelo cinemático de movimentação de um veículo (Seção 2.4).

O RRT padrão tem suas vantagens, mas também tem problemas conhecidos relacionados à métrica e convergência. As próximas seções discutirão os problemas identificados e as soluções utilizadas para minimizá-los.

3.1 Métrica

A principal desvantagem de métodos baseados em RRT é a sensibilidade de desempenho de acordo com a escolha da métrica [14]. Todos os algoritmos de planejamento aleatórios sofrem da dificuldade de se determinar ou estimar a métrica ideal [13]. Mesmo utilizando métricas ruins, o resultado do RRT continua probabilisticamente completo. Entretanto, problemas de convergência se tornam ainda mais importantes [13]. Para planejamento *kinodynamic*, a métrica ideal (ou pseudo-métrica devido à assimetria) é a que retorna o custo da trajetória ótima entre dois estados [13] [14]. Entretanto, determinar esta métrica é pelo menos tão difícil quanto resolver o problema original [13] [14].

A Figura 9 será utilizada para ilustrar o problema da métrica. Nela, os retângulos 1 e 2 representam vértices do RRT e o estado 3 representa um estado aleatório do veículo.

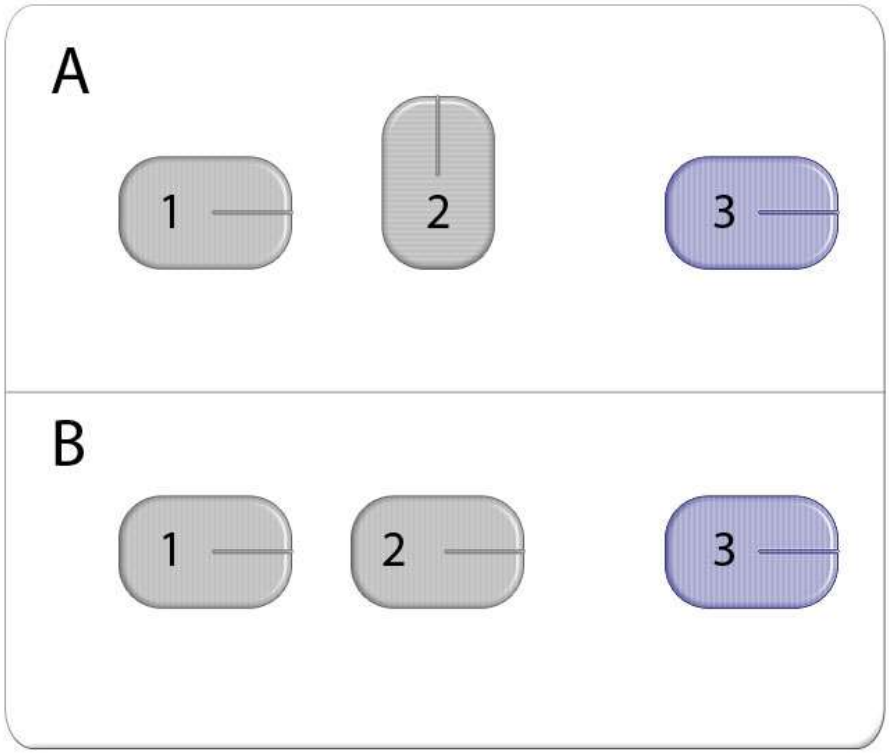


Figura 9 - Problema da métrica

Considere a função NEAREST_NEIGHBOR do RRT, que recebe um vértice aleatório, x_{rand}, e retorna o vértice mais próximo da árvore, x_{near}. Se a métrica de distância euclidiana for utilizada na Figura 9-A, a função NEAREST_NEIGHBOR retornará o vértice 2, pois é o vértice que se encontra mais próximo do 3. Porém essa escolha não é ideal, pois para que o vértice 2 alcance o estado 3, várias manobras teriam que ser executadas. Porém, se o vértice 1 fosse escolhido, o veículo teria apenas que seguir em frente. Apesar do vértice 2 estar mais perto do 3, o custo para se movimentar do estado 2 para o estado 3 é superior ao custo para se movimentar do estado 1 para o estado 3.

Outras métricas podem ser utilizadas para que NEAREST_NEIGHBOR tenha resultados melhores. Por exemplo, no trabalho [23] a distância de Dubin foi utilizada para reduzir os problemas com métrica. Esta distância utiliza as curvas de Dubin, que resolvem o problema de cinemática inversa em robôs com restrições Ackerman que têm apenas as seguintes possibilidades de movimento: seguir em frente, virar o máximo para a esquerda ou para a direita. Caso a distância de Dubin seja usada como métrica, na Figura 9-A o NEAREST_NEIGHBOR retornaria o vértice 1, o caso ideal.

Na Figura 9-B, independente da métrica utilizada (euclidiana ou Dubin) a função NEAREST_NEIGHBOR retornará o vértice 2, o que é um bom resultado caso a velocidade do vértice 2 seja maior ou igual à velocidade do vértice 1. Entretanto, se a velocidade do vértice 1 for maior que a velocidade do vértice 2, estas métricas podem gerar resultados ruins. Por exemplo, considerando que o estado 2 tenha v=0 m/s e o estado 1 tenha v=10m/s o melhor retorno de NEAREST_NEIGHBOR seria o estado 1, pois ele pode alcançar o estado 3 muito mais rápido (menor custo).

Outro problema de escolher o vértice mais próximo apoiando apenas na métrica é não considerar obstáculos. Por exemplo, um vértice pode ser escolhido como mais próximo de um determinado estado aleatório, mas uma parede pode existir entre os dois, o que impossibilitaria x_{near} de avançar em direção de x_{rand}.

Métricas mal escolhidas podem causar problemas como dificuldade de convergência e trajetórias que divergem da ótima [14]. Entretanto, o desenvolvimento de uma métrica ideal é muito difícil. Por isso, diferentes estratégias têm sido utilizadas para melhorar o desempenho do RRT mesmo utilizando métricas imperfeitas, como por exemplo: Rechability-Guided RRT (RG-RRT) [24] e Resolution Complete RRT (RC-RRT) [25]. RG-RRT torna a seleção de vértices menos sensível a métrica. Isto é atingido armazenando uma lista de estados possíveis de serem alcançados para cada vértice. Esta lista então é utilizada na escolha do vértice para a expansão. O vértice selecionado é aquele que tem em sua lista o estado mais próximo do estado aleatório (dentre todos os estados possíveis de serem alcançados a partir de todos os vértices da árvore). Isto aumenta a probabilidade que o vértice escolhido gere estados na direção do estado aleatório. Já o RC-RRT ameniza o problema da métrica adicionando a cada vértice a informação de quão promissor ele é. Esta informação é obtida pelo resultado de falhas em expansões passadas. A seleção de vértices para expansão passa a priorizar os vértices considerados mais promissores.

Neste trabalho, foi adotada como métrica a distância euclidiana em conjunto com a heurística RC-RRT descrita em detalhes a seguir.

3.1.1 Heurística RC-RRT

Segundo [25], a degradação do desempenho do RRT ocorre pelas seguintes razões:

1. O RRT escolhe x_{near} dependendo apenas da métrica, p. Quando p fornece informações ruins, a exploração do espaço inteiro é dificultada. Outro problema é que se apenas p for considerada, os estados podem ser escolhidos inúmeras vezes, mesmo que seus novos estados estejam destinados a resultar em colisão.
2. O comando, u_{best} , que leva x_{near} para x_{new} é selecionado baseando-se apenas em p. Isto pode levar a caminhos ruins. O comando que gera boa exploração pode ter sido descartado, porque o novo estado, x_{new}, derivado do comando, está mais distante de x_{rand} que outros possíveis estados derivados de outros comandos.

O RC-RRT resolve esses problemas discretizando os possíveis comandos U. Uma estrutura de dados associada a cada vértice armazena uma lista de comandos, u, que já foram aplicados. Cada comando é aplicado apenas uma vez em um vértice. Caso todos os possíveis comandos forem aplicados, o vértice é descartado de futuras expansões.

Além disto, vértices são penalizados quando expansões têm a probabilidade de falhar, baseado na Frequência de Violação de Restrições (Constraint Violation Frequency – CVF). Cada novo vértice inicialmente tem CVF igual a 0. Quando um comando leva a colisão ou viola alguma restrição do veículo (por exemplo, velocidade acima do limite), o CVF é aumentado em $1/m$, onde m é o número de possíveis comandos discretizados em U. Além disto, o CVF do vértice parente é aumentado em $1/m^2$. Aplicando recursividade, o CVF do k_ésimo parente é incrementado em $1/m^{k+1}$. Cada CVF é limitado entre 0 e 1. Durante a seleção de vértice, a probabilidade de descarte de um vértice é igual ao valor de seu CVF. A Figura 10 mostra a função NEAREST_NEIGHBOR modificada utilizando as melhorias apresentadas.

```
NEAREST_NEIGHBOR(x_rand, T)
{
1.     d_min = ∞;
2.     for all x in T
3.     {
4.           if (x.cvf != 1.0)
5.           {
6.                 if (random() > x.cvf)
7.                 {
8.                       d = p(x,x_rand);
9.                       if (d < d_min)
10.                      {
11.                            d_min = d;
12.                            x_best = x;
13.                      }
14.                }
15.          }
16.    }
17.    return x_best;
}
```

Figura 10 - NEAREST_NEIGHBOR modificado

Com as melhorias proporcionadas pelo RC-RRT, caso um vértice seja escolhido várias vezes, comandos que já foram executados não serão avaliados novamente, o que economiza processamento, proporcionando uma melhora geral no desempenho do algoritmo. Armazenar os comandos já executadas por um vértice ainda evita que o crescimento da árvore estagne, pois impede o caso em que um mesmo vértice seja selecionado várias vezes e um u_{best} já executado seja escolhido gerando assim um x_{new} repetido. Porém, o RC-RRT não previne o caso em que sucessivas expansões geram um ciclo na árvore e vértices repetidos sejam adicionados. Este problema pode fazer com que o planejamento continue executando, mesmo que todo espaço já tenha sido explorado. Para evitar que isto aconteça basta proibir a inserção de vértices considerados repetidos na árvore. Essa solução foi denominada como RC-RRT$_2$ [26].

A Figura 11 mostra o algoritmo EXTEND com as modificações necessárias para impedir vértices repetidos na árvore.

```
EXTEND(x_near, x_rand)
{
1.    if (NEW_STATE(x_near, x_rand, x_new, u_new))
2.    {
3.          if (!T.add_vertex(x_new))
4.          {
5.                if (x_new.cost >= x_tree.cost)
6.                      return TRAPPED;
7.                x_new = x_tree;
8.          }
9.          T.add_edge(x_near, x_new, u_new);
10.
11.         if (x_new == X_goal)
12.               return REACHED;
13.         else
14.               return ADVANCED;
15.   }
16.   return TRAPPED;
}
```

Figura 11 - EXTEND modificado

Na linha 3, a função *add_vertex* foi modificada. Antes de inserir o x_{new} no RRT, uma chave é gerada levando em conta todo seu estado (x, y, θ, v, φ). Caso a chave já exista, a função retorna false; caso contrário a função retorna *true* e x_{new} é adicionado no RRT. Note que dependendo da resolução adotada nos cálculos para gerar as chaves, dois estados muito parecidos podem gerar chaves iguais. Na função EXTEND, caso *add_vertex* falhe (retorne *false*) e o custo de x_{new} seja maior ou igual ao custo de seu vértice equivalente na árvore, x_{tree}, a função EXTEND retornará TRAPPED cancelando assim a expansão. Caso contrário, o algoritmo prossegue e adiciona o arco que liga x_{near} a x_{tree} na árvore.

3.2 Convergência

A convergência do RRT está fortemente relacionada à métrica utilizada. O tempo de processamento em alguns problemas varia dramaticamente com a variação da métrica [14]. As modificações da seção anterior, utilizadas para minimizar o problema com a métrica, também melhoram a convergência do algoritmo. Ao utilizar o RC-RRT, o algoritmo se torna *resolution complete*, ou seja, uma solução sempre será encontrada se tal existir em uma determinada resolução. Observe que caso seja adotada uma resolução de alta granularidade nos cálculos das chaves, uma

solução pode não ser encontrada; e caso seja adotada uma resolução de baixa granularidade, o tempo para encontrar uma solução pode ser alto. Sabe-se que uma solução será encontrada, mas não há indicações de quão rápido o algoritmo convergirá. Algumas modificações foram feitas no RRT que o ajudam a convergir mais rápido. Elas foram baseadas nos dois problemas a seguir:

1. Mesmo após encontrar uma primeira solução, o RRT continua crescendo para regiões que não tem chance de melhorar a solução encontrada.
2. Se informações do ambiente não forem consideradas a árvore crescerá de forma uniforme, indo para direções indesejadas. Por exemplo, se o objetivo for gerar uma trajetória mantendo o carro na pista a árvore deveria focar o crescimento nos limites da pista.

Para tratar o primeiro problema, uma solução proposta foi podar os vértices da árvore que não têm chance de gerar trajetórias melhores. O RRT mantém o custo da melhor trajetória, $cost_{goal}$. Inicialmente, $cost_{goal} = \infty$. Quando uma nova trajetória é encontrada, o seu valor é atualizado para $cost_{goal} = \min(cost_{goal}, cost_{new\_traj})$, onde $cost_{new\_traj}$ é o custo da nova trajetória encontrada. Além disso, todos os vértices da árvore que não têm chance de gerar trajetórias melhores são removidos. Estes vértices são identificados pela seguinte condição:

$$\left(x_{new}.cost + \text{cost}\left(x_{new}, x_{goal}\right)\right) \geq cost_{goal}$$

onde $x_{new}.cost$ é o custo gasto para que x_{init} alcance x_{new} e $\text{cost}\left(x_{new}, x_{goal}\right)$ é uma função que retorna o custo mínimo gasto para que x_{new} alcance x_{goal}. A Figura 12 mostra o algoritmo EXTEND modificado para descartar x_{new}, que não tem chance de gerar trajetórias melhores que a atual.

Com esta solução, sempre que uma trajetória é encontrada, uma limpeza é feita na árvore, eliminando os vértices que não têm chance de fazer parte de uma solução melhor. Esta melhoria permite que o RRT encontre de forma mais fácil trajetórias próximas da ótima, pois não é gasto tempo com vértices não promissores.

```
EXTEND(x_near, x_rand)
{
1.    if (NEW_STATE(x_near, x_rand, x_new, u_new))
2.    {
3.          if ((x_new.cost + cost(x_new, x_goal)) >= cost_goal)
4.                return TRAPPED;
5.          if (!T.add_vertex(x_new))
6.          {
7.                if (x_new.cost >= x_tree.cost)
8.                      return TRAPPED;
9.                x_new = x_tree;
10.         }
11.         T.add_edge(x_near, x_new, u_new);
12.
13.         if (x_new == X_goal)
14.               return REACHED;
15.         else
16.               return ADVANCED;
17.   }
18.   return TRAPPED;
}
```

Figura 12 - EXTEND modificado 2

Podar vértices que não têm chances de gerar trajetórias melhores faz com que o RRT pare de crescer em direção a regiões não promissoras depois que uma primeira solução é encontrada. Porém, o RRT pode demorar a encontrar a primeira trajetória, sobrando pouco tempo para encontrar soluções melhores. Para tratar este segundo problema, uma solução proposta foi considerar informações sobre a estrutura do ambiente para que o foco do crescimento da árvore seja em regiões promissoras. Neste trabalho, o RRT foi utilizado para planejamento de movimento em dois ambientes:

- Rua: o ambiente é estruturado e o carro deve sempre andar na sua faixa da pista.
- Área livre: não existe nenhuma estrutura, mas é esperado que uma trajetória próxima da ótima seja gerada.

A seguir, será explicado como o problema foi resolvido em cada um destes casos.

3.2.1 Rua

Um dos casos em que o RRT é utilizado é para planejamento de movimento na rua. A Figura 13 mostra um mapa típico desse ambiente. Este mapa representa um trecho de faixa simples do anel da UFES.

Figura 13 - Mapa de um trecho do anel da UFES

Ao planejar movimentos na rua, as trajetórias geradas deveriam ter comportamento parecido com uma trajetória feita por um humano. Entretanto, se o RRT fosse utilizado sem nenhuma modificação, trajetórias que saem com frequência do limite da pista seriam esperadas. Foram considerados importantes os seguintes comportamentos:

- Sempre que possível, manter o veículo no centro de sua faixa, evitando ao máximo invadir à contra mão;
- Evitar a execução de manobras na pista, como marcha ré.

Para que a trajetória gerada tenha essas características, algumas modificações tiveram que ser feitas no RRT, apresentadas a seguir.

3.2.1.1 Mapa de Custo da Pista

Foi introduzido ao RRT um mapa de custo da pista. A Figura 14 ilustra este mapa, que foi construído a partir do mapa da Figura 13.

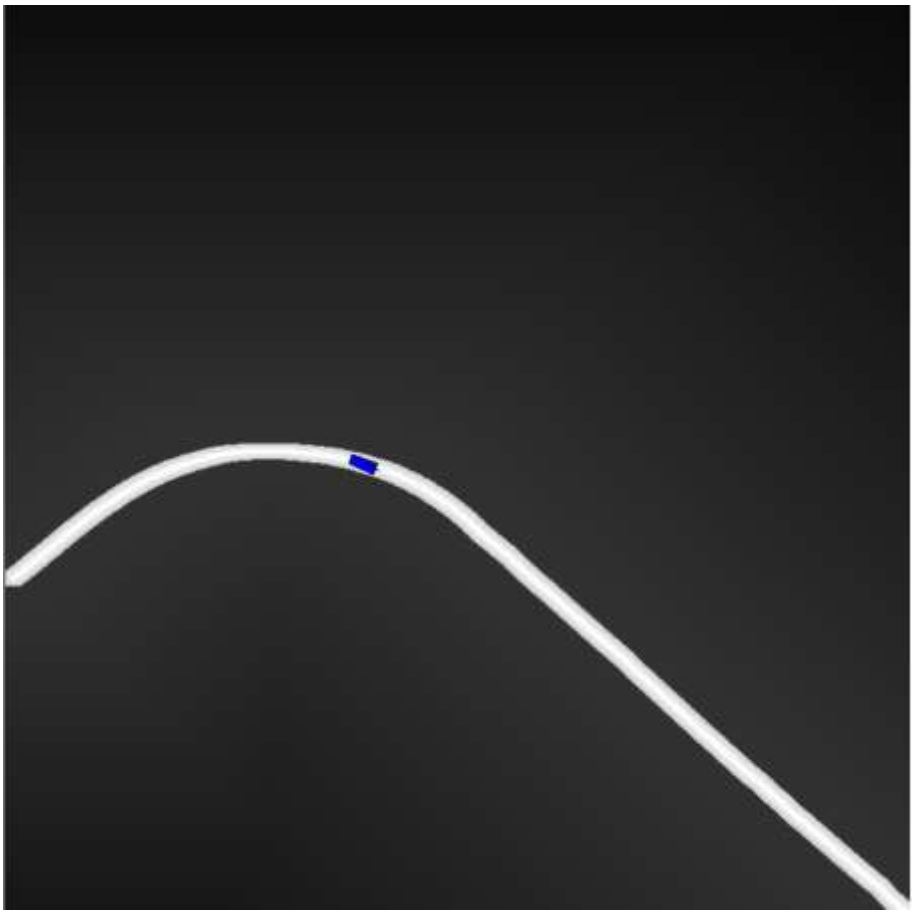


Figura 14 - Mapa de custo de pista

Para construir o mapa de custo de pista é preciso conhecer os pontos que correspondem ao centro da pista. Estes pontos poderiam ser adquiridos automaticamente a partir do processamento de imagens da pista, ou manualmente em que o usuário a partir de um mapa selecionaria as regiões de pista. Neste trabalho, uma estratégia semiautomática foi adotada, em que um motorista dirige o carro manualmente sempre no centro da pista pelas regiões onde ele pretende

utilizar futuramente o veículo autônomo. Os dados da localização do carro são salvos em um log e são utilizados para estimar o centro da pista.

O mapa de custo de pista é construído da seguinte forma:

1. Um mapa é criado e todas suas células são inicializadas com zero, com exceção das células que correspondem ao centro da pista que são atribuídas com um valor máximo, $\max\_value$.
2. Então, o mapa é percorrido duas vezes, na primeira, começando pela primeira célula, indo da esquerda para direita e, na segunda, começando pela última célula indo da direita para a esquerda. A cada iteração do loop, a célula corrente, c_{ij}, é atualizada com a seguinte fórmula:

$$c_{ij} = \max\left(c_{ij}, c_{\max\_neighbor} - map\_resolution\right)$$

onde $c_{\max\_neighbor}$ é o valor do vizinho mais custoso de c_{ij} e $map\_resolution$ é a resolução do mapa em metros. Ao término destes dois loops, $(\max\_value - c_{ij})$ corresponde a menor distância de uma célula qualquer para o centro da pista.

3. O último passo para gerar o mapa de custo é normalizar o mapa para que suas células tenha valores variando entre 0 e 1. Para as células dentro do limite da pista, seu valor é atualizado da seguinte forma:

$$c_{ij} = \left(1 - \frac{c_{ij} - lane\_limit\_value}{max\_value - lane\_limit\_value}\right)0.2$$

onde

$$\text{lane\_limit\_value} = \max\_value - (lane\_width/2 \ * map\_resolution).$$

Caso contrário:

$$c_{ij} = 0.8 + \left(1 - \frac{c_{ij} - \min\_value}{\max\_value - \min\_value}\right)0.2$$

onde $\min\_value$ corresponde ao valor da menor célula do mapa e $lane\_width$ é a largura da pista em metros.

O mapa de custo da pista é interpretado da seguinte forma: quanto mais escura uma célula, mais alto o custo: quanto mais clara, mais baixo o custo. O estado do veículo tem custo máximo na posição mais distante da pista e custo mínimo no centro da pista. O mapa de custo é utilizado para que a escolha de u_{best} considere a centralização do x_{new} em relação à pista. Ele também é adicionado ao custo de x_{new}. Assim, o critério de melhor trajetória passa a considerar informações da pista. A Figura 15 mostra a função NEW_STATE, adaptada para o RC-RRT e considerando informação da estrutura do problema para a escolha de u_{best}.

```
NEW_STATE (x_near, x_rand, x_new, u_best)
{
1.     cost_min = ∞
2.
3.     for all u in U
4.     {
5.           if u has been expanded
6.                 continue;
7.           x' = simulate(x_near, u);
8.           cost = command_cost(x', u, x_rand);
9.           if (cost < cost_min)
10.          {
11.                if (D(x'))
12.                {
13.                      u_best = u;
14.                      x_new = x';
15.                }
16.                else
17.                {
18.                      Mark u as expanded;
19.                      x_near.update_cvf();
20.                }
21.          }
22.    }
23.
24.    if (cost_min != ∞)
25.    {
26.          x_new.cost = x_near.cost + u_best.time + env_cost(x_new);
27.          return true;
28.    }
29.    else
30.    {
31.          x_near.maximize_cvf();
32.          return false;
33.    }
}
```

Figura 15 - Função NEW_STATE modificada

A função *command_cost* (linha 8) é utilizada para avaliar o comando u. No caso de planejamento na rua, um custo é retornado considerando os seguintes critérios:

1. A distância de x' em relação ao x_{rand}, onde x' é um candidato a x_{new};
2. O custo de x' em relação à pista, calculado a partir do mapa de custo da pista;
3. O custo de movimentação: comandos como marcha ré e baixas velocidades são penalizados.

A função *env_cost* (linha 26) retorna um custo relacionado ao ambiente no qual x_{new} está localizado. Esta informação é somada ao custo do vértice. No caso de planejamento de rua, *env_cost* retorna uma penalização de acordo com o mapa de custo da pista.

É importante ressaltar que o uso de custos no planejador de movimento não limita as trajetórias possíveis. Por exemplo, se um carro estiver parado na frente do veículo autônomo e nenhuma trajetória na pista for encontrada, o algoritmo tem a possibilidade de gerar uma trajetória que invade a contramão para ultrapassá-lo.

3.2.1.2 Foco de crescimento do RRT na pista

Para que o crescimento do RRT seja focado na pista e uma rápida convergência seja obtida, o algoritmo BUILD_RRT foi modificado. A Figura 16 mostra essa alteração.

A primeira modificação aparece na linha 2. A função *add_last_traj* adiciona à árvore os vértices da última trajetória que são livres de colisão. Isto proporciona duas principais vantagens:

1. O esforço do último planejamento não é descartado. Caso a trajetória antiga ainda esteja boa, ela será reaproveitada e o foco do RRT será em aperfeiçoá-la ao invés de explorar novas regiões.
2. Os comandos enviados ao veículo passam a ser mais estáveis. Novas trajetórias têm a possibilidade de utilizar trechos da última trajetória, mantendo assim uma continuidade.

Na linha 7, a função RANDOM_STATE_LANE é introduzida. Ao invés de gerar estados aleatórios de forma uniforme, com uma dada probabilidade p_{lane}, é gerado um estado aleatório na pista na região entre x_{init} e x_{goal}. Além disso, na linha 13, foi adicionada a função CONNECT, para fazer com que a árvore cresça agressivamente na direção de x_{goal}, caso permitido pelas restrições de colisão e de movimentação.

```
BUILD_RRT_LANE(x_init)
{
1.    T.init(x_init);
2.    T.add_last_traj();
3.    cost_goal = ∞;
4.
5.    while (time_now < timeout)
6.    {
7.          x_rand = RANDOM_STATE_LANE();
8.          x_near = NEAREST_NEIGHBOR(x_rand, T);
9.
10.         status, x_new = EXTEND(x_near, x_rand);
11.
12.         if (status == ADVANCED)
13.               status, x_new = CONNECT(x_new, x_goal);
14.
15.         if (status == REACHED)
16.         {
17.               cost_goal = min(cost_goal, x_new.cost);
18.               T.discard_umpromissing_vertices();
19.         }
20.   }
}
```

Figura 16 - Função BUILD_RRT_LANE

A condição de saída do loop do BUILD_RRT também foi modificada para considerar tempo, ao invés de número de iterações. Deste modo, RRT continuará tentando encontrar ou melhorar sua trajetória, enquanto tenha tempo disponível. Caso nenhuma trajetória seja encontrada dentro do tempo disponível, uma parada de emergência é executada.

3.2.2 Área livre

Quando o planejador de movimento é utilizado em área livre, o foco é planejar movimentos de forma que o veículo chegue o mais próximo possível do x_{goal},

possivelmente realizando manobras e estacionando. Os seguintes problemas foram identificados para alcançar tal objetivo:

1. O RRT pode demorar a convergir para uma região próxima de x_{goal}. Isto pode acontecer devido a um crescimento totalmente uniforme em que não existe nenhum foco na direção do objetivo, ou devido à complexidade do ambiente, que dificulta que uma solução seja encontrada.
2. O estado desejado, x_{goal}, pode requerer que o veículo execute manobras complexas. Problemas de métrica e na escolha de u_{best} podem fazer com que o algoritmo demore muito para convergir.

Para resolver o primeiro problema, duas modificações foram feitas. A primeira foi utilizar o algoritmo RRT-GoalZoom (Seção 2.5). Deste modo, o foco do RRT é progressivamente direcionado para a região destino, x_{goal}, tornando a convergência mais rápida. Entretanto, mesmo utilizando o RRT-GoalZoom, a convergência do algoritmo pode ser ruim. Por exemplo, um ambiente com presença de muitos obstáculos pode dificultar o crescimento da árvore. Por isso, outra modificação foi feita para que o crescimento do RRT seja guiado para direções promissoras. Para fazer essa condução, um Mapa de Custo de Gradiente foi utilizado. A Figura 17 mostra este mapa de custo aplicado em um ambiente de estacionamento.

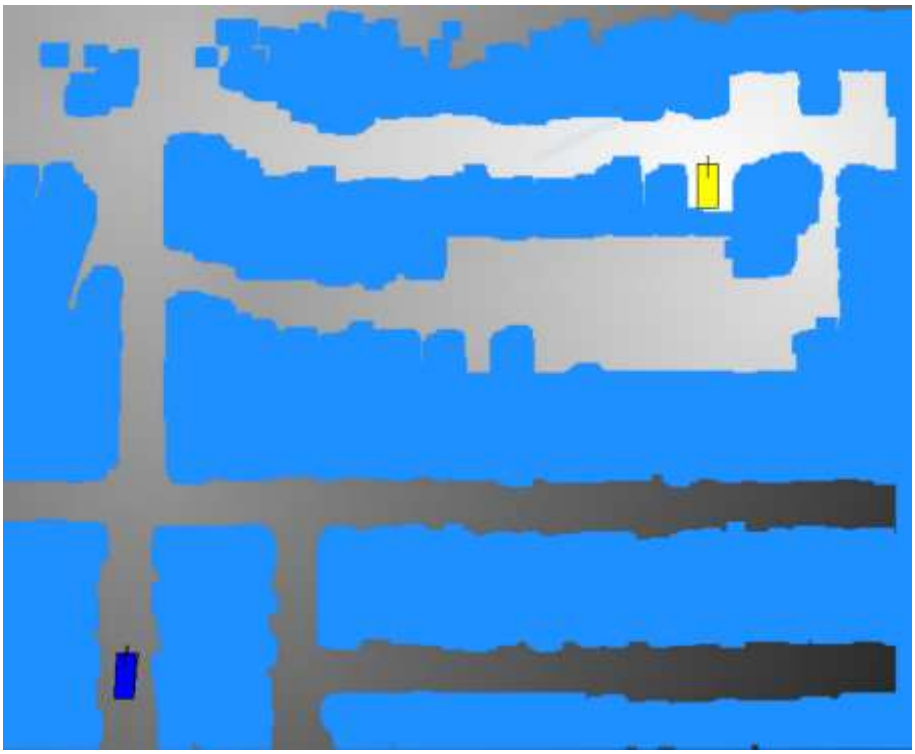


Figura 17 - Mapa de custo de gradiente

O retângulo amarelo representa o estado desejado. Quanto mais próxima uma célula do mapa estiver do destino, mais clara ela será. Para descobrir a melhor trajetória de qualquer célula para x_{goal}, basta seguir o gradiente. Por exemplo, considerando uma célula c_{ij} qualquer, o vizinho com menor custo de c_{ij} é selecionado e o mesmo é feito com seu vizinho recursivamente até chegar à célula que representa x_{goal} (que possui custo igual à zero).

O Mapa de Custo de Gradiente é gerado conforme o *Gradient Field* que faz parte do *Gradient Method* [27], planejador de movimento em tempo real para robôs diferenciais. O *Gradient Field* é construído da seguinte forma. Inicialmente, toda a região de pontos do x_{goal} são atribuídas com 0 e todos os outros pontos são atribuídos com um custo infinito. Os pontos da região do x_{goal} são adicionados em uma lista de pontos ativos. A cada iteração do algoritmo, os pontos da lista ativa são removidos e seus vizinhos são atualizados. Considerando um ponto arbitrário p que acabou de ser atualizado. Este ponto está cercado por 8 vizinhos. Para cada um dos vizinhos, q, o custo de estender p para esse ponto é calculada pela seguinte equação:

$$F(P) = \sum_i I(p) + \sum_i A(p, q)$$

onde, I e A são funções arbitrárias. No caso deste trabalho, I representa uma função de custo de proximidade de obstáculo e A é uma função de distância euclidiana. Se o novo custo para alcançar q é menor que seu custo atual, o custo é substituído e q é adicionado à lista de pontos ativos. O processo se repete até que a lista de pontos ativos se torne vazia. *Gradient Field* representa o caminho ótimo de todos os pontos do mapa até a região destino.

É importante ressaltar que não é possível utilizar a trajetória gerada pelo G*radiente Field* em um veículo convencional, pois, ela considera que o robô pode girar em torno do próprio eixo, o que não é a realidade de um carro. Por isso, o *Gradient Field* foi utilizado apenas como auxilio no crescimento do RRT.

A Figura 18 mostra o algoritmo BUILD_RRT, para o caso de áreas livres. Na linha 7, a função RANDOM_STATE foi modificada para gerar vértices na região do x_{goal} com a probabilidade $p_{goalbias}$. Na linha 11, a função CONNECT_GRAD foi introduzida. Esta função utiliza o G*radiente Field* e faz com que x_{near} seja estendido na direção do gradiente enquanto for possível com a probabilidade p_{grad}; caso contrário, a construção do RRT segue na sua forma original.

```
BUILD_RRT_FREE_AREA(x_init)
{
1.    T.init(x_init);
2.    T.add_last_traj();
3.    cost_goal = ∞;
4.
5.    while (time_now < timeout)
6.    {
7.          x_rand = RANDOM_STATE();
8.          x_near = NEAREST_NEIGHBOR(x_rand, T);
9.
10.         if (random() < p_grad)
11.               status, x_new = CONNECT_GRAD(x_near);
12.         else
13.               status, x_new = EXTEND(x_near, x_rand);
14
15.         if (status == REACHED)
16.         {
17.               cost_goal = min(cost_goal, x_new.cost);
18.               T.discard_umpromissing_vertices();
19.         }
20.   }
}
```

Figura 18 - Função BUILD_RRT_FREE_AREA

Esta mudança faz com que o RRT convirja rapidamente para áreas próximas do destino. Porém, o veículo pode chegar com uma orientação diferente do x_{goal}, e devido a problemas com a métrica e a simplicidade de escolha de u_{best}, uma solução que gere as manobras necessárias pode demorar a ser encontrada.

Para que o RRT convirja mais rapidamente para x_{goal}, foi utilizada cinemática inversa com o método Reeds-Shepp [28]. Este método descreve como um estado inicial chega a outro estado qualquer utilizando um conjunto de 48 possíveis caminhos. Estes caminhos são compostos por no máximo 5 comandos de movimentação. Os possíveis comandos são:

- Frente ou ré reto;
- Frente ou ré virando o máximo para direita;
- Frente ou ré virando o máximo para esquerda.

A função NEW_STATE foi modificada para tentar utilizar o método Reeds-Shepp, caso x_{near} tente atingir x_{goal}. Reeds-Shepp não é utilizado sempre em NEW_STATE, pois as trajetórias geradas por ele exigem que o veículo pare

completamente a cada mudança no ângulo do volante. Como ele é utilizado apenas no final da trajetória, para chegar no x_{goal}, este comportamento não é ruim. As pessoas normalmente precisam parar o carro para manobrá-lo.

3.3 Incertezas

Outro problema típico da robótica são as incertezas. O planejamento de movimento sofre deste problema, pois todas as informações que utiliza possuem incertezas. Alguns exemplos de incertezas sofridas pelo planejador são:

- Sensores possuem erro. Por exemplo, GPS tem erro de alguns metros, a odometria pode informar o ângulo do volante errado, laser tem erro de precisão de alguns centímetros.
- Localização: por utilizar sensores imprecisos a localização também é imprecisa. Deste modo, a posição recebida pela localização pode estar mais perto de um obstáculo que o esperado.
- Controle do veículo: o controle da velocidade e ângulo da roda sofre tanto de imprecisão em nível de software quanto em nível de hardware. Por exemplo, uma trajetória enviada para o veículo pode ser executada de forma um pouco diferente.
- Modelo de Movimentação do Veículo: a modelagem do veículo utilizada pelo planejador de movimento não é perfeita. Um veículo real sofre de problemas difíceis de serem modelados. Por exemplo, terrenos irregulares, pista molhada, pneu não calibrado e velocidade alta tornam a modelagem do carro menos precisa, gerando assim trajetórias que não são perfeitamente seguidas pelo veículo.
- Latência relativa ao tráfego de mensagens na rede. Por exemplo, a mensagem de localização chega com atraso e um comando enviado não é executado instantaneamente.

A solução utilizada foi tratar as incertezas tentando manter as trajetórias longe de obstáculos, dando assim uma margem de segurança, caso pequenos erros aconteçam. Para implementar este comportamento, um mapa de custos de

obstáculo é gerado. A Figura 19 mostra este mapa, quanto mais escura uma posição for, maior o seu custo.

Figura 19 - Mapa de custo de obstáculo

Para que o RRT utilize esse mapa, as funções *command_cost* e *env_cost* utilizadas em NEW_STATE (Figura 15) foram modificadas para considerar o custo de obstáculo. Deste modo, a proximidade de obstáculo é considerada na escolha de comandos e no custo de uma trajetória.

Uma abordagem simples em que o tamanho do robô é aumentado poderia ser utilizada. Entretanto, este tipo de abordagem impossibilita trajetórias em lugares muito estreitos. Já na estratégia utilizada, a trajetória tende a ficar longe de obstáculos. Todavia, caso não exista outra possibilidade, um planejamento próximo de obstáculos ainda é possível.

4 METODOLOGIA

Este capítulo apresenta a metodologia utilizada para avaliar o desempenho do planejador de movimento para veículos convencionais proposto neste trabalho, descrevendo o hardware, software e métricas utilizadas.

4.1 Hardware

Para testar o planejador de movimento desenvolvido neste trabalho foi utilizado o robô IARA (*Intelligent Autonomous Robotic Automobile*) (Figura 20). IARA é a plataforma robótica do LCAD (Laboratório de Computação de Alto Desempenho) construída com base no automóvel de passeio Ford Escape Hybrid.

Figura 20 - Ford Escape Hybrid

Diversas adaptações foram aplicadas no veículo, entre elas a instalação de sensores e mecanismos que permitem controlar o acelerador, freio, posição do volante, etc. por meio de computadores instalados no porta-malas (Figura 21). A tecnologia eletrônica de acionamento dos atuadores (volante, acelerador, freio, entre outros) do automóvel foi desenvolvida pela empresa Torc Robotics [29].

Figura 21 - Recursos computacionais

A Figura 22 mostra os sensores disponíveis na plataforma robótica que incluem câmeras estéreo Bumblebee XB3 da Point Grey [30], *Light Detection And Ranging* (LIDAR) HDL-32E da Velodyne [31], e o *Attitude and Heading Reference System* (AHRS) MTi-G da Xsens [32].

Figura 22 - Sensores utilizados

Para processar todos os dados dos sensores, IARA também conta com seis computadores Dell Precision R5500 (2 Processadores Intel Xeon 2.13 GHZ, 12 GB de memória DDR3 1333MHZ, 2 HDs SSD 120GB em RAID0, 2 Placas de Rede 1GB, Placa de Vídeo Quadro 600, Placa de Vídeo Tesla C2050).

4.2 Software

O planejador de movimento foi implementado na linguagem C++ como um módulo na plataforma CARMEN toolkit.

4.2.1 CARMEN Toolkit

CARMEN toolkit é uma coleção de softwares para controle de robôs. Foi projetada para fornecer uma interface consistente e um conjunto de primitivas básicas para pesquisa em robótica em uma ampla variedade de plataformas de robôs comerciais. O objetivo da plataforma é diminuir a barreira entre implementar novos códigos em robôs simulados e reais, e também facilitar o compartilhamento de algoritmos entre diferentes instituições [33].

CARMEN utiliza uma arquitetura de software modular. Cada habilidade do robô é construída em um módulo separado; por exemplo: módulo de mapeamento, módulo de localização, módulo de planejamento de movimento. A comunicação entre os módulos é feita através do protocolo de comunicação chamado de *Inter Process Communication* (IPC) [34], utilizando principalmente o paradigma de comunicação *Publish-Subscribe*.

O IPC fornece uma forma flexível e eficaz de se trocar mensagens entre processos baseada no *Publish-Subscribe*, que permite enviar e receber estruturas de dados complexas, incluindo listas e vetores de tamanho variável. Além da capacidade de conexão de alto nível entre processos, a biblioteca do IPC oferece também facilidades para comunicação entre computadores diferentes de forma transparente ao usuário através de envio de mensagem utilizando o protocolo TCP/IP.

A Figura 23 mostra o modelo de comunicação *Publish-Subscribe*. Neste modelo, um servidor de aplicativos central independente é conectado a um número de processos específicos (módulos). O servidor central (programa *central* do IPC) é um repositório de informação de todo o sistema responsável pelo roteamento das mensagens. Os módulos (*Subscribers*) indicam o seu interesse em receber certos tipos de mensagens para o servidor central. Outros módulos (*Publishers*) publicam mensagens para o servidor central. Este, por sua vez recebe todas as mensagens publicadas e as envia para os módulos interessados em recebê-las.

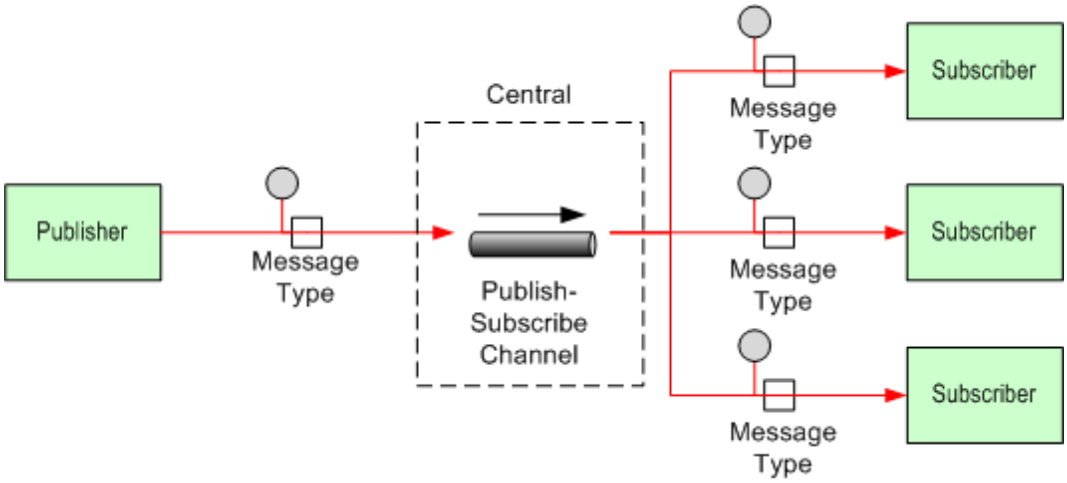


Figura 23 - Modelo de comunicação *Publish-Subscribe* [35]

Nesse modelo, a recepção de mensagens é assíncrona, ou seja, cada assinante (*Subscriber*) possui uma função de *callback*, que é invocada para cada instância do tipo de mensagem recebida. Os módulos publicadores (P*ublishers*) devem se conectar ao processo *central* do IPC, definir suas mensagens, publicá-las, e, eventualmente, ouvir mensagens que assinaram. O uso de modularização e do IPC proporciona vantagens como:

- Escalabilidade: os módulos não precisam ser executados em um mesmo processador. Módulos críticos podem ser executados em um computador separado e a comunicação acontece pela rede via IPC.
- Confiabilidade: a falha de um módulo não faz com que outro módulos parem de funcionar.

4.2.2 LCAD-CARMEN

CARMEN está preparado para controlar vários robôs comerciais com cinemática diferencial, como Nomad XR4000 e Scout e ActivMedia Pioneers. Entretanto, o foco desse trabalho é o desenvolvimento de um planejador de movimento para veículos convencionais. Por isso, CARMEN foi estendida pelo Laboratório de Computação de Alto Desempenho (LCAD) adicionando as funcionalidades necessárias.

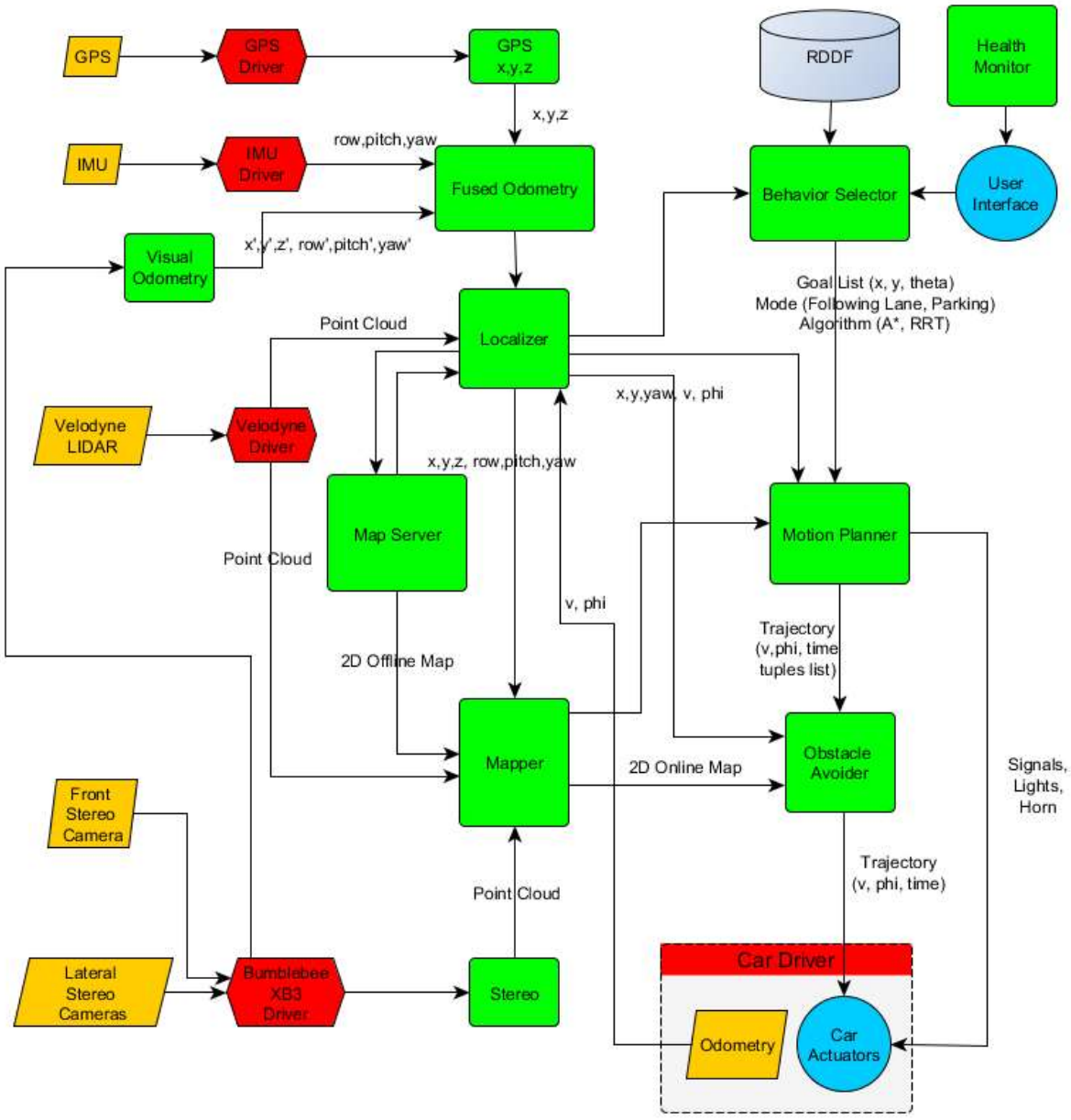


Figura 24 - Módulos LCAD-CARMEN

A Figura 24 mostra os módulos que compõe LCAD-CARMEN. Na figura, os retângulos amarelos representam os sensores que compõe a plataforma IARA. Os retângulos em vermelho são módulos drivers. Estes módulos são responsáveis por estabelecer uma comunicação baixo nível com os sensores, encapsular os dados recebidos em mensagens e publicá-las, tornando-as assim disponíveis para outros módulos. Os retângulos em verde representam módulos de mais alto nível, que recebem mensagens de outros módulos realizam algum tipo de processamento e publicam os dados processados. As setas representam o fluxo das mensagens entre os módulos. Abaixo uma breve descrição dos módulos relevantes para este trabalho:

***Map Server*:** módulo responsável por publicar um mapa previamente feito do ambiente em torno da posição atual do veículo.

***Mapper*:** módulo responsável por construir o mapa do ambiente em torno do veículo autônomo, a partir de dados obtidos do *Velodyne*, localização do veículo e do mapa do *Map Server*.

Localize: módulo responsável por estimar a localização do veículo a partir de dados obtidos do *Velodyne* e do mapa enviado pelo módulo *Map Server.*

***Behavior Selector*:** módulo responsável por manter e publicar a lista de estados destinos que o veículo deve seguir. Além disso, este módulo controla o comportamento do planejador de acordo com o ambiente (rua ou área livre). Por exemplo, se o veículo estiver na rua, o planejador deve ser capaz de manter o veículo na pista correta e evitar colisão com obstáculos móveis e dinâmicos. Neste caso, o *Behavior Selector* publica listas de estados destinos compostas por posições localizadas no centro da pista do veículo. Em áreas livres o planejador deve ser capaz de gerar trajetórias em que o veículo possa realizar manobras de estacionamento, neste caso o *Behavior Selector* publica uma lista com apenas um estado composto pela posição x, y e orientação que o veículo deve alcançar.

***Motion Planner*:** módulo responsável por implementar o planejamento de movimento, foco deste trabalho.

***Obstacle Avoider*:** módulo responsável por receber o planejamento do M*otion Planner* e, caso necessário, interferir reduzindo as velocidades para que o veículo não colida.

***Car Driver*:** módulo responsável pela comunicação de baixo nível com o robô. Recebe uma lista de comandos gerados pelo planejador de movimento e as traduz em esforço de freio, acelerador e mudança no ângulo do volante.

4.3 Métrica

Neste trabalho, foi utilizada como métrica de comparação uma aproximação para a energia gasta em um percurso. A métrica é calculada a partir da equação de energia cinética [36] em função das velocidades do veículo no eixo x e y ao decorrer da execução de um percurso, conforme as fórmulas:

$$v\_x_t = \cos(\theta)\, v_t$$

$$v\_y_t = \text{sen}(\theta)\, v_t$$

$$E_x = \sum \frac{m(v\_x_t{}^2 - v\_x_{t-1}^2)}{2}$$

$$E_y = \sum \frac{m(v\_y_t{}^2 - v\_y_{t-1}^2)}{2}$$

$$E = E_x + E_y$$

onde v_t é a velocidade do veículo no tempo t, v_{t-1} é a velocidade do veículo no tempo $t-1$ e m é a massa do veículo. Trajetórias que oscilam muito o ângulo do volante e a velocidade tendem a resultar em valores altos de energia, já trajetória em que o ângulo do volante e velocidade mudam de forma mais suaves tendem a gerar valores menores de energia.

A métrica utilizada não considera diretamente a trajetória gerada pelo planejador de movimento. Ao invés disto, a execução da trajetória no carro é avaliada a partir de

seu estado estimado pelo modulo de localização. Por isso, se um percurso em comum for utilizado, é possível utilizar a métrica proposta para realizar comparações entre diferentes planejadores de movimento e até mesmo com motoristas. Porém, por considerar apenas a execução das trajetórias a métrica está sujeito a problemas relacionados ao controle do veículo e a incertezas da localização.

5 EXPERIMENTOS E RESULTADOS

Este capítulo apresenta os experimentos e resultados do planejador de movimento desenvolvido neste trabalho.

5.1 Experimento

Para possibilitar a comparação do algoritmo desenvolvido neste trabalho com outros planejadores de movimento ou mesmo com motoristas, uma métrica de comparação foi proposta na Seção 4.3. Esta métrica tem o objetivo de avaliar a eficiência das trajetórias executadas em termos de energia gasta.

Para que o experimento fosse mais interessante, dois motoristas com perfis diferentes foram convidados para participar. Eles foram orientados a dirigir o veículo com a mesma velocidade máxima que o planejador de movimento estava sujeito. Os motoristas tinham o seguinte perfil:

- Motorista 1: motorista experiente e acostumado a dirigir o Ford Escape Hybrid.
- Motorista 2: motorista recém habilitado e com pouca experiência. Além disso, não está acostumado com o Ford Escape Hybrid.

O experimento funcionou da seguinte forma:

1. Um percurso foi definido.
2. O veículo foi controlado pelo planejador de movimento ou pelo motorista no percurso.
3. Durante a execução do trajeto, um *log* foi gravado contendo informações da localização, odometria e comando enviado ao veículo.
4. Os *logs* foram processados gerando gráficos para realizar comparações.

5.2 Local do Percurso

A Figura 25 mostra o local onde os experimentos foram realizados, um estacionamento da UFES.

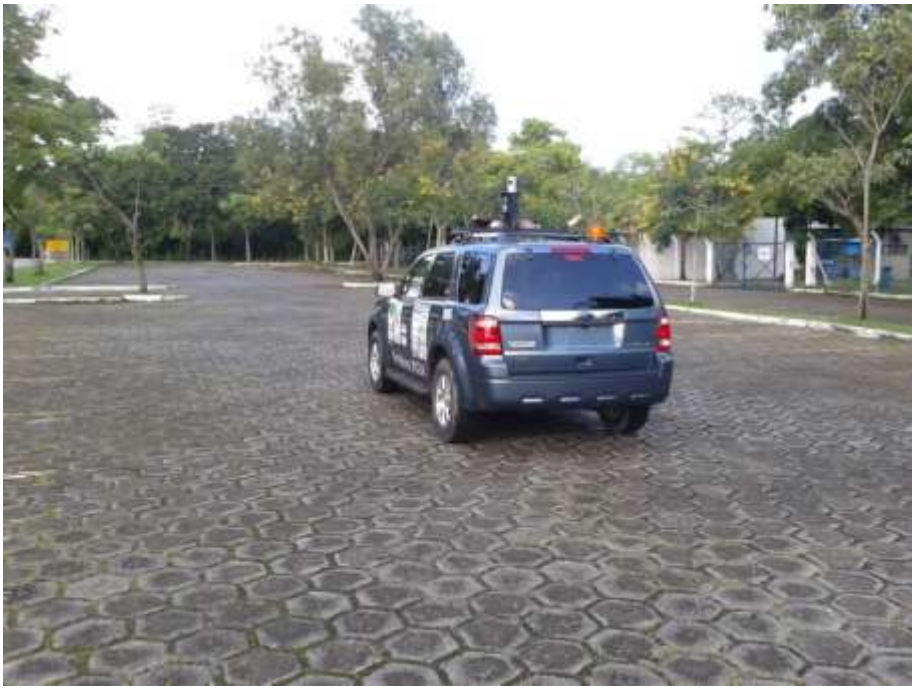


Figura 25 - Demonstração da IARA na UFES

A Figura 26 mostra o mapa deste estacionamento. À esquerda, o mapa é representado como imagem de satélite e, à direita, na forma de *grid* de ocupação. Este local foi escolhido porque possui espaço suficiente para a realização de bons percursos e possui trânsito reduzido de veículos.

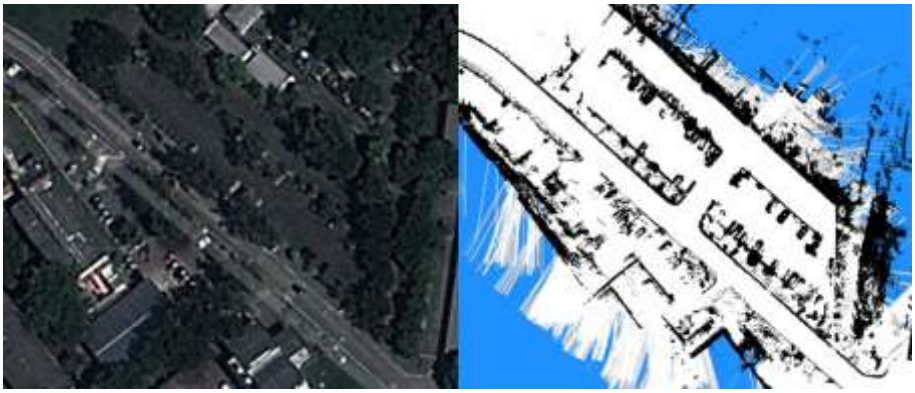


Figura 26 - Mapa do estacionamento dos experimentos

5.3 Parâmetros

Os parâmetros específicos do veículo e do planejador utilizados no experimento são especificados na tabela abaixo.

Parâmetro	Valor
Velocidade Máxima	2,5 m/s
Ângulo da Roda Dianteira Máximo	26,356 graus
Aceleração Máxima	2,7 m/s^2
Desaceleração Máxima	1 m/s^2
Taxa Máxima de Curvatura da Roda Dianteira	27,699 graus/s
Tempo Mínimo de Planejamento	0,08 s
Tempo Máximo de Planejamento (*Timeout*)	0,8 s
Distância Máxima entre Vértices	3,5 m
Distância ente Estados Destino	15,0 m
Massa do Veículo	1500 kg

Tabela 1 - Parâmetros do planejador de movimento

No experimento, o planejador de movimento pôde gerar trajetórias a 12,5 hertz. A frequência em que as trajetórias são geradas é controlada pelo parâmetro Tempo Mínimo de Planejamento. Frequências muito altas podem gerar rápidas variações nas trajetórias enviadas para o veículo. Além disso, pode piorar a qualidade das trajetórias, na ocorrência de falta de tempo para melhorar a solução encontrada. Caso o planejador não encontre uma trajetória dentro do intervalo de tempo do parâmetro Tempo Máximo de Planejamento, o veículo executa uma parada de emergência.

O parâmetro Distância Máxima entre Vértices determina o tamanho máximo do passo para crescimento da árvore. O passo é a distância máxima entre o vértice mais próximo ao aleatório e o novo vértice inserido na árvore. O passo pode ser menor que a distância máxima em dois cenários: quando o comando escolhido tenha velocidade zero (o novo estado pode parar antes de percorrer a distância máxima) e quando o estado aleatório estiver mais próximo que a distância máxima.

Aumentar o tamanho máximo do passo faz com que o planejador gere trajetórias com poucos comandos que são aplicados por um longo período, e diminuir o tamanho máximo do passo provoca o efeito contrário (trajetórias com muitos comandos que são aplicados por um curto período). O tamanho máximo do passo também tem efeito no desempenho do algoritmo. Se os passos forem grandes, menos estados serão explorados. Entretanto, uma solução pode deixar de ser considerada porque uma área com potencial de gerar solução pode deixar de ser explorada. Já se os passos forem pequenos, a árvore terá potencial de explorar um maior número de estados. Em consequência, o tempo de processamento do algoritmo crescerá, aumentando assim a chance de *timeout*.

O parâmetro Distância entre Estados Destinos controla a distância entre os estados presentes na lista de estados destinos publicada pelo módulo Behavior_Selector (Seção 4.2.2). O planejador de movimento recebe a lista de estados destinos e gera trajetórias que leva o robô até o primeiro ponto da lista. A cada atualização na localização do veículo autônomo uma nova lista de estados destinos é gerada. A variação do parâmetro Distância entre Estados Destinos pode afetar o desempenho e o comportamento do planejador. Utilizar o parâmetro com um valor muito alto faz com que o planejador de movimento gere trajetórias maiores, aumentando assim o tempo médio gasto no planejamento. Caso exista um obstáculo móvel como, por exemplo, um carro entre o veiculo autônomo e o estado destino, uma trajetória desviando desse carro será gerada. Porém, se o carro estiver na mesma velocidade que o veiculo autônomo, a trajetória com desvio não será executada, podendo causar apenas oscilações do carro na pista o que é um comportamento indesejável. Caso o parâmetro Distância entre Estados Destinos seja usado com valores pequenos, trajetórias menores são geradas. Em consequência, o planejamento torna-se mais rápido já que um espaço menor será explorado. Caso exista um obstáculo estático ou móvel com velocidade menor que a do veiculo autônomo depois do estado destino, uma trajetória para desviar do obstáculo pode ser gerada tardiamente causando comportamentos em que o veiculo decide desviar do obstáculo muito em cima da hora ou, no pior caso, o veículo pode ficar sem tempo para executar uma manobra de evasão.

5.4 Resultados

A Figura 27 mostra o planejador de movimento em ação no percurso do experimento.

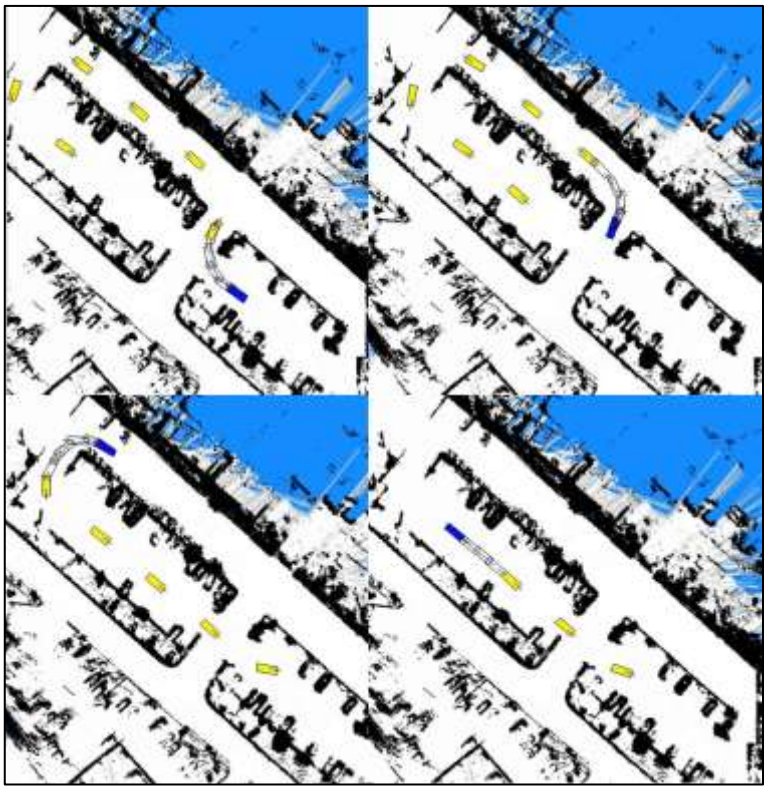


Figura 27 - Planejamento no percurso do experimento

O retângulo em azul representa a posição simulada do veículo; os retângulos em amarelo representam o percurso pré-programado, ou seja, as coordenadas por onde o veículo deve percorrer; e os retângulos sem preenchimento representam a trajetória resultante do planejamento de movimento, ou seja, os pontos por onde o veículo planeja passar. O percurso pré-programado é o percurso ideal, gerado a partir de um *log* com um motorista dirigindo no percurso desejado. Ao executar com o veículo no modo autônomo, o módulo Behavior_Selector (Seção 4.2.2) realiza processamentos no *log* e gera listas de objetivos de forma que o veículo trafegue no percurso pré-programado.

A figura apresenta quatro trajetórias em diferentes situações: três trajetórias de curva e uma de reta. É possível notar que as trajetórias de curva mantém certa distância de segurança dos obstáculos. Isto é alcançado utilizando o mapa de custo

de obstáculo. Caso este mapa não fosse utilizado, a trajetória ótima seria puramente a de menor distância, o que faria com que o veículo andasse bastante próximo dos obstáculos nas curvas. Neste caso, qualquer imprecisão no sensoriamento, no controle ou na modelagem do veículo poderia provocar colisão.

A Figura 28 apresenta um gráfico com os percursos executados no experimento. Este gráfico foi construído a partir da plotagem das localizações do veículo durante as execuções do percurso. A legenda deste gráfico e dos próximos que serão apresentados nesta seção está organizada da seguinte forma:

- Motorista 1: refere-se às localizações do veículo real controlado pelo motorista experiente durante a execução do percurso .
- Motorista 2: refere-se às localizações do veículo real controlado pelo motorista inexperiente ao longo do percurso .
- RRT: refere-se às localizações do veículo real controlado pelo planejador de movimento desenvolvido neste trabalho durante a execução do percurso.
- Simulação: refere-se às localizações do veículo simulado controlado pelo planejador de movimento desenvolvido neste trabalho durante a execução do percurso.

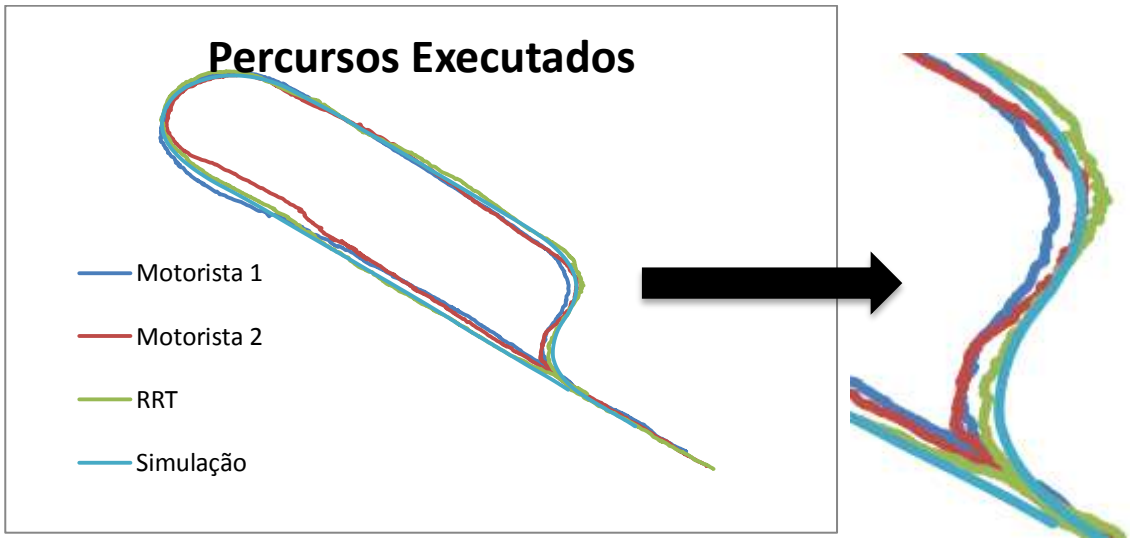


Figura 28 - Percursos executados

No gráfico da Figura 28, é possível notar que os caminhos percorridos pelo planejador e pelos motoristas foram similares. Entretanto, algumas diferenças podem ser notadas. O motorista 1 (experiente) fez a primeira curva em S bastante

fechada. Provavelmente fez isso porque tem plena confiança em suas habilidades e sabe que uma curva fechada tem a vantagem de percorrer uma menor distância. Já o motorista 2 (inexperiente) fez esta curva mais aberta, comportamento similar ao percurso executado pelo veículo autonomamente. O planejador de movimento e o motorista 2 preferem ser mais conservadores ao realizar a curva por motivos similares: o motorista porque sabiamente não confia totalmente em suas habilidades e também não tem certeza de como o carro reagirá aos seus comandos, já que não está acostumado a dirigi-lo; e o planejador de movimento porque não confia totalmente no seu conhecimento de como o carro se movimentará ao receber um comando (modelo de movimentação implementado é uma aproximação) e nem na localização e no controle do veículo devido às incertezas. Assim que o motorista 2 ganhar mais segurança em sua habilidade e se adaptar ao carro, ele terá mais confiança em fazer uma curva fechada ou outras manobras. O mesmo vale para o veículo autônomo: à medida em que as incertezas diminuírem (melhorias na localização, controle e modelagem do veículo), o planejador de movimento poderá confiar mais em suas habilidades e adotar custos menores para trajetórias próximas de obstáculos.

Após o término da curva em U, o motorista 2 se desviou um pouco dos outros caminhos, provavelmente porque demorou a desvirar o volante depois de fazer a curva. Já o planejador não teve dificuldades e se manteve próximo da trajetória gerada pelo motorista 1.

É importante também comparar a trajetória gerada pelos motoristas com aquela gerada pelo planejador de movimento durante a simulação. O veículo simulado está sujeito a menos incertezas do que o veículo real. Com isso, a localização é muito mais precisa, gerando pouca oscilação na estimativa da posição do veículo. Além disso, a modelagem de movimento implementada na simulação é próxima daquela implementada no planejador de movimento, havendo assim pouca divergência entre a trajetória planejada e a executada. A soma desses fatores faz com que o caminho gerado na simulação seja tão bom ou melhor do que aquele gerado pelo motorista 1.

A Figura 29 mostra o gráfico da quantidade de energia consumida pelos motoristas e pelo planejador de movimento tanto durante a execução real (ao controlar o veículo real) quanto durante a simulação (ao controlar o veículo simulado).

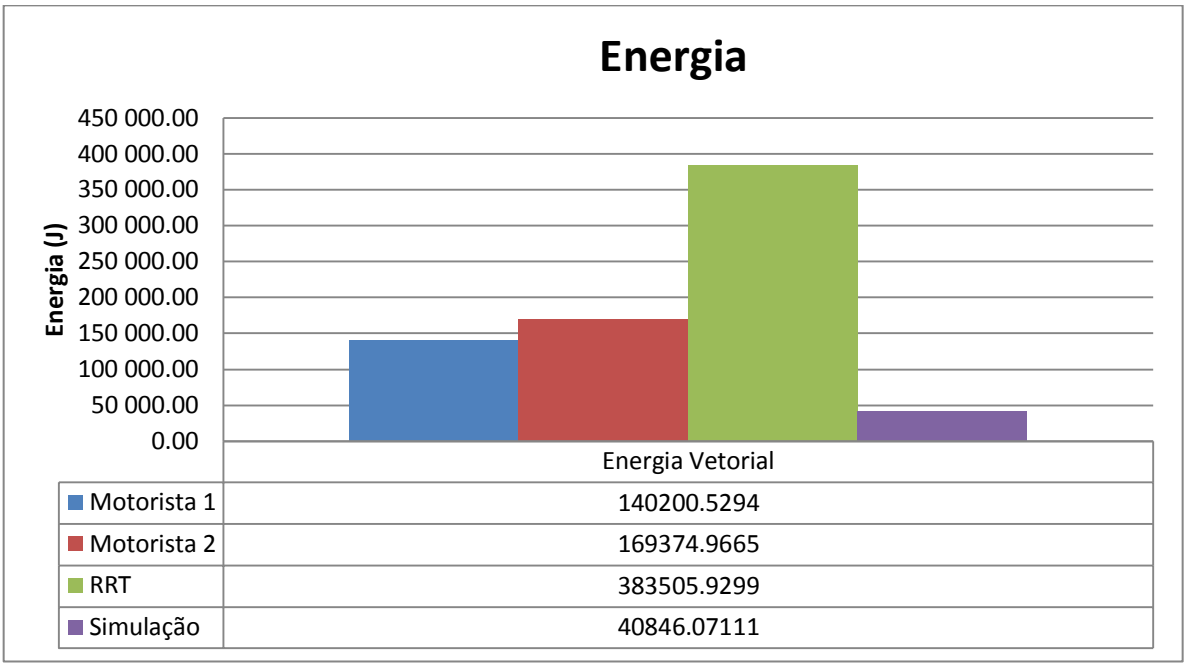


Figura 29 - Gráfico da quantidade de energia consumida

No gráfico da Figura 29, é possível notar que houve pouca diferença na quantidade de energia consumida entre os motoristas. A energia gasta pelo planejador de movimento durante a simulação foi muito menor do que a gasta pelos motoristas e pelo planejador de movimento durante a execução real. Já se a comparação for feita apenas entre os motoristas e o planejador de movimento durante a execução real, os motoristas ganham com vantagem.

A energia consumida está fortemente relacionada com a velocidade do veículo durante o percurso. Por este motivo, gráficos da velocidade do veículo em função do tempo de execução do percurso serão utilizados para explicar melhor os resultados.

A Figura 30 mostra a velocidade do veículo em função do tempo de execução do percurso enquanto o veículo simulado foi conduzido pelo planejador de movimento.

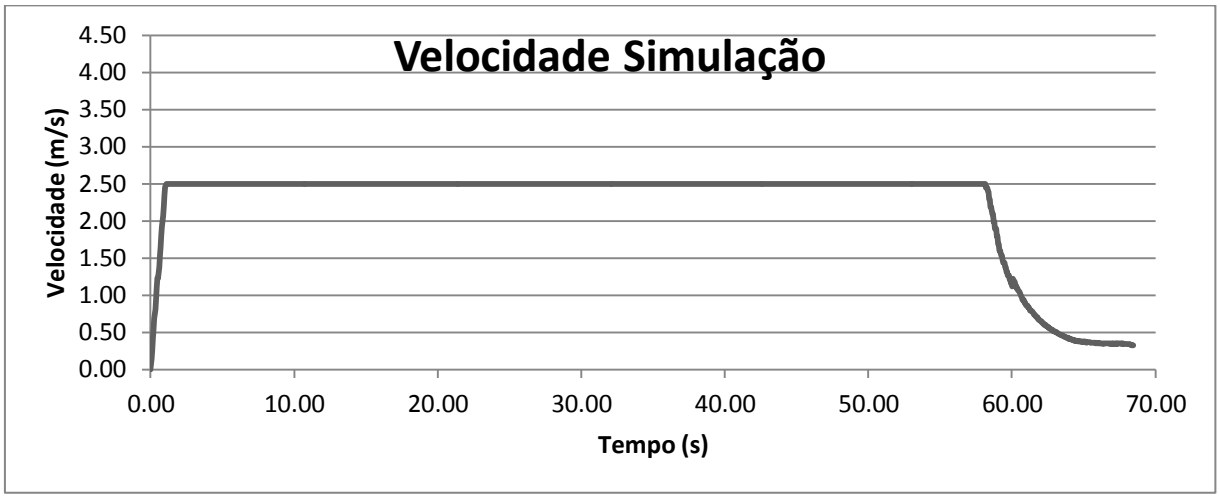


Figura 30 - Gráfico de velocidade na simulação

Como discutido anteriormente, a simulação é o caso mais próximo da perfeição, por sofrer poucas incertezas e por possuir uma modelagem de movimento do veículo muito próximo à modelagem do planejador. Pelo gráfico, é possível notar que o planejador acelera rapidamente o veículo até a velocidade máxima e depois sai dessa velocidade apenas quando uma parada suave é realizada no final do percurso. Assim que a velocidade máxima foi atingida, ela não foi reduzida nem mesmo nas curvas. O experimento foi executado com velocidade relativamente baixa e o planejador julgou o carro capaz de realizar a curva sem nenhuma redução, deste modo houve pouca variação dos comandos enviados para o carro. Como a métrica utilizada penaliza variações nos comandos, a energia gasta na simulação foi mínima.

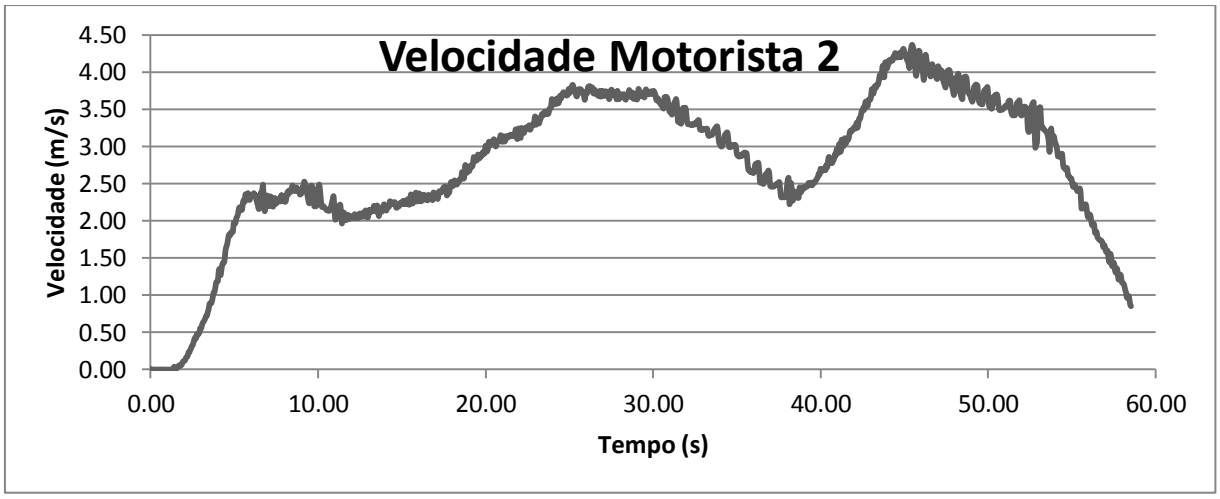


Figura 31 - Gráfico velocidade motorista 2

A Figura 31 mostra a velocidade do veículo em função do tempo de execução do percurso enquanto o veículo foi dirigido pelo motorista 2.

Pela aparência do gráfico da Figura 31, é possível imaginar que o percurso foi bastante desagradável, pois o veículo oscilou constantemente sua velocidade. Entretanto, o motorista manteve a velocidade suave, apenas diminuindo um pouco nas curvas e acelerando nas retas. As oscilações observadas no gráfico são causadas pela imprecisão da odometria. Esta imprecisão afetou negativamente a energia gasta pelos motoristas. Portanto, o que gerou uma discrepância tão alta entre a energia gasta na simulação e a energia gasta pelos motoristas foi o comportamento humano de reduzir nas curvas e acelerar nas retas, em conjunto com as incertezas relativas à medição da odometria do veículo.

A Figura 32 mostra a velocidade do veículo em função do tempo de execução do percurso enquanto o veículo foi conduzido pelo planejador de movimento durante experimentação no veículo real.

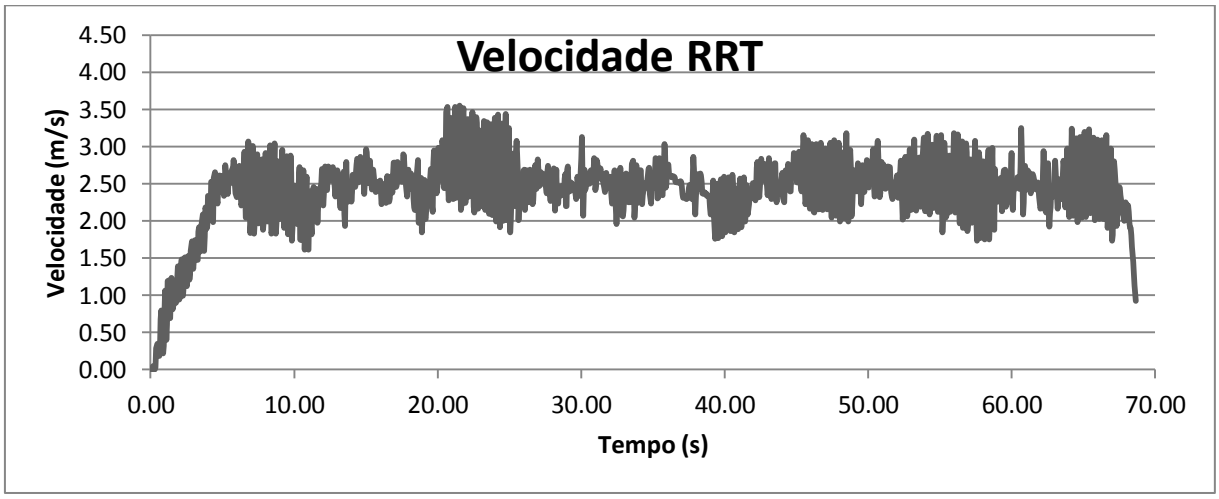


Figura 32 - Gráfico de velocidade RRT

No gráfico da Figura 32, é possível notar que a velocidade oscilou muito durante o percurso. Isto aconteceu por causa da imprecisão da odometria e do módulo responsável pelo controle do veículo (módulo base descrito na Seção 4.2.2, que está fora do escopo deste trabalho). Devido à imprecisão do módulo de controle do veículo, as trajetórias enviadas pelo planejador não foram seguidas com perfeição.

Se as oscilações na velocidade forem desconsideradas, é possível notar que o gráfico da Figura 32 (velocidade durante a execução real) é bastante parecido com o gráfico da Figura 30 (velocidade durante a simulação). O planejador tenta acelerar o veículo para a velocidade máxima permitida, 2.5 m/s, mantém a velocidade durante todo o percurso e ao final a reduz suavemente.

A soma da imprecisão da odometria com a do módulo responsável pelo controle do veículo fez com que os percursos conduzidos pelo planejador de movimento no veículo real apresentassem o pior gasto de energia. Para realizar uma comparação mais justa, a velocidade dos *logs* foi filtrada para remover oscilações provenientes de incertezas da odometria e do controle do veículo. Os gráficos da Figura 33 e da Figura 34 mostram as velocidades dos gráficos da Figura 31 e da Figura 32 após utilizar o filtro passa baixo *Buttherworth* [37] com os parâmetros frequência de corte 0,05 hz e frequência de amostragem 5 hz. O comportamento da velocidade nos gráficos após a filtragem permaneceu o mesmo, mas a maior parte das oscilações foi removida.

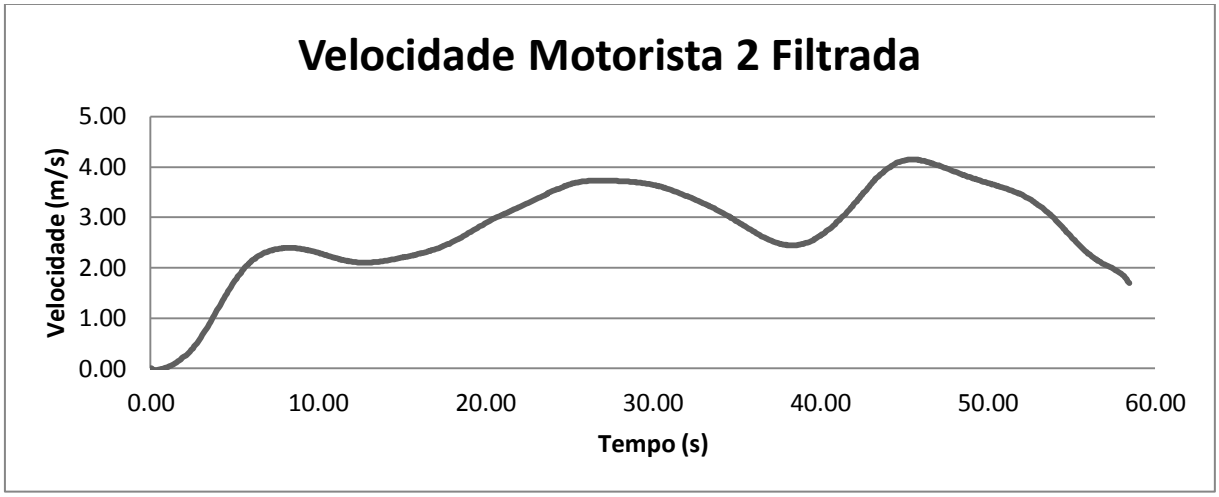


Figura 33 - Gráfico velocidade filtrada motorista 2

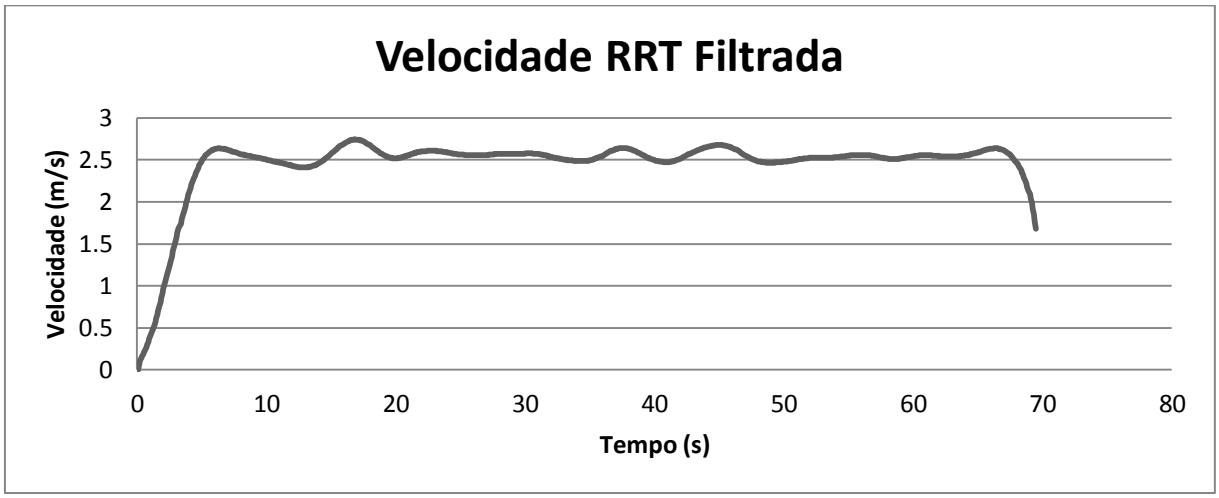


Figura 34 - Gráfico de velocidade filtrada RRT

A Figura 35 mostra a energia consumida caso a odometria do veículo tivesse retornado as velocidades filtradas durante a execução dos percursos. O planejador na simulação continua apresentado o melhor resultado. Entretanto, desta vez, os percursos executados pelo planejador de movimento no veículo real passaram a gastar um valor próximo do motorista experiente. Isso mostra que, caso a imprecisão do módulo de controle fosse resolvida, o planejador de movimento desenvolvido neste trabalho teria potencial de conduzir veículos de forma a gastar energia compatível com aquela gasta por motoristas.

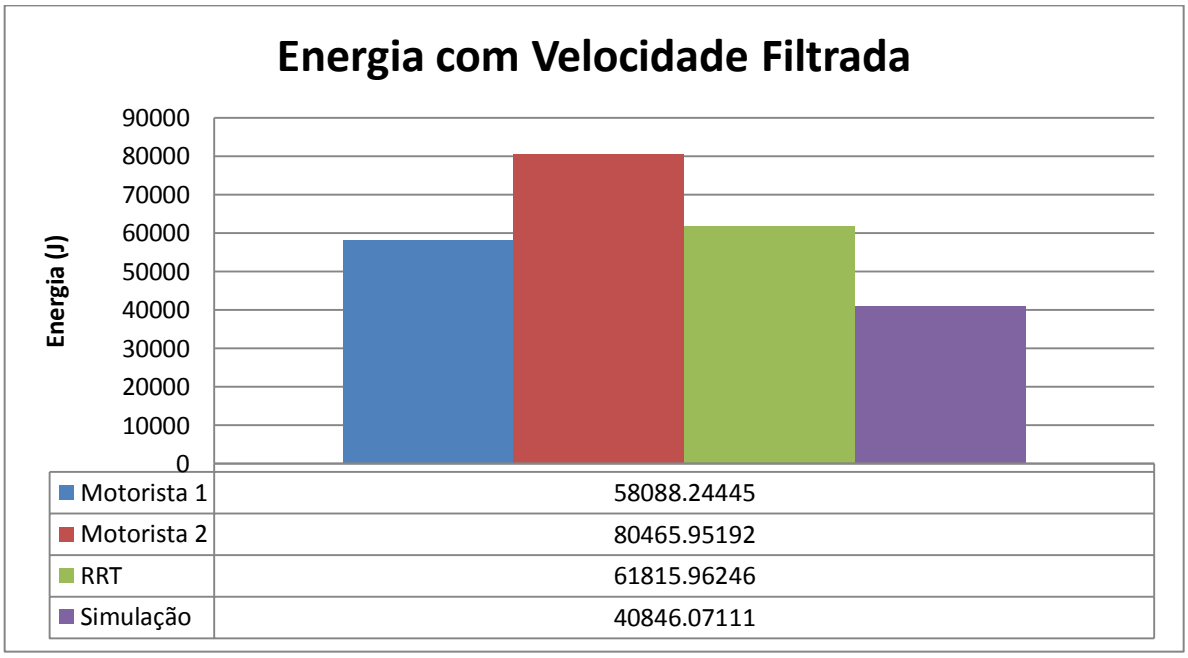


Figura 35 - Energia com velocidade filtrada

A Figura 36 mostra os resultados do planejador de movimento quando executado no modo áreas livres.

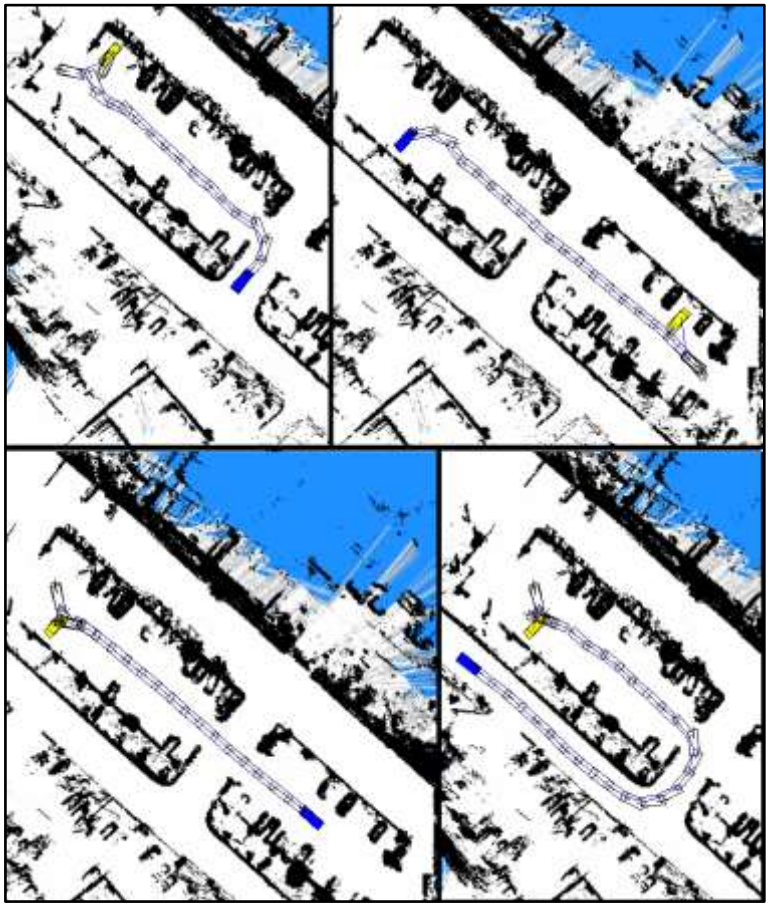


Figura 36 - Planejamento em área livre

Neste modo, o planejador é executado sem restrições relativas ao posicionamento do veículo na pista. Além disso, não há penalização de comandos de ré e uma solução é encontrada apenas quando um novo estado suficientemente próximo do estado destino for adicionado à árvore. No caso do planejamento na pista, uma solução é encontrada ao adicionar um novo estado com distância menor que 2 metros do estado destino.

Caso o estado destino seja posicionado em vagas de estacionamento, o planejador é capaz de gerar as manobras necessárias para que o veículo estacione. O planejador de movimento no modo área livre utiliza o mapa de custo de gradiente para focar o crescimento da árvore nas direções com maior potencial de alcançar o estado de destino. Assim que vértices próximos ao estado destino forem

adicionados, o método Reeds-Shepp é utilizado para tentar encontrar os comandos que os levam para o estado destino.

Na Figura 36, quatro exemplos de trajetórias são apresentados, nos quais o estado destino foi posicionado de forma que as trajetórias encontradas estacione o veículo. Nas trajetórias ilustradas, não há muitas mudanças no ângulo do volante e no sentido da movimentação (frente ou ré) enquanto os estados intermediários das trajetórias estão longe do estado destino. No entanto, no final das trajetórias é possível notar mudanças bruscas no ângulo do volante e no sentido da movimentação do veículo para estacioná-lo, que são características de comandos gerados utilizando Reeds-Shepp.

O veículo autônomo, IARA, foi apresentado à imprensa em diversas ocasiões na UFES [38] [39] [40] [41] [42] e uma vez no programa Mais Você no Projac (Projeto Jacarepaguá, como é conhecida a Central Globo de Produção) [43]. Nestas demonstrações, o veículo percorreu autonomamente percursos pré-programados utilizando o planejador de movimento desenvolvido neste trabalho e apresentou bom desempenho. Vídeos com as demonstrações podem ser acessados em [38] [39] [40] [41] [42].

6 DISCUSSÃO

Este capítulo discute o trabalho desenvolvido, apresentando os trabalhos correlatos e análise crítica do trabalho.

6.1 Trabalhos Correlatos

O problema de planejamento de movimento pode ser resolvido de diversas formas. Esta seção apresentará trabalhos que resolvem este problema para carros utilizando diferentes abordagens.

O trabalho [44] apresenta o planejador de movimento que foi utilizado pela equipe de Standford na Darpa Urban Challenge alcançando o segundo lugar. O planejador foi utilizado principalmente em áreas livres em tarefas como navegar em um estacionamento, executar curvas em U e lidar com cruzamentos bloqueados. O planejador de movimento foi baseado no algoritmo A* com estados híbridos compostos por posições x, y, orientação e direção do movimento (frente ou ré). Diferentemente do planejador desenvolvido nesta dissertação, velocidade e ângulo de volante não são consideradas em nível de planejamento. Ao invés disto, depois que a trajetória é encontrada, um pós-processamento ainda é aplicado para otimizar a solução e encontrar os comandos de velocidade e ângulo de volante.

A dissertação [45] avalia o algoritmo apresentado no trabalho anterior [44] na plataforma robótica IARA, a mesma utilizada para validar o planejador desta dissertação.

O trabalho [46] apresenta o planejador de movimento da equipe Tartan Racing que venceu o Darpa Urban Challenge. O sistema é composto por dois planejadores que são usados em dois cenários: estruturado (seguir a pista) e não estruturado (manobras em estacionamentos). Um modelo de movimento preciso do veículo foi utilizado nos planejadores de movimento. Este modelo considera limites de curvatura, taxa máxima de curvatura, aceleração e desaceleração máxima, e um modelo da latência dos comandos. Quando o veículo está na pista (ambiente estruturado), as trajetórias são geradas a partir de um planejador simples que

mantem o carro no centro da pista enquanto faz progressos em direção ao destino. Quando o veículo está em regiões não estruturadas, trajetórias computadas off-line (sem considerar obstáculos) a partir do modelo de movimento do veículo são utilizadas como solução inicial do algoritmo Anytime D*. Este algoritmo realiza buscas em um espaço de quatro dimensões (posição, orientação, direção do carro) gerando rapidamente soluções iniciais sub-ótimas que são posteriormente melhoradas com o decorrer do tempo. Assim como a solução do trabalho [46], o trabalho apresentado nesta dissertação também usa um modelo de movimento parecido, porém não considera latência. A estratégia utilizada quando o carro está em regiões estruturadas também é similar a usada nessa dissertação.

O trabalho [47] modifica o RRT para gerar trajetórias seguras para um robô *indoor* com restrição Ackerman. Isso é alcançado com uma versão do RRT que trata explicitamente as incertezas da localização do robô. Os estados da árvore passam a representar incerteza mantendo sua posição estimada e a covariância associada. Além disto, cada novo estado é gerado a partir de um algoritmo de localização baseado no Extended Kalman Filter (EKF). No planejador de movimento desenvolvido nesta dissertação, as incertezas são tratadas de forma implícita por meio do mapa de custo de obstáculos. A abordagem utilizada tem a vantagem de ser computacionalmente mais eficiente.

O trabalho [24] propõe o algoritmo Reachability-Guided RRT (RG-RRT). Este algoritmo apresenta uma solução para tornar a seleção de vértices menos sensível a métrica. Isto é atingido armazenando uma lista de estados possíveis de serem alcançados para cada vértice. Esta lista então é utilizada na escolha do vértice para a expansão. O vértice selecionado é aquele que tem em sua lista o estado mais próximo do estado aleatório (dentre todos os estados possíveis de serem alcançados a partir de todos os vértices da árvore). Isto aumenta a probabilidade que o vértice escolhido gere estados na direção do estado aleatório. Entretanto, esta solução torna a seleção de vértices mais pesada. Por isso não é utilizada no planejador de movimento desta dissertação.

Outra solução para tornar a seleção de vértices menos sensível à métrica é o Resolution Complete RRT (RC-RRT) [25]. No algoritmo, uma estrutura de dados associada a cada vértice armazena uma lista de comandos que já foram aplicados.

O algoritmo gera a informação de quão promissor um vértice é com base em seus comandos aplicados que resultaram em expansões mal sucedidas. A seleção de vértices para expansão passa a priorizar os vértices considerados mais promissores. A lista de comandos é usada também para evitar que comandos repetidos sejam aplicados a um mesmo vértice. Além disso, estados muito parecidos são impedidos de serem inseridos na árvore. Assim, evita-se a exploração de estados repetidos.

O trabalho [23] apresenta o planejador de movimento baseado em RRT utilizado pela equipe do MIT na competição Darpa Urban Challange em que alcançaram o quarto lugar. O planejador é capaz de fazer com que o veículo obedeça às regras de trânsito, negocie entrada em rotatória, ultrapasse outros veículos, etc. Para verificar se uma trajetória colide ou viola alguma regra de trânsito, o planejador utiliza um mapa de dirigibilidade. Este mapa armazena todos os dados obtidos pelo veículo por meio de sensores, como obstáculos estáticos e dinâmicos, proximidade de obstáculos, limites da pista, etc. O planejador de movimento desenvolvido nesta dissertação utiliza mapa de custo da pista, mapa de custo de obstáculo e mapa de ocupação de *grid* em conjunto para atingir objetivo similar ao mapa de dirigibilidade.

6.2 Análise Crítica

Algumas limitações foram identificadas nesse trabalho. Primeiro, o planejador de movimento desenvolvido neste trabalho não considera explicitamente obstáculos móveis. Porém, planejamentos na presença de tais obstáculos ainda são possíveis. Os obstáculos móveis são mapeados como obstáculos estáticos em um mapa de alta frequência. Em conjunto com planejamentos frequentes é possível evitá-los.

Segundo, em modo autônomo, a direção do veículo não é tão suave quanto à direção humana. Problemas no módulo de controle do veículo podem estar contribuindo negativamente para este quesito.

Terceiro, mais detalhes do ambiente ao redor do veículo deveriam ser mapeados e considerados pelo planejador de movimento. Por exemplo, mão, contra mão, quebra mola, sinais de trânsito, etc. Atualmente o planejador sabe apenas onde está o centro da pista e consegue manter o veículo na sua mão.

Finalmente, trajetórias com manobras estão muito sensíveis aos problemas do controle do veículo, tornando-se difícil a execução delas no veículo real.

7 CONCLUSÃO E TRABALHOS FUTUROS

Este capítulo apresenta as conclusões e direções para trabalhos futuros.

7.1 Conclusão

O objetivo deste trabalho foi desenvolver um planejador de movimento para veículos convencionais. Esses veículos estão sujeitos a restrições *kinodynamic*, ou seja, restrições cinemáticas e dinâmicas.

O algoritmo RRT foi utilizado como base do planejador de movimento, porque possui uma boa estrutura para lidar com as restrições do veículo e, além disso, funciona bem em problemas com muitas dimensões.

Diversas modificações foram feitas para melhorar a eficiência e o comportamento do planejador de movimento. Primeiro, foi adicionada ao algoritmo a capacidade de lidar com incertezas. Para isso, trajetórias próximas a obstáculos têm seu custo penalizado. Deste modo, trajetórias seguras (que mantém distância de segurança de obstáculos) são priorizadas. Segundo, trajetórias do último planejamento são reaproveitadas em novos planejamentos, melhorando assim a eficiência do algoritmo e causando menos oscilações nos comandos enviados ao veículo. Terceiro, sempre que uma trajetória é encontrada, podas são feitas na árvore removendo estados que não têm chance de gerar trajetórias melhores. Quarto, ao planejar na pista, trajetórias com manobras que se afastam do centro dela são penalizadas e estados aleatórios na região da pista são priorizados. Finalmente, ao planejar em áreas livres, o *gradient field* foi introduzido para guiar mais rapidamente o crescimento da árvore para regiões próximas do estado destino. Modificações também foram feitas para encontrar manobras precisas que levem de um estado qualquer para o estado destino utilizando Reed-Sheep, facilitando assim planejamento para estacionamento.

O planejador de movimento foi implementado na plataforma CARMEN e testado em ambiente de simulação e no veículo autônomo IARA. Diversas apresentações com a IARA foram feitas para a mídia. O algoritmo desenvolvido neste trabalho foi utilizado

em todas elas, demonstrando bom funcionamento. Experimentos foram realizados comparando a energia gasta pelo planejador de movimento com a energia gasta por motoristas. Os resultados mostraram que o algoritmo tem potencial de conduzir um veículo de forma a gastar uma quantidade de energia equivalente àquela gasta por um motorista.

7.2 Trabalhos Futuros

Uma direção para trabalho futuro seria aperfeiçoar os outros módulos que compõe o veículo autônomo, como, por exemplo, o módulo de localização, o módulo de mapeamento e o módulo de controle. O planejador de movimento está sujeito a problemas referentes a estes módulos. Por isso, melhorias em qualquer um deles causariam efeitos positivos no planejador de movimento.

Outras direções para trabalhos futuros seriam: considerar obstáculos móveis em nível de planejamento; suavizar o volante para se parecer mais com um humano; criar mecanismos para que o planejamento de movimento obedeça a regras de trânsito; e melhorar a execução de trajetórias compostas por manobras para estacionamento.

8 REFERÊNCIAS BIBLIOGRÁFICAS

1. BUEHLER, M.; IAGNEMMA, K.; SINGH, S. (Eds.). **The 2005 DARPA Grand Challenge:** The Great Robot Race. [S.l.]: Springer, 2007.

2. BUEHLER, M.; IAGNEMMA, K.; SINGH, S. (Eds.). **The DARPA Urban Challenge:** Autonomous Vehicles in City Traffic. [S.l.]: Springer, 2009.

3. SMARTER THAN YOU THINK - Google Cars Drive Themselves, in Traffic. **New York Times**, 2010. Disponivel em: <http://www.nytimes.com/2010/10/10/science/10google.html?_r=1&>. Acesso em: 22 Setembro 2013.

4. THE self-driving car logs more miles on new wheels. **Google Official Blog**, 2012. Disponivel em: <http://googleblog.blogspot.hu/2012/08/the-self-driving-car-logs-more-miles-on.html>. Acesso em: 22 Setembro 2013.

5. AUTONOMOS Labs. **AutoNOMOS Labs**. Disponivel em: <http://www.autonomos.inf.fu-berlin.de/>. Acesso em: 22 Setembro 2013.

6. AUTONOMOS Labs. **Autonomous Car Navigates the Streets of Berlin**, 2011. Disponivel em: <http://autonomos.inf.fu-berlin.de/news/press-release-92011>. Acesso em: 22 Setembro 2013.

7. MERCEDES-BENZ mostra carro que anda sozinho. **G1**, 2013. Disponivel em: <http://g1.globo.com/carros/noticia/2013/09/mercedes-benz-mostra-carro-que-anda-sozinho-em-previa-de-frankfurt.html>. Acesso em: 22 Setembro 2013.

8. GIZMAG takes a ride in Volvo's most autonomous car yet. **Gizmag**, 2013. Disponivel em: <http://www.gizmag.com/volvo-autonomous-cars/28161/>. Acesso em: 22 Setembro 2013.

9. NISSAN Says: Leave the Driving to Us. **IEE SPECTRUM**, 2013. Disponivel em: <http://spectrum.ieee.org/tech-talk/green-tech/advanced-cars/nissans-ghosn-says-leave-the-driving-to-us>. Acesso em: 22 Setembro 2013.

10. HART, P.; NILSSON, N.; RAPHAEL, B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. **IEEE Transactions on Systems Science and Cybernetics**, 4, n. 2, 1968. 100-107.

11. STENTZ, A. Optimal and Efficient Path Planning for Unknown and Dynamic Environments. **International Journal of Robotics and Automation**, 10, 1993. 89-100.

12. KAVRAKI, L. E. et al. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. **IEEE Transactions on Robotics and Automation**, 12, n. 4, 1996. 566-580.

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

14. LAVALLE, S. M.; KUFFNER, J. J. Rapidly-Exploring Random Trees: Progress and Prospects. In: _____ **Algorithmic and Computational Robotics:** New Directions. [S.l.]: A K Peters/CRC Press, 2001. p. 293-308.

15. HOWIE, C. et al. **Principles of Robot Motion:** Theory, Algorithms and Implementations. [S.l.]: Bradford, 2005.

16. DONALD, B. et al. Kinodynamic motion planning. **Journal of the ACM (JACM)**, 40, n. 5, 1993. 1048-1066.

17. ELFES, A. **Occupancy grids:** A stochastic spatial representation for active robot perception. Sixth Conference on Uncertainty in Artificial Intelligence. [S.l.]: [s.n.]. 1990. p. 136-146.

18. VOUGIOUKAS, S. G. **Optimization of Robot Paths Computed by Randomized Planners**. International Conference on Robotics and Automation. [S.l.]: [s.n.]. 2005. p. 2148-2153.

19. EL-KHATIB, M. M.; HAMILTON, D. J. **A Layered Fuzzy Controller For Nonholonomic Car-Like Robot Motion Planning**. International Conference on

Mechatronics. [S.l.]: [s.n.]. 2006. p. 194-198.

20. LEE, K. et al. **Car parking control using a trajectory tracking controller**. International Joint Conference SICE-ICASE. [S.l.]: [s.n.]. 2006. p. 2058-2063.

21. GUO, Y.; BALAKRISHNAN, M. **Complete Coverage Control for Nonholonomic Mobile Robots in Dynamic Environments**. International Conference on Robotics and Automation. [S.l.]: [s.n.]. 2006. p. 1704-1709.

22. FERGUSON, D.; HOWARD, T. M.; LIKHACHEV, M. Motion Planning in Urban Environments. **Journal of Field Robotics**, 25, n. 11-12, 2008. 939-960.

23. KUWATA, Y. et al. **Motion Planning for Urban Driving using RRT**. International Conference on Intelligent Robots and Systems. [S.l.]: [s.n.]. 2008. p. 1681-1686.

24. SHKOLNIK, A.; WALTER, M.; TEDRAKE, R. **Reachability-guided sampling for planning under differential constraints**. International Conference on Robotics and Automation (ICRA}. [S.l.]: [s.n.]. 2009. p. 2859-2865.

25. CHENG, P.; LAVALLE, S. M. **Reducing metric sensitivity in randomized trajectory design**. International Conference on Intelligent Robots and Systems. [S.l.]: [s.n.]. 2001. p. 43-48.

26. CHENG, P.; LAVALLE, S. M. **Resolution Complete Rapidly-Exploring Randon Trees**. International Conference on Robotics & Automation. [S.l.]: [s.n.]. 2002. p. 267-272.

27. KONOLIGE, K. **A gradient method for realtime robot control**. International Conference on Intelligent Robots and Systems. [S.l.]: [s.n.]. 2000. p. 639-646.

28. REEDS, J. A.; SHEPP, L. A. Optimal Paths for a Car tha goes both Forwards and Backwards. **Pacific Journal of Mathematics**, 145, n. 2, 1990.

29. **TORC Robotics**. Disponivel em: <http://www.torcrobotics.com/>. Acesso em: 01 jul. 2013.

30. **Point Grey**. Disponivel em: <http://ww2.ptgrey.com/>. Acesso em: 01 jul. 2013.

31. **Velodyne Lidar**. Disponivel em: <http://www.velodynelidar.com>. Acesso em: 2013 jul. 01.

32. **Xsens**. Disponivel em: <http://www.xsens.com>. Acesso em: 03 jul. 2013.

33. MONTEMERLO, M.; ROY, N.; THRUN, S. **Perspectives on standardization in mobile robot programming:** The carnegie mellon navigation (CARMEN) toolkit. International Conference on Intelligent Robots and Systems (IROS). [S.l.]: [s.n.]. 2003. p. 2436-2441.

34. SIMMONS, R. The inter-process communication (IPC) system. Disponivel em: <http://www.cs.cmu.edu/afs/cs/project/TCA/www/ipc/ipc.html>.

35. HOHPE, G.; WOOLF, B. **Enterprise Integration Patterns:** Designing, Building, and Deploying Messaging Solutions. 1. ed. [S.l.]: Addison-Wesley Professional, 2003.

36. YOUNG et al. **Física 1:** Mecânica. 12ª. ed. São Paulo: Pearson Addison Wesley.

37. MATTHAEI, G.; JONES, E. M. T.; YOUNG, L. **Microwave Filters, Impedance-Matching Networks, and Coupling Structures**. [S.l.]: McGraw-Hill, 1964.

38. ES TV 1ª Edição. **Departamento de Informática da Universidade Federal do ES desenvolve 'carro autônomo'**, 2013. Disponivel em: <http://g1.globo.com/videos/espirito-santo/estv-1edicao/t/edicoes/v/departamento-de-informatica-da-universidade-federal-do-es-desenvolve-carro-autonomo/2500413/>. Acesso em: 14 jul. 2013.

39. G1. **Alunos e professores do ES desenvolvem carro que anda sozinho**, 2013. Disponivel em: <http://g1.globo.com/espirito-santo/noticia/2013/04/alunos-e-professores-do-es-desenvolvem-carro-que-anda-sozinho.html>. Acesso em: 14 jul. 2013.

40. JORNAL Hoje. **Professores e alunos de universidade do ES criam carro que**

anda sozinho, 2013. Disponivel em: <http://g1.globo.com/jornal-hoje/videos/t/edicoes/v/professores-e-alunos-de-universidade-do-es-criam-carro-que-anda-sozinho/2515111/>. Acesso em: 14 jul. 2013.

41. G1. **G1 faz 'test-drive' em carro que atropelou Ana Maria Braga**, 2013. Disponivel em: <http://g1.globo.com/espirito-santo/noticia/2013/04/g1-faz-test-drive-em-carro-que-atropelou-ana-maria-braga.html>. Acesso em: 14 jul. 2013.

42. TV Capixaba - Espaço Sustentável. **Carro Sustentável da UFES**, 2013. Disponivel em: <http://www.youtube.com/watch?feature=player_embedded&v=xYnfNkCHeb4>. Acesso em: 14 jul. 2013.

43. MAIS Você. **Ana Maria chega ao Mais Você em carro que anda sem motorista**, 2013. Disponivel em: <http://tvg.globo.com/programas/mais-voce/videos/t/programas/v/ana-maria-chega-ao-mais-voce-em-carro-que-anda-sem-motorista/2530185/>. Acesso em: 14 jul. 2013.

44. DOLGOV, D. et al. Path Planning for Autonomous Driving in Unknown Environments. In: KHATIB, O.; KUMAR, V.; PAPPAS, G. J. **Experimental Robotics - The Eleventh International Symposium**. [S.l.]: Springer Berlin Heidelberg, 2009. p. 55-64.

45. GONÇALVES, M. A. **Algoritmo A-Estrela de Estado Hibrido Aplicado à Navegação Autônoma de Veículos**. [S.l.]: [s.n.], 2013.

46. URMSON, C. et al. Autonomous driving in urban environments: Boss and the urban challenge. **Journal of Field Robotics: Special Issues on the 2007 DARPA Urban Challenge**, 2008. 425-466.

47. PEPY, R.; LAMBERT, A. **Safe Path Planning in an Uncertain-Configuration**. International Conference on Intelligent Robots and Systems. [S.l.]: [s.n.]. 2006. p. 5376-5381.