MAURÍCIO PIRAMIDES TEIXEIRA SOARES
LUCAS NEACHIC VASQUES

MODELO DE NAVEGAÇÃO PARA VEÍCULOS AUTÔNOMOS VALIDADO EM MANOBRAS DE ESTACIONAMENTO

São Paulo
2023

MAURÍCIO PIRAMIDES TEIXEIRA SOARES
LUCAS NEACHIC VASQUES

MODELO DE NAVEGAÇÃO PARA VEÍCULOS AUTÔNOMOS VALIDADO EM MANOBRAS DE ESTACIONAMENTO

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Mecatrônico.

São Paulo
2023

MAURÍCIO PIRAMIDES TEIXEIRA SOARES
LUCAS NEACHIC VASQUES

MODELO DE NAVEGAÇÃO PARA VEÍCULOS AUTÔNOMOS VALIDADO EM MANOBRAS DE ESTACIONAMENTO

Trabalho apresentado à Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro Mecatrônico.

Orientador:

Profa. Dra. Larissa Driemeier

Co-orientador:

Felipe Gomes de Melo D'Elia

São Paulo
2023

Catalogação-na-publicação

Soares, Maurício
Modelo virtual de planejamento e decisão para estacionamento de veículos autônomos / M. Soares, L. Vasques -- São Paulo, 2023.
57 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1.Robótica 2.Veículos autônomos 3.Direção veicular 4.Controle I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos II.t. III.Vasques, Lucas

AGRADECIMENTOS

Ao co-orientador Felipe Gomes de Melo D'Elia, pela motivação e suporte ao longo do desenvolvimento do trabalho.

À Professora Doutora Larissa Driemeier, pelo acolhimento, paciência e paixão pelo projeto.

RESUMO

O erro humano no trânsito se mostra um grande causador de acidentes e consequentes mortes no ambiente urbano. Apesar da desconfiança pública em relação aos veículos autônomos, estes demonstram uma navegação mais segura, em média, do que os motoristas humanos. Este trabalho se esforça para identificação e aplicação de uma tecnologia mais inteligível e facilmente auditável que a alternativa de aprendizado de máquina para planejamento de rota e navegação, validando-a por meio da execução de manobras de estacionamento em ambiente simulado.

A tecnologia selecionada para a estruturação do sistema foi a de Árvores de Comportamento, uma estrutura de Inteligência Artificial básica, que garante maior inteligibilidade, auditabilidade, modularidade e escalabilidade que outras opções, como Redes Neurais ou Máquinas de Estados. A plataforma para a construção do sistema foi a biblioteca *Navigation2* para *Robot Operating System 2*, que emprega a tecnologia para navegação robótica.

Prevendo os desafios associados ao planejamento de rotas em espaços urbanos repletos de obstáculos, a implementação do *Simultaneous Localization and Mapping* (SLAM) permitiu mapear as vias efetivamente, gerando uma base sólida para o planejamento de rota e navegação.

O uso de recursos da biblioteca *Navigation2*, como o planejador *Smac Hybrid A** em conjunto com o controlador RPP (do inglês *Regulated Pure Persuit*), revelou-se eficaz para a navegação em geral. Contudo, apresentou limitações ao planejar rotas em espaços muito próximos a obstáculos, sugerindo a necessidade de desenvolver uma Árvore de Comportamento personalizada para manobras de estacionamento paralelo, embora esta, por mais que projetada, não foi validada.

Palavras-Chave – Robótica, Veículos autônomos, Direção veicular, Controle.

ABSTRACT

Human error in traffic is a major cause of accidents and consequent deaths in urban environments. Despite public skepticism about autonomous vehicles, they demonstrate, on average, safer navigation than human drivers. This work strives for the identification and application of a technology that is more understandable and easily auditable than the machine learning alternative for route planning and navigation, validating it through the execution of parking maneuvers in a simulated environment.

The technology selected for the system's structure was Behavior Trees, a basic Artificial Intelligence framework that ensures greater intelligibility, auditability, modularity, and scalability than other options, such as Neural Networks or State Machines. The platform for building the system was the *Navigation2* library for *Robot Operating System 2*, which employs the technology for robotic navigation.

Anticipating the challenges associated with route planning in urban spaces filled with obstacles, the implementation of Simultaneous Localization and Mapping (SLAM) allowed for effective mapping of the roads, generating a solid foundation for route planning and navigation.

The use of resources from the *Navigation2* library, such as the *Smac Hybrid A** planner in conjunction with the Regulated Pure Pursuit (RPP) controller, proved effective for general navigation. However, it showed limitations when planning routes in spaces very close to obstacles, suggesting the need to develop a custom Behavior Tree for parallel parking maneuvers, although this, despite being designed, was not validated.

Keywords – Robotics, Autonomous vehicles, Vehicle steering, Control.

SUMÁRIO

1 INTRODUÇÃO

Acidentes de trânsito causam cerca de 1,35 milhão de mortes todos os anos e são a principal causa entre jovens de 5 a 29 anos [1]. Além disso, dentre 20 a 50 milhões de pessoas são feridas ou tem sequelas devido a colisões no tráfego a cada ano [2]. As principais causas dessas ocorrências são excesso de velocidade, conversões impróprias e violações de regulações de trânsito [3], sendo elas categorizadas como erro humano.

Veículos autônomos (VAs) e seus sistemas de percepção, decisão e controle são áreas de pesquisa e desenvolvimento nas quais o Engenheiro Mecatrônico pode empregar suas capacitações em automação para impactar a segurança no tráfego urbano por meio da prevenção de acidentes causados por erro humano. Nos últimos anos, estes campos de desenvolvimento mostraram significativos avanços tecnológicos no ambiente acadêmico e no mercado automotivo de ponta [4].

Atualmente, os Sistemas Avançados de Assistência ao Motorista (ADAS, do inglês Advanced Driver-Assistance System), que são automações pontuais para situações recorrentes no trânsito, são vastamente utilizados nos veículos presentes nas ruas. Alguns exemplos desse tipo de automação são: frenagem automática para evitar colisões frontais, aviso e prevenção de saída de faixa em casos de risco e detecção de ponto cego e previsão de colisão traseira. A utilização dessas tecnologias é um primeiro passo para a automação total mas já se mostra eficaz na aumento da segurança por meio da prevenção de acidentes [5].

Naturalmente, conclui-se que implementações com maiores níveis de automação resultam em mais segurança, mas para levar tal solução às ruas é preciso atentar-se à tecnologia do sistema de planejamento de navegação utilizada. Análises de dados de acidentes envolvendo direção autônoma identificam que a maioria das falhas ocorrem no sistema de decisão implementado com o uso de aprendizado de máquina e alertam para a necessidade de melhoria contínua da tecnologia [6]. Adicionalmente, as implementações que utilizam de tal recurso são recentes, nebulosas e imprevisíveis ao olhar humano, dificultando suas certificações e implementação da tecnologia nas ruas [7]. Em vista disso, nota-se a neces-

sidade de implantar o sistema de decisão destes veículos utilizando uma tecnologia mais facilmente auditável e clara, de modo a popularizar VAs o quanto antes visando preservar vidas, uma vez que estes alcançam níveis de segurança maiores que a direção de um motorista comum [8].

Para ultrapassar esses obstáculos, torna-se necessário o esforço na pesquisa e seleção da ferramenta ideal para a tarefa, a qual deve ser robusta, acessível e inteligível tanto para a máquina quanto para o homem. Desse modo, será possível desfrutar de benefícios que vão além da segurança, como eficiência energética, redução do esforço físico e mental de motoristas, otimização do tráfego de veículos e redução do tempo de deslocamento [9] [10].

Este trabalho tem como objetivo contribuir para o aumento da segurança no trânsito por meio da implementação de um sistema de planejamento e execução de rota automático, validado em cenários simulados, como a execução de manobras de estacionamento. A simulação será fundamentada na configuração e recursos de um protótipo em escala específico, referenciado na Seção 2.4, de forma a facilitar futura implementação embarcada.

2 REVISÃO DO ESTADO DA ARTE

2.1 Veículos autônomos na segurança do trânsito

Diversos campos de pesquisa buscam elevar a segurança no trânsito, sendo esses esforços classificados em duas categorias: a remediação, por meio de tecnologias de absorção de impacto [11]; e a prevenção, como a aplicação da engenharia de tráfego [12] [13] e o desenvolvimento de veículos autônomos. À seguir, será capturado e analisado o contexto tecnológico da área de direção automática, na qual este projeto pretende atuar.

A construção de um VA envolve uma série de tecnologias avançadas que trabalham em conjunto para garantir uma condução precisa e segura. Seu funcionamento interno subdividido em três partes correlacionadas. A percepção, que é a capacidade de receber e processar informações sensoriais do ambiente em que está operando [14], o planejamento, caracterizado pela seleção de ações conforme o contexto dado pela percepção [15], e o controle, correspondendo à operação do sistema de direção, aceleração e frenagem do veículo, de acordo com a decisão tomada e atentando à dinâmica do sistema [16]. Dentre tais subdivisões, a que se destaca por seus métodos de aplicação inovadores e desenvolvimento intenso na academia e na indústria é a que lida com planejamento e tomada de decisão [4].

Atualmente, VAs são taxonomicamente categorizados em níveis numerados de 0 a 5, sendo que o nível zero é destacado por oferecer recursos básicos como avisos e frenagem de emergência e, conforme o nível aumenta, suas limitações e a necessidade de interação do motorista durante a viagem decrescem, até o último nível, caracterizado pela automação total e funcionamento sobre quaisquer condições [17]. A Figura 2.1 ilustra as categorias citadas.

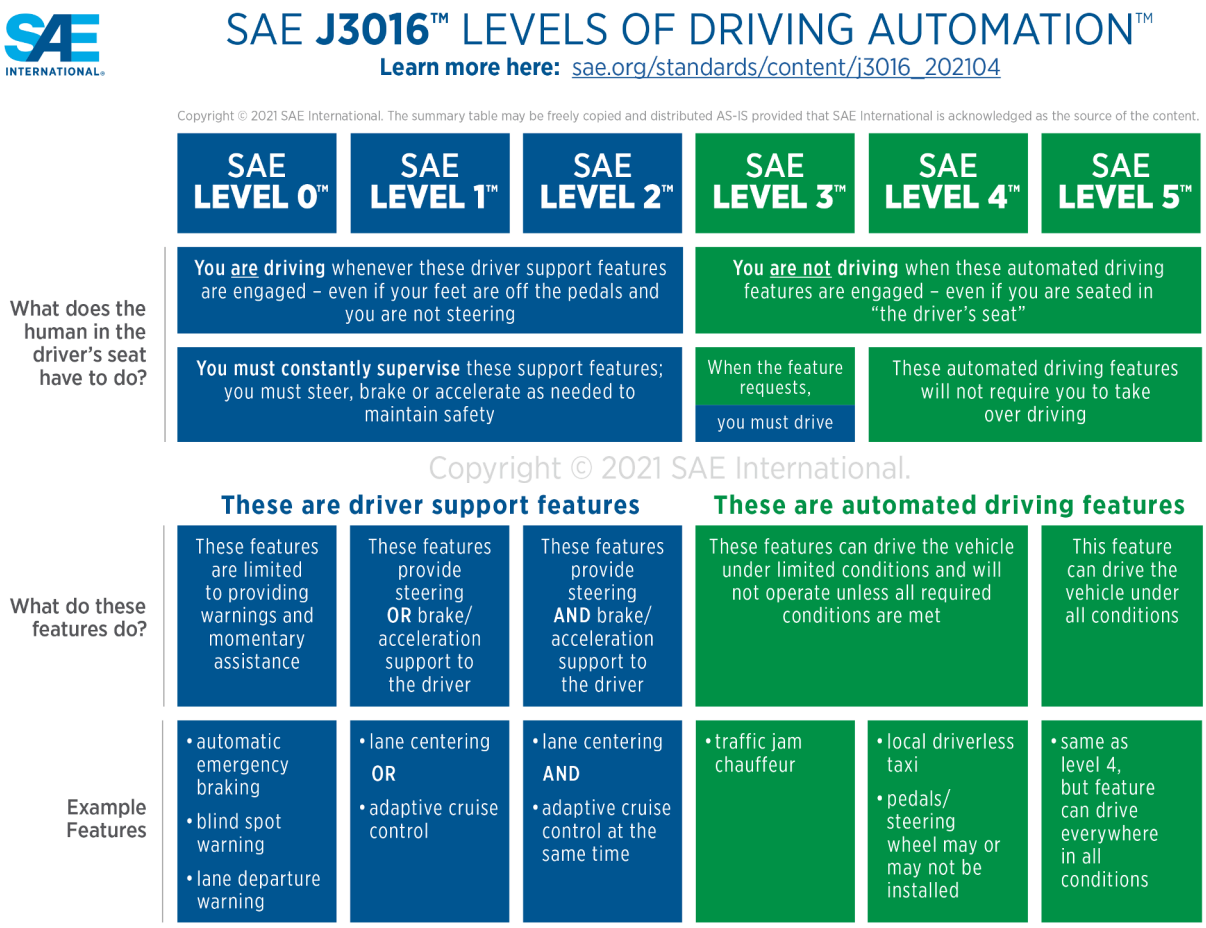


Figura 2.1: Classificação taxonômica dos níveis de automação veicular (Reproduzido de: [17])

A fim de atingir cada vez mais altos níveis na escala, soluções de automação tais como o uso de redes neurais e aprendizado por reforço têm sido exploradas no desenvolvimento de sistemas de planejamento e tomada de decisão dos veículos autônomos [18]. Contudo, tal metodologia levanta questões acerca da garantia segurança, como, por exemplo, quanto treinamento é suficiente para um desempenho seguro da rede? Métodos como esse ainda não possuem um sistema claro e objetivo de análise de desempenho e segurança [18]. Além disso, uma metodologia eficiente de validação, que comprove a robustez da rede, inclusive em casos extremos, ainda é incipiente.

Tendo em vista os problemas mencionados no desenvolvimento e certificação de sistemas de planejamento e tomada de decisão de VAs, alternativas que utilizam técnicas de IA clássica se apresentam como foco de outros trabalhos na área. Dentre essas técnicas está a Máquina de Estados Finitos (FSM do inglês Finite State Machine), que, por mais que seja vastamente utilizada devido à sua estrutura intuitiva, torna-se complexa e poluída rapi-

damente com o aumento da complexidade do sistema. Isso levou à criação das Máquinas de Estado Finitos Hierárquicas (HFSM do inglês Hierarchical Finite State Machine) que introduz contextos e transições entre contextos além de somente estados e transições entre estados. Ainda assim, existe um modelo de IA clássica que garante escalabilidade, modularidade e legibilidade para sistemas complexos: a Árvore de Comportamento (do inglês *Behaviour Trees*) [19].

2.2 Árvores de Comportamento

2.2.1 Princípios

Atualmente, as Árvores de Comportamento são usadas em diferentes áreas, incluindo robótica industrial, jogos eletrônicos, robótica móvel e sistemas multi-robôs, sendo, portanto, capazes de controlar robôs de forma autônoma e adaptativa [20]. Conforme ilustra a Figura 2.2, são estruturas hierárquicas compostas por nós que representam condições (formas ovais) ou ações (formas retangulares) que o sistema pode executar. Esses nós são conectados a outros nós lógicos que estabelecem a ordem das execuções de seus nós filhos. Quando colocada em funcionamento, a Árvore de Comportamento, partindo do nó raiz e da esquerda para a direita, executa seus nós folhas em certa frequência, que podem retornar respostas de execução de *sucesso, falha* ou *pendente*. Essas respostas são utilizadas pelos nós lógicos para seguir com a execução da árvore [19].

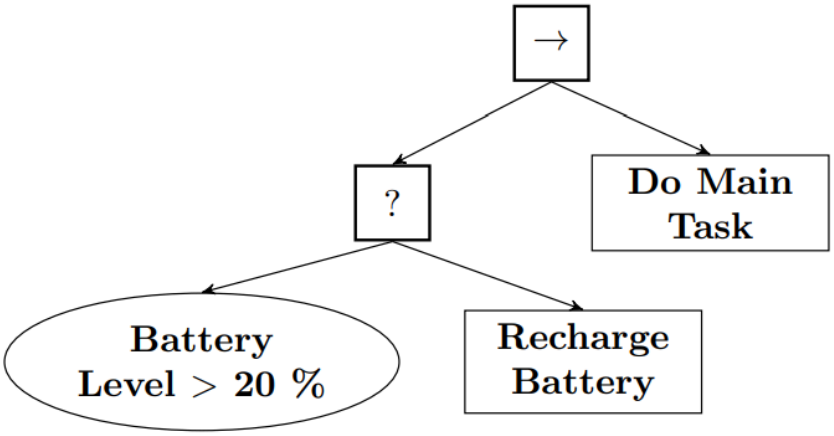
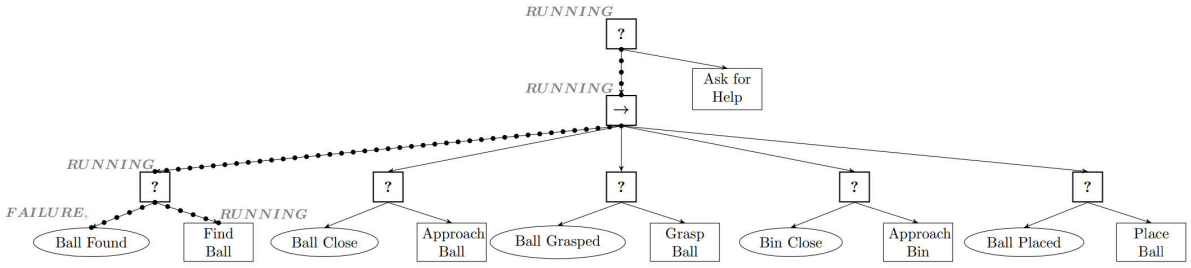


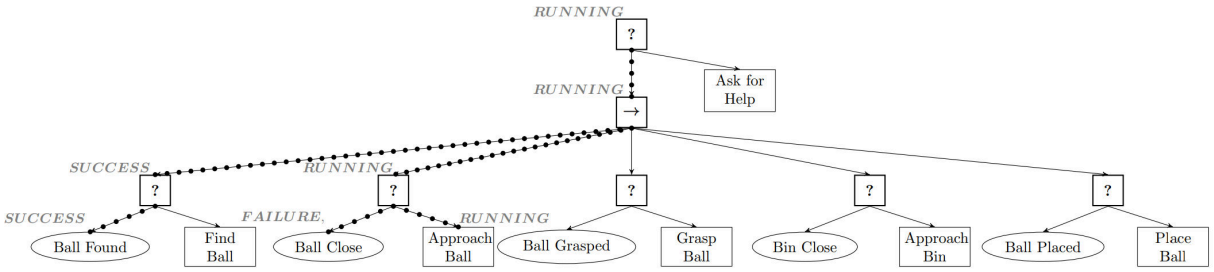
Figura 2.2: Exemplo de Árvore de Comportamento básica para um robô que necessita de recarga de bateria (Reproduzido de: [19])

Dentre os nós lógicos estão: o nó de *fallback* (representado por "?"), o nó de *sequência* (representado por "→") e o nó de *paralelismo* (representado por "⇉"). O nó de *fallback*

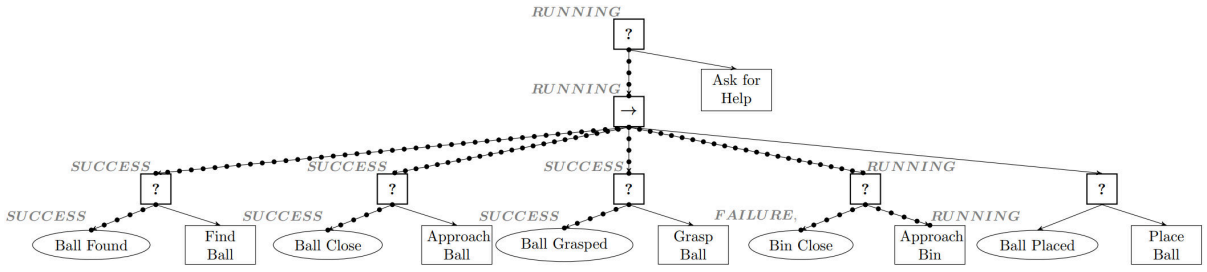
comporta-se analogamente a um operador "ou", executando nós filhos da esquerda para a direita, retornando *sucesso* quando o primeiro nó filho retornar *sucesso*, e retornando *falha* quando todos retornarem falha. Ele é comumente utilizado com um nó de condição que indica o sucesso de uma operação em seu nó filho à esquerda, seguido de ações que buscam alcançar essa condição à direita, em ordem prioritária. O nó de *sequência*, por sua vez, age como um operador "e", e executa os nós filhos, da esquerda para direita e retorna *sucesso* se todos retornarem *sucesso*, e *falha* quando o primeiro retornar *falha*. Esse nó é geralmente utilizado para dividir rotinas com sequência definida em operações menores. Por fim, o nó de *paralelismo* invoca seus nós filhos simultaneamente, retornando *sucesso* se pelo menos certo número definido de nós filhos retornarem *sucesso*, e *falha* caso contrário [19]. Um exemplo de comportamento de algumas dessas operações é ilustrado na Figura 2.3.



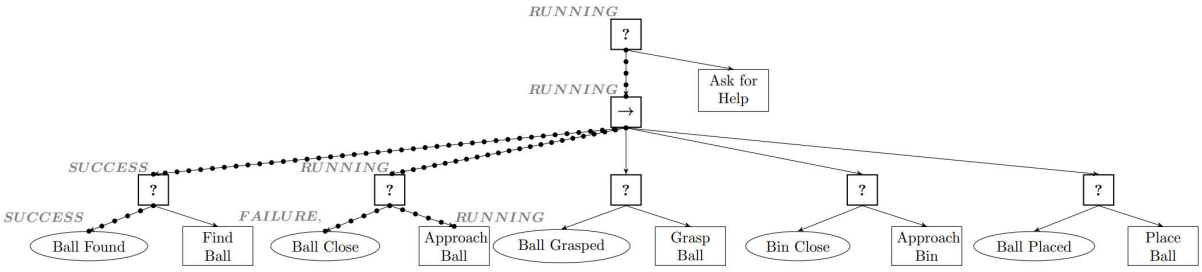
(a) Robô procurando a bola



(b) Robô se aproximando da bola



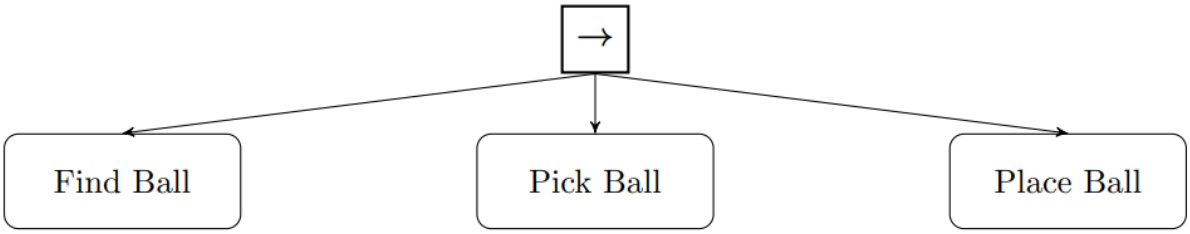
(c) Robô se aproximando da cesta



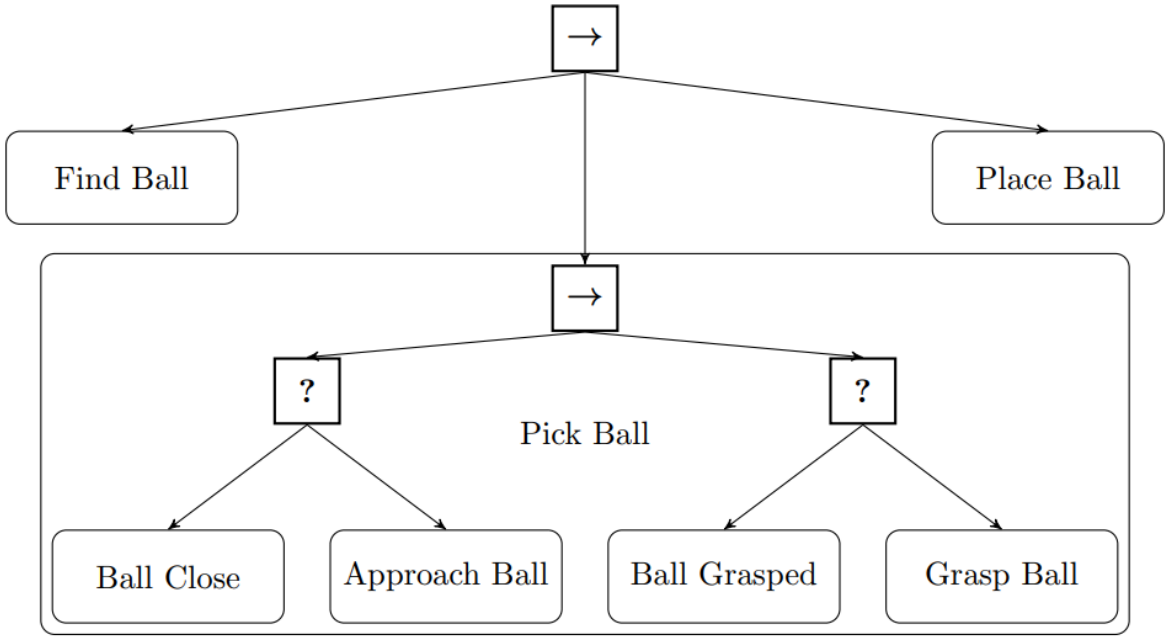
(d) Robô se aproximando da bola novamente, após ser retirada do robô

Figura 2.3: Exemplo de execução de uma Árvore de Comportamento para um manipulador robótico (Reproduzido de: [19])

A modularidade das Árvores de Comportamento é evidente quando um nó de condição ou ação é substituído por uma sub-árvore que representa seu comportamento, garantindo granularidade das operações executadas. Esse aspecto é exemplificado na Figura 2.4.



(a) Árvore de comportamento de alto nível



(b) Árvore de Comportamento de nível menor

Figura 2.4: Exemplo de modularidade das Árvores de Comportamento para um manipulador robótico (Reproduzido de: [19])

Em [20], os autores discutem exemplos específicos de como as Árvores de Comportamento foram implementadas em áreas como a robótica, manipulação, robôs móveis, veículos aéreos e submarinos. Esses exemplos mostram que elas podem ser aplicadas para controlar o comportamento dos robôs de forma autônoma e adaptativa, incluindo tarefas como navegação, desvio de obstáculos, e manipulação de objetos.

Além disso, o estudo compara as Árvores de Comportamento com outras técnicas de planejamento e controle de comportamento, como a FSM e lógica *fuzzy*. É argumentado que as Árvores de Comportamento são mais flexíveis e escaláveis do que as outras técnicas citadas, permitindo a modularização e composição de comportamentos mais complexos a partir de outros mais simples. Isso significa que a árvore pode ser expandida facilmente adicionando novos nós que representam novos comportamentos. Ademais, como as Árvores de Comportamento são hierárquicas, elas podem ser facilmente reorganizadas para acomodar novas ações ou mudanças no ambiente [20].

O estudo também discute desafios e limitações associados ao uso de Árvores de Comportamento. Por exemplo, os autores destacam que a complexidade das árvores pode se tornar um problema em ambientes muito dinâmicos ou com muitos robôs interagindo, visto que a construção não é feita automaticamente. Isso pode levar a erros de gerenciamento e dificuldade na depuração e manutenção do sistema, tornando-o mais propenso a erros e falhas, necessitando de maior emprego de conhecimento técnico em sua confecção [20].

Apesar de apresentar outras tecnologias, no estudo não foi realizada uma comparação completa com outras abordagens para controlar o comportamento robótico, sem exemplos de situações em que árvores são mais vantajosas. Além disso, os autores apresentam um viés positivista sobre a tecnologia, não discutindo adequadamente as desvantagens ou limitações dessa abordagem [20].

2.2.2 Aplicação

Na implementação de Árvores de Comportamento para aplicações em robótica, existem diversas abordagens e ferramentas disponíveis. No contexto deste trabalho, adotamos a biblioteca *BehaviorTree.CPP*, compatível com a biblioteca *Navigation2*, que também será utilizada para navegação e cujos detalhes serão explicados na Seção 2.3. A *BehaviorTree.CPP* é uma biblioteca escrita em C++ dedicada ao desenvolvimento de código para comportamentos em robótica. O comportamento que define nós e suas lógicas internas é definido utilizando funções em C++, enquanto a estrutura da árvore é criada em

XML (do inglês *Extensible Markup Language*), que pode, por sua vez, ser importado e exportado pelo editor gráfico *Groot*, permitindo também a visualização em tempo real da árvore durante operação [21].

A seguir, é apresentado um exemplo simplificado de uma Árvore de Comportamento em *XML*:

```
<root BTCPP_format="3">
    <BehaviorTree ID="Untitled">
        <Fallback>
            <Sequence>
                <VerificarBateria/>
                <Sequence>
                    <MoverParaPonto Ponto=""/>
                    <SelecionaProximoPonto Ponto=""/>
                </Sequence>
            </Sequence>
            <Recarregar/>
        </Fallback>
  </BehaviorTree>

    <TreeNodesModel>
        <Action ID="MoverParaPonto"> <input_port name="Ponto"/> </Action>
        <Action ID="Recarregar" editable="true"/>
        <Action ID="SelecionaProximoPonto"> <inout_port name="Ponto"/ </Action>
        <Condition ID="VerificarBateria"/>
  </TreeNodesModel>
</root>
```

Listagem 2.1: Árvore de Comportamento em XML

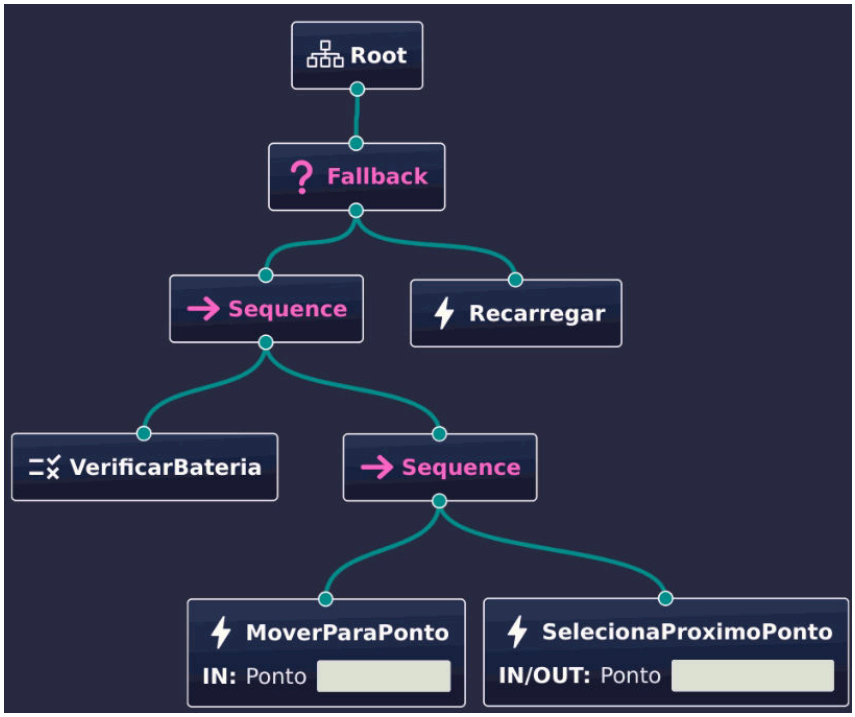


Figura 2.5: Exemplo de Árvore de Comportamento básica gerada pelo *Groot*

Neste exemplo, temos um cenário de controle de um robô move para um ponto específico e atualiza esse ponto quando chega ao destino. Mantém esse ciclo até que o nível de bateria não seja mais suficiente, retornando falha no nó de condição, de modo que o segundo filho do nó de controle *fallback* é executado, sendo ele a ação de recarregar a bateria. Como indica a Figura 2.5, no visualizador *Groot* é possível adicionar portas de entrada e saída, permitindo a interação entre nós de uma árvore.

Assim como no modelo teórico, na *BehaviorTree.CPP*, uma Árvore de Comportamento é composta por nós, agrupados em quatro categorias principais. Os *ControlNodes* (Nós de Controle) organizam a lógica da árvore, decidindo a execução de seus filhos com base em condições predefinidas. Por outro lado, os *DecoratorNodes* (Nós Decoradores) modificam o resultado de seus filhos ou executam esses nós repetidamente, oferecendo uma flexibilidade valiosa. *ConditionNodes* (Nós de Condição) avaliam aspectos do sistema, sem alterá-lo. Enquanto isso, os *ActionNodes* (Nós de Ação) representam as unidades fundamentais de execução, realizando ações específicas no ambiente do robô [22].

Além dos nós padrão, a *BehaviorTree.CPP* oferece integração com o ROS 2. Nós específicos do ROS 2 pertencentes às quatro categorias (*action*, *condition*, *control*, *decorator*) podem ser importados e utilizados na construção de Árvores de Comportamento, facilitando a comunicação e coordenação com outros componentes do sistema robótico [23].

A sinergia entre Árvores de Comportamento e o ROS 2 abre caminho para uma abordagem modular e reutilizável no desenvolvimento de sistemas robóticos. Essa integração oferece uma solução eficaz e flexível para coordenar o comportamento de robôs em ambientes dinâmicos e complexos. Além disso, a *BehaviorTree.CPP* proporciona não apenas uma gama completa de tipos de nós, mas também a conveniência de acessar árvores completas já implementadas na biblioteca de *Navigation2*, proporcionando uma solução robusta e eficiente [24].

2.3 *Navigation2* - O experimento Marathon 2

Sistemas de navegação robótica com o uso da tecnologia de Árvore de Comportamento já foram implementados e validados em determinadas condições. Tal esforço resultou na produção e disponibilização de uma biblioteca de navegação de código aberto para a plataforma *Robotic Operational System 2* (ROS 2) [25], a *Navigation2*, que utiliza da biblioteca *BehaviorTree.CPP* como base. O recurso provém de uma refatoração e modernização da principal biblioteca de navegação do ROS original, a *ROS Navigation Stack*, a fim de adicionar o suporte a novos sistemas de percepção e utilizar o sistema de comunicação sólido da nova edição da plataforma.

Para implementação, o uso do ROS 2 facilita o desenvolvimento de projetos robóticos e sistemas autônomos, uma vez que é uma plataforma modular e colaborativa de software de código aberto, com diversas bibliotecas disponíveis e em desenvolvimento. Um dos principais diferenciais do ROS 2 é seu sistema de comunicação entre unidades de software, chamadas de *nós*, por meio de um sistema de publicação e subscrição em canais de eventos, chamados de *tópicos*. Essa comunicação utiliza o padrão *Data Distribution Service* (DDS), que proporciona robustez e escalabilidade para sistemas em tempo real [26], além de segurança, aspecto que é reconhecidamente insatisfatório no ROS 1 [25].

No que se refere à técnica de planejamento utilizada pelo pacote, a tecnologia de Árvores de Comportamento se mostrou facilmente configurável e permite o reuso de um vasto número de primitivos evitando o retrabalho. Em [25], o autor também ressalta que tal modelo de IA clássica é amplamente utilizado em tarefas robóticas como execução de missões e manipulação móvel e complexa de objetos. Na biblioteca, o recurso é aplicado na orquestração de *nós* ROS 2 escritos em diversas linguagens, cada um gerenciado por um *nó* da Árvore de Comportamento. Essa arquitetura de controle, quando somada a organização modular e paralelizada da plataforma, permite o multiprocessamento e uso otimizado dos recursos computacionais [25], o que garante o tempo real de resposta para

aplicações nas quais isso é crítico, como os veículos autônomos.

A *Navigation2* foi feita com modularidade e configurabilidade em mente, a fim de oferecer suporte a sistemas de percepção e modelos físicos de robôs diversos sem complicações. O uso do modelo de Árvores de Comportamento colabora nessa iniciativa, pois, além de fornecer alta reusabilidade de módulos e acoplamentos desses, o ato de modificar sua estrutura se resume à trivial edição de um arquivo em formato XML, que pode ser carregado em tempo de execução. A árvore utilizada pode ser customizada livremente, permitindo ao usuário definir protocolos de percepção, controle e recuperação específicos ao seu projeto [25].

A validação da biblioteca, no entanto, se deu por meio do experimento *Marathon 2*, o qual implementou o sistema em 2 robôs industriais: Tiago e RB-1, cada qual com suas especificidades físicas e sistemáticas, e os submeteu à tarefa de percorrer um circuito na Universidade Rey Juan Carlos, um ambiente com alta movimentação e obstáculos dinâmicos, ao longo do comprimento de uma maratona sem intervenção humana conforme ilustra a Figura 2.6. Este experimento confirmou a alta robustez e segurança do pacote na plataforma ROS 2 com o sucesso na execução da tarefa, mas se restringiu ao uso de robôs com direção holonômica e os autores prevêem obstáculos na implementação de um sistema de planejamento que leve em conta a incapacidade de rotação *in loco* de robôs que assemelham-se a carros reais [25].

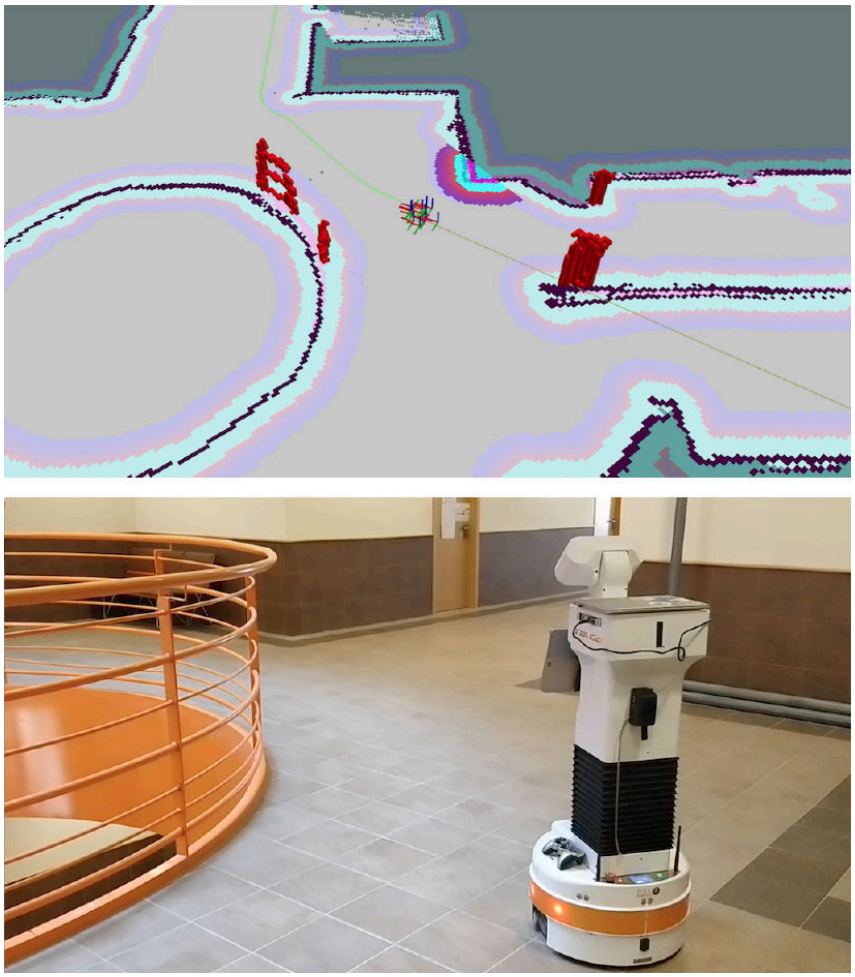


Figura 2.6: Experimento Marathon2 (Reproduzido de: [25])

2.4 Plataforma legada

Este projeto busca superar o desafio de implementar um sistema de planejamento de rota para veículos com geometria padrão Ackermann utilizando a ferramenta de Árvores de Comportamento e a biblioteca *Navigation2* desenvolvida para a plataforma ROS 2 em ambiente simulado, com modelo e recursos virtuais similares aos disponíveis no protótipo de carro autônomo em escala 1:10 presente no Laboratório de Impacto de Estruturas (GMSIE) da POLI-USP, de forma a facilitar posterior embarcação do sistema. O protótipo que foi produzido em [27] e está ilustrado na Figura 2.7, consiste em um projeto de carro autônomo em escala reduzida, que foi construído com um sistema de energia mecânica composto por um motor sem escova *Maxon EC-4 pole* com 30mm e 24V, conectado a um sistema de transmissão de engrenagens com trem diferencial nas rodas traseiras. O sistema de energia elétrica utiliza o controlador de velocidade eletrônico de código aberto VESC (*Vedder Eletronic Speed Controler*) e 2 células de baterias LiPo de 2000mAh, enquanto

o sub-sistema de sensores e processamento é composto por uma unidade principal de processamento *Jetson Xavier NX*, 3 módulos de câmeras estéreo USB ELP 1MP 120FOV e 1 módulo de câmera estéreo CSI IMX218-83. O projeto incluiu também a fabricação de um conjunto de cones de trânsito miniaturizados (1:10 de acordo com a norma NBR 15071) com uma impressora 3D por deposição de material fundido, utilizados originalmente como obstáculo reconhecido pelo sistema de percepção [27]. Estes obstáculos serão reproduzidos em ambiente simulado e serão úteis para a validação do sistema de planejamento.

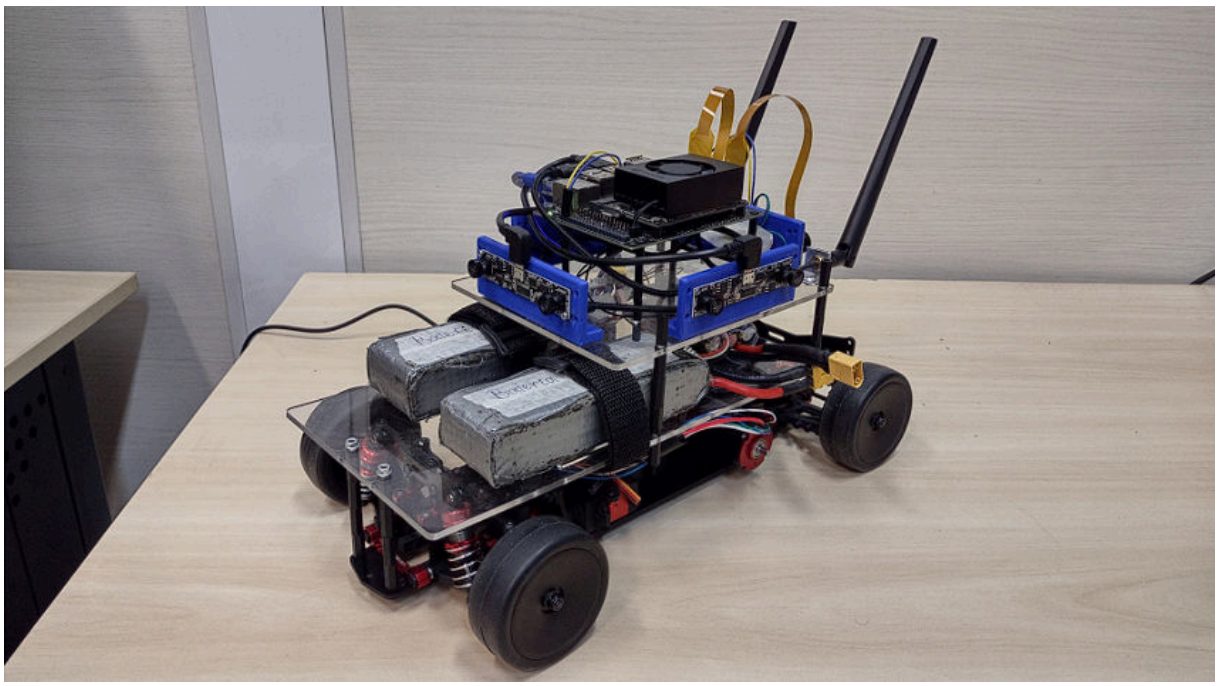


Figura 2.7: Protótipo de veículo em escala 1:10 (Reproduzido de: [27])

2.4.1 Percepção

A partir das câmeras e sensores, obtém-se uma compreensão do ambiente ao redor do veículo. Para tal, foi necessária a tratativa e processamento dos dados recebidos a fim de construir um sistema de percepção para condução autônoma. Primeiramente, foi preciso realizar a calibração das câmeras, corrigindo erros de distorções radiais, devido à curvatura das lentes, e tangenciais, por problemas de alinhamento da lente com o sensor. Em seguida, foi preciso gerar a homografia planar, que é o mapeamento projetivo do plano visualizado nas câmeras para o plano superior, obtido através de uma multiplicação matricial, visto que se trata de uma relação homogênea entre planos [27].

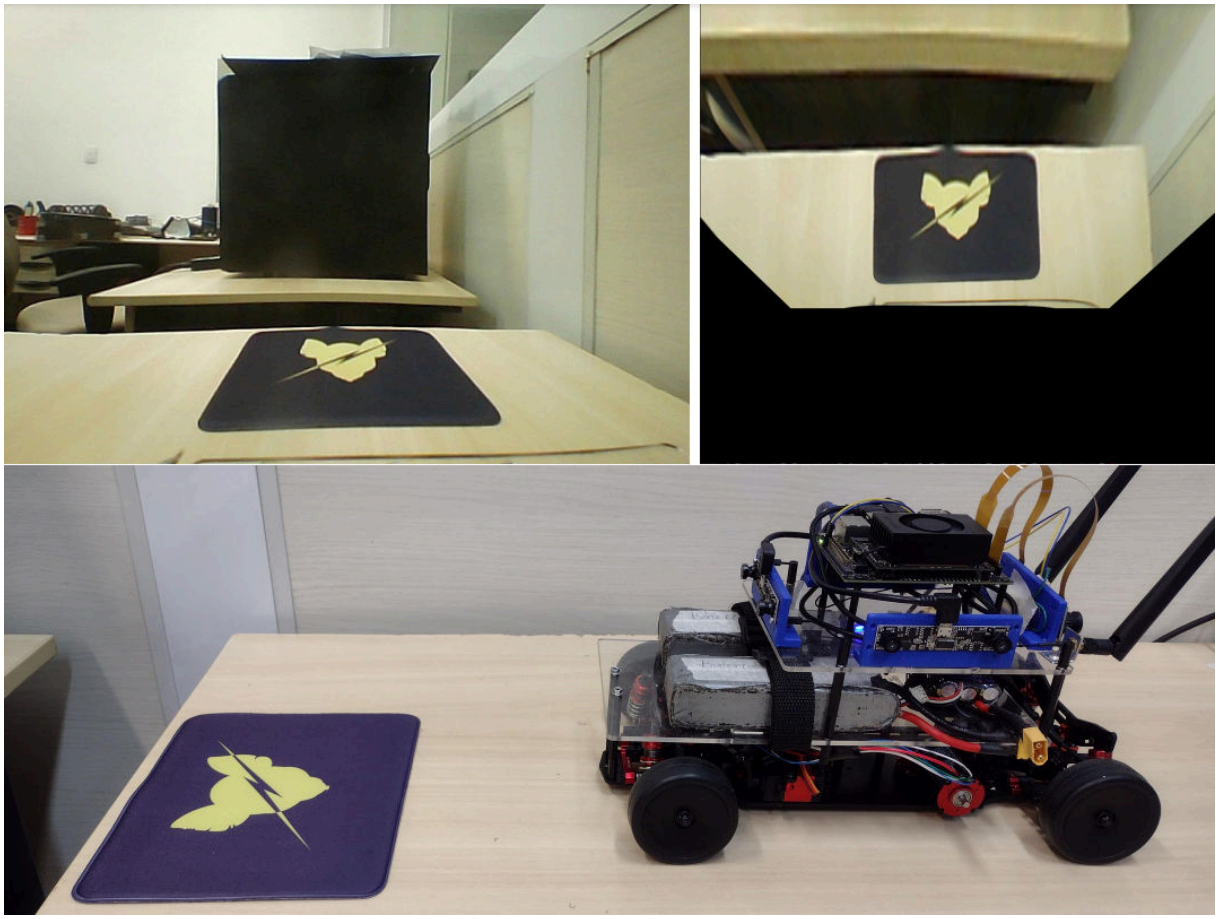


Figura 2.8: Imagem virtual superior a partir da homografia planar (Reproduzido de: [27])

Para reconhecimento de cena, foi proposto um sistema de 4 módulos de câmeras nas laterais do protótipo, alimentando o algoritmo VSLAM (do inglês *Visual Simultaneous Localization and Mapping*) para base da navegação no ambiente. Foi utilizado o ambiente de simulação *CARLA* e o princípio de aproximação geométrica para reconstrução de cena com o processamento de imagem via GPU, permitindo a computação em tempo real, fornecido pela *Jetson Xavier NX*, placa especializada em processamento de imagem embarcado [27].

As imagens obtidas do sistema de câmera proposto foram usadas para alimentar uma rede neural profunda para detecção de objetos. Como um veículo autônomo depende de tempos de resposta rápidos para funcionar corretamente, foi escolhida a solução *YOLOv7*, que supera suas versões anteriores em menor densidade de parâmetros, tempos de inferência mais rápidos e maior precisão. Mais especificamente, utilizou-se a *YOLOv7-tiny*, uma versão reduzida do *YOLOv7*, otimizada para *Jetson Xavier NX*, que processa imagens aproximadamente quatro vezes mais rápido do que a versão completa, mas com uma precisão um pouco menor [27].

Foi utilizada uma combinação do banco de dados *FSOCO* e um conjunto de dados construído com imagens de cones tiradas no campus principal da USP e rotuladas manualmente para treinar a rede. As imagens foram processadas com técnicas de aumento de dados antes da exportação do conjunto de dados. O treinamento da rede usando *YOLOv7-tiny* para reconhecimento de cones obteve um tempo médio de detecção de 52ms e uma predição de mais de 90% [27]. Em função da identificação dos cones de tamanhos definidos, o protótipo se localiza no ambiente e consegue estimar sua posição futura com base em sua velocidade e ângulo de direção, em caso de falha na identificação dos cones.

É válido mencionar que o reconhecimento de objetos da rede de percepção é limitado a cones que seguem a norma NBR 15071, e nenhum outro objeto. Além disso, é relatada a ocorrência de falsos positivos nesse sistema, efeito que pode ser reduzido por meio de filtros temporais e aumento no limite de confiança da percepção [27].

2.4.2 Controle

Para o controle lateral, foi utilizado um controlador de modos deslizantes que recebe o sinal de erro entre o ângulo de direção atual e o ângulo desejado. Para o controle longitudinal, um controlador PID foi usado com a velocidade desejada sendo determinada por parâmetros de ajuste e distância para o ponto seguido ao longo do caminho simulado no *software CARLA* [28]. O controle permitiu a estabilização do sistema e guiou o veículo virtual pelo percurso desejado no simulador a partir de uma lista de pontos de rota pré-programados [27], conforme ilustra a Figura 2.9.

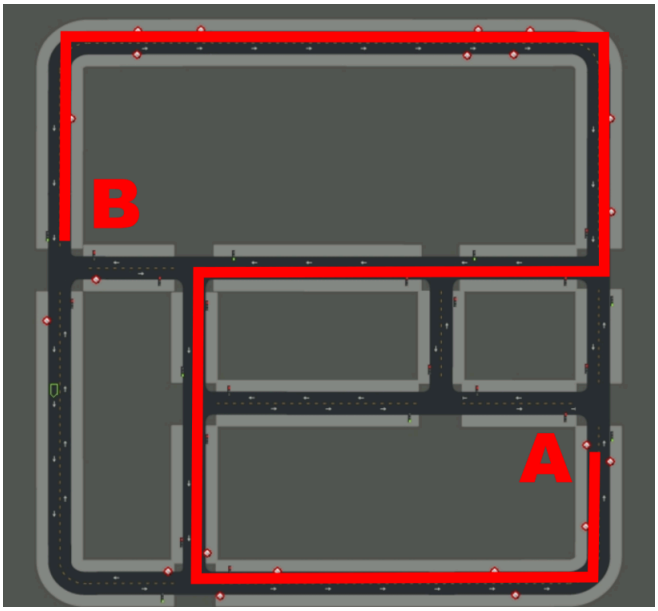


Figura 2.9: Ilustração do caminho percorrido no *CARLA* (Reproduzido de: [27])

Em [27], os autores propuseram-se a desenvolver os sistemas de percepção, de controle e de planejamento do protótipo, mas deixaram de implementar o controle no protótipo físico e o sistema de tomada de decisão autônoma, principalmente por restrição de tempo [27]. Em vista disso, este projeto enxerga tal legado como uma oportunidade, e relatará o trabalho realizado sobre o que já foi construído, com o objetivo de obter um modelo de planejamento e tomada de decisão funcional em ambiente simulado, que evite colisões com obstáculos e seja capaz de realizar manobras, como o estacionamento paralelo, de forma compatível ao protótipo físico para futura embarcação.

2.4.3 *Robot Operating System 2* - ROS 2

Uma vez apresentado o protótipo, é válido esclarecer em detalhes o funcionamento da plataforma ROS 2 e como ela foi utilizada para atingir os seus resultados finais. Como já enunciado, a plataforma ROS 2 proporciona um ambiente modular com um protocolo de comunicação confiável para aplicações de robótica em tempo real, e, por isso, foi escolhido como *framework* tanto do projeto anterior como do presente. Tal modularidade garante facilidade na contribuição e cooperação no desenvolvimento de projetos mecatrônicos. Isto em vista, torna-se possível o aproveitamento do progresso de [27] para a confecção de um modelo virtual funcional.

Desde sua primeira versão, para oferecer a modularidade e colaboração desejada para a plataforma, o ROS é estruturado em processos relativamente independentes chamados *nós*, que se comunicam por interfaces de dados chamadas *tópicos*. Sua primeira versão possuía um protocolo de comunicação entre *nós* próprio baseado em TCP/UDP, que dependia de uma unidade de controle, referida por *rosmaster* [29], o que, inevitavelmente, insere um ponto único de falha crítica para sistemas ROS. Além disso, seu suporte a múltiplas linguagens de programação foi desenvolvido de forma independente para cada uma delas, o que torna a manutenção e o desenvolvimento repetitivo e custoso.

Em vista disso, o ROS 2 surge para adereçar esses problemas, adotando o *middleware* de comunicação DDS, que, além de ser altamente testado e certificado [30], remove a responsabilidade dos desenvolvedores ROS de manter seu próprio modelo de comunicação. Em adição a isso, a adoção de uma biblioteca base escrita em *C*, a *rcl*, garante um foco de manutenção e desenvolvimento centralizado, mas que ainda permite que bibliotecas em outras linguagens de programação possam ser desenvolvidas em cima desta, como a *rclpy*, para *Python*, e a *rclcpp*, para *C++* [31]. Por fim, de modo a aproveitar a refatoração da plataforma, também foi adicionado o uso de recursos mais modernos de *Python3* e *C++*

de 2011, não disponíveis na primeira versão do ROS, além de suporte para plataforma *Windows*, *Mac* e *Linux*.

2.4.3.1 Implementação

A implementação do *driver* do protótipo pode ser ilustrada pelo diagrama da Figura 2.10, onde as figuras ovais representam, de forma sistemática, *nós* de processamento e as retangulares representam *tópicos*, onde informações são publicadas para serem consumidas por outros *nós*. Vale ressaltar o fluxo desejado do sistema de percepção com o uso de imagens estéreo para a determinação de profundidade, está representado com setas tracejadas, pois não foi implementado.

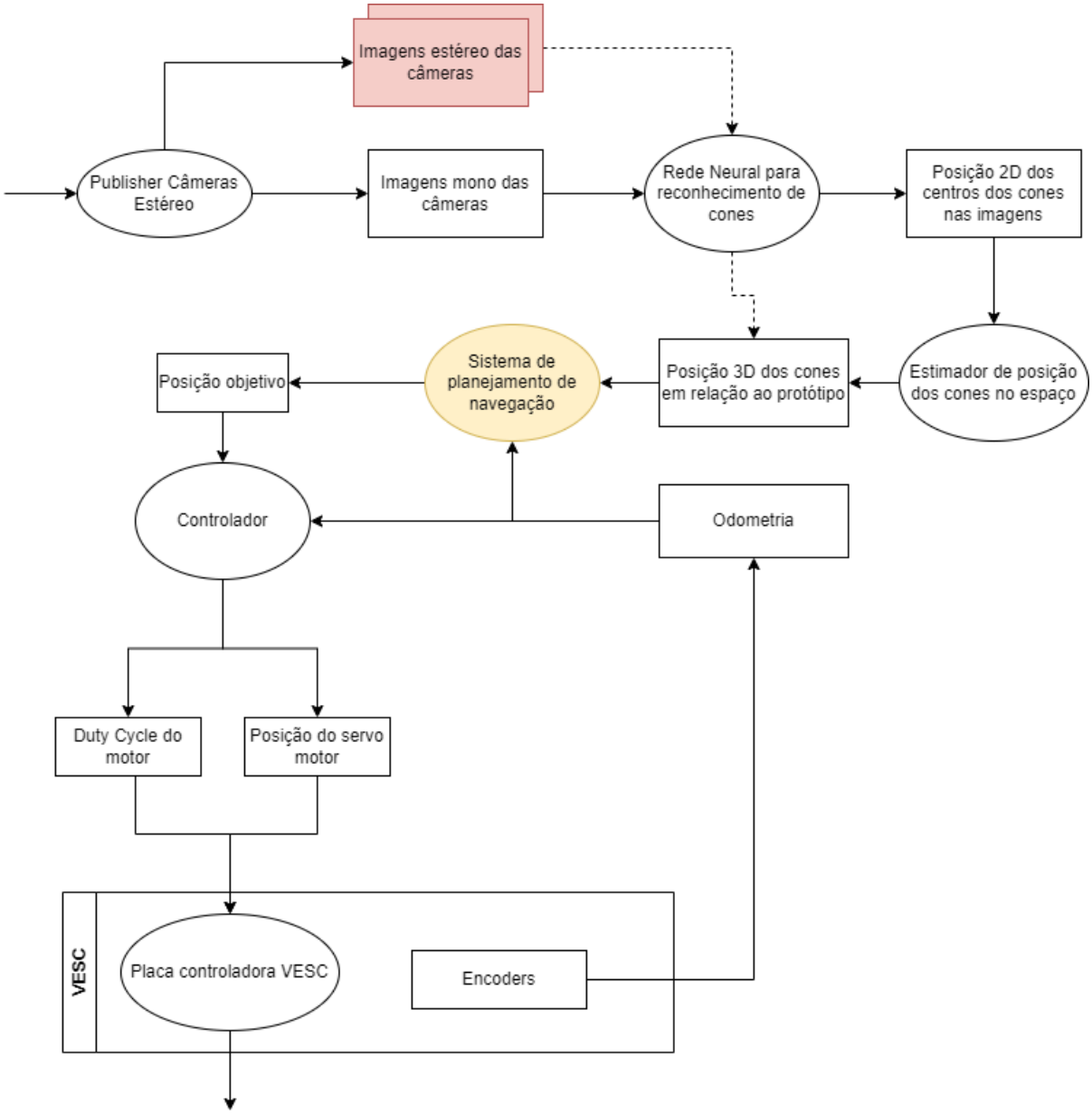


Figura 2.10: Estrutura de *nós* e *tópicos* do *driver* do protótipo do projeto legado

Em resumo, o *driver* comercial das câmeras captura as imagens de todas as 8 câmeras presentes no protótipo, e é feito um tratamento de fusão entre as capturas das câmeras esquerdas de cada módulo estéreo. Essa única imagem passa pela rede neural de detecção de objetos, que retorna como posição as coordenadas do centro dos cones na imagem. A partir desses dados, considerando a distância ao horizonte e a altura da câmera em relação ao chão, estima-se e publica-se a distância dos cones em relação ao protótipo. Em ambientes planos, essa aproximação se provou altamente efetiva. Indicado em amarelo na Figura 2.10 está o foco do trabalho a ser desenvolvido, que é o sistema de planejamento de navegação do modelo em ambiente simulado. Esse sistema simplesmente gerava trajetórias na direção de um cone capturado pelo sistema de percepção ao utilizar as informações

de posições do cone e do protótipo por odometria, enviada pela placa VESC. A cada instante, a posição alvo é consumida pelo *nó* de controle e, utilizando as informações de odometria, emite dados para controle lateral e longitudinal, executadas pela placa VESC.

2.5 Simulador *CARLA*

Para validação inicial do projeto legado [27], diversos simuladores foram avaliados para uso, tendo em vista seus requisitos. Dentre eles estão: o *CarSim*, o *SVL* e o *CARLA*. O simulador CarSim tem foco em oferecer uma física realista para seus usuários, mas seu acesso é feito através de uma licença proprietária. O *SVL*, por sua vez, teve seu desenvolvimento descontinuado recentemente. Por fim, está o *CARLA*, escolha final do projeto.

O uso do simulador *CARLA*, de código aberto, foi selecionado pelo projeto legado devido ao seu foco em realismo de imagem, uma vez que utiliza a tecnologia *Unreal Engine 4* para criação de cenários. Os cenários são utilizados para gerar entradas verossímeis para o sistema de detecção de objetos, um dos grandes focos do trabalho [28]. Outra facilidade do uso do *CARLA* é a existência da API para ROS 2 *carla-ros2-bridge* [32], que trivializa a comunicação entre o modelo desenvolvido e o *software*. Isso também colabora para a validação por meio do simulador, uma vez que mínimas alterações no modelo simulado são necessárias para aplicá-lo de forma embarcada. A Figura 2.11 ilustra o fluxo de dados no ambiente da simulação realizada pelo projeto que desenvolveu o protótipo.

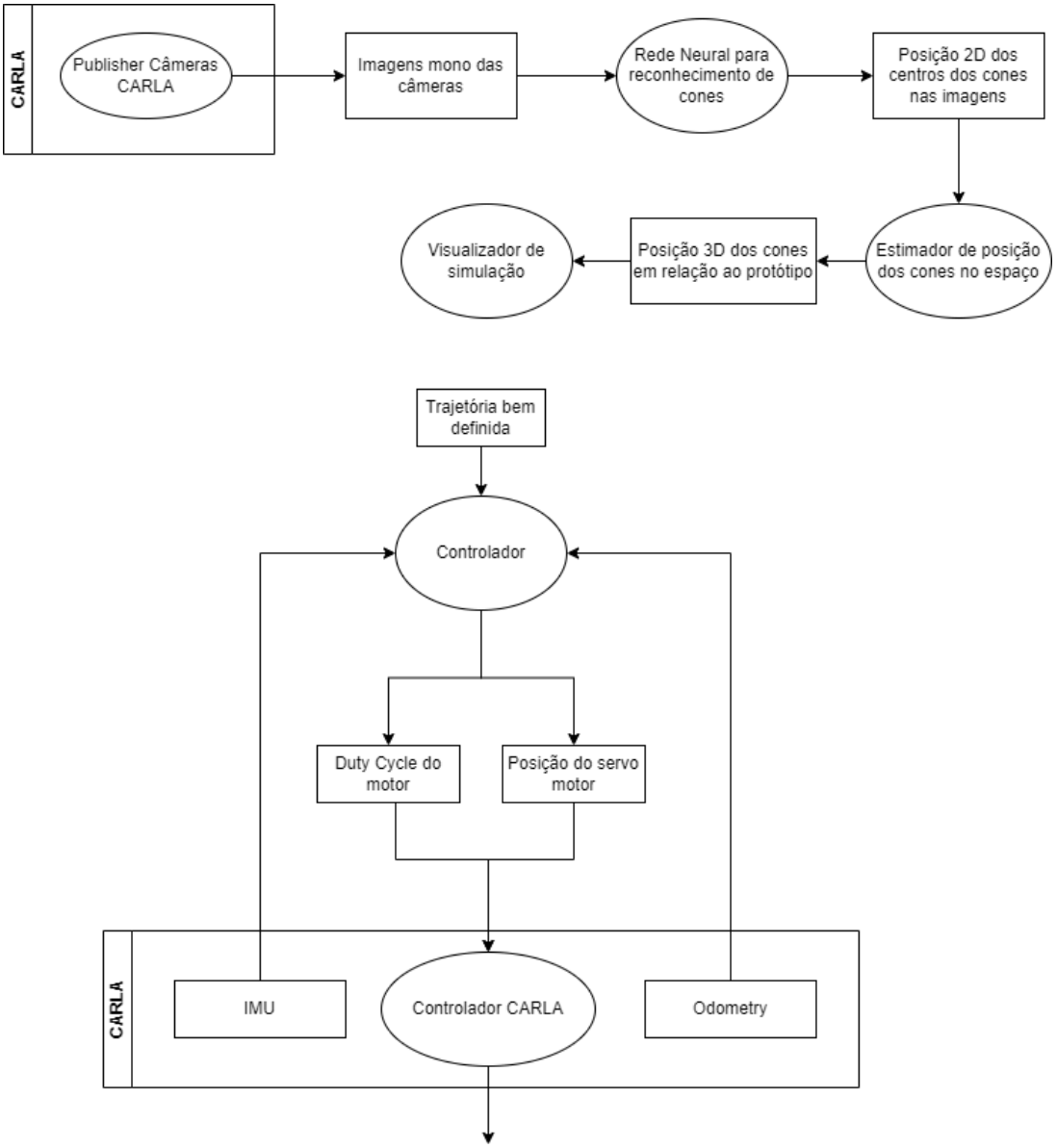


Figura 2.11: Estrutura de *nós* e *tópicos* da simulação feita com o software *CARLA* e *driver* do protótipo do projeto legado

Muito da estrutura do fluxo de dados do protótipo é mantida, sendo ressaltadas as seguintes diferenças: as interfaces com o simulador *CARLA*, que fornece as imagens das câmeras, de odometria e de IMU (do inglês Inertial Measurement Unit), e consome as informações de comando dos motores de controle longitudinal e lateral; e a ausência de um sistema de elaboração de trajetória, uma vez que ela foi gerada previamente para a navegação no mapa oferecido pelo simulador.

Para manter a continuidade do trabalho, o mesmo simulador *CARLA* será usado ao longo do desenvolvimento deste projeto.

3 ESPECIFICAÇÃO

3.1 Requisitos funcionais

Implementar SLAM com visão estéreo O software deve ser capaz de mapear os obstáculos do ambiente por meio de dados de visão estéreo, e localizar o veículo neste espaço.

Definir trajetória O software deve ser capaz de receber as coordenadas de destino, calcular a navegar o veículo pela trajetória ideal até o destino, levando em consideração as condições do ambiente, como a presença de obstáculos, e as limitações do próprio veículo.

Realizar manobra de estacionamento inclinado a 45° O software deve ser capaz de executar uma manobra de estacionamento em vagas inclinadas a 45 graus, garantindo que o veículo autônomo possa se posicionar corretamente em relação à inclinação da vaga. Isso inclui a coordenação precisa dos movimentos do veículo, como direção, aceleração e frenagem, para realizar a manobra de baliza inclinada com segurança, seguindo as marcações específicas para esse tipo de estacionamento.

Realizar manobra de estacionamento paralela O software deve ser capaz de executar uma manobra de estacionamento em vagas paralelas à via, garantindo que o veículo autônomo possa se posicionar corretamente em relação aos veículos ao redor. Isso inclui a coordenação precisa dos movimentos do veículo, como direção, aceleração e frenagem, para realizar a manobra de baliza com segurança.

Garantir a segurança do veículo O sistema deve realizar toda a navegação e manobras sem choque com obstáculos, preservando a integridade do automóvel e seus passageiros.

3.2 Requisitos não funcionais

Escalabilidade O software deve ser escalável, permitindo a expansão e adição de novas funcionalidades no futuro.

Auditabilidade O sistema deve ser facilmente validável, possibilitando testes e verificações para garantir sua eficácia e segurança, incluindo a legibilidade de seus sistemas de navegação.

Modularidade O software deve ser modular, permitindo a substituição de subárvores e ramificações, facilitando a manutenção e evolução do sistema.

Eficiência O sistema deve executar suas operações de forma ágil e responsiva, possibilitando uma resposta rápida mesmo em situações complexas de análise. Isso se faz necessário em um ambiente onde o tempo de reação é fator primordial.

Robustez O sistema deve ser robusto o suficiente para lidar com possíveis falhas e exceções, mantendo um desempenho consistente e retornando a um estado seguro em caso de erros inesperados. Isso garante a confiabilidade e estabilidade do sistema durante sua operação.

Desempenho O sistema de planejamento e decisão deve ser capaz de processar de forma eficiente e rápida as informações provenientes dos sensores, bem como analisar e tomar decisões complexas em tempo real. Ele deve garantir uma resposta ágil e precisa mesmo em situações de análise e processamento intensivo, a fim de evitar atrasos significativos que possam comprometer a segurança e o tempo de reação do veículo autônomo.

3.3 Casos de Uso

3.3.1 Navegar para destino escolhido

Breve descrição Processo pelo qual o passageiro interage com o sistema para selecionar o destino desejado para o veículo autônomo.

Atores Passageiro

Pré-condições Sistemas de mapeamento, localização, planejamento e decisão em execuçã e ambiente previamente mapeado por algoritmo SLAM.

Fluxo principal

1. O usuário inicia a interação com o sistema.
2. O sistema exibe um mapa do ambiente para escolher o destino.
3. O passageiro seleciona o destino desejado.
4. O sistema registra a seleção do destino escolhido pelo usuário.
5. O sistema inicia o processo de planejamento da rota com base no destino escolhido, considerando as limitações de movimento do veículo.
6. A trajetória é executada.

Pós-condições Veículo posicionado no destino e mapeamento do novo ambiente em torno do veículo realizado durante o percurso é salvo em sua memória interna.

3.3.2 Estacionar em vaga inclinada a 45°

Breve descrição O veículo autônomo realiza a manobra de estacionamento em um local designado para estacionamento inclinado a 45 graus.

Atores Passageiro

Pré-condições Sistemas de mapeamento, planejamento e decisão em execução, identificação dos marcadores específicos para vagas inclinadas a 45 graus e mapeamento da região próxima.

Fluxo principal

1. O passageiro seleciona a função de estacionar o veículo autônomo em vagas a 45 graus.
2. O veículo autônomo identifica um local disponível com a inclinação adequada para estacionamento inclinado.
3. O sistema realiza a análise do espaço disponível para verificar se é apropriado para a manobra de estacionamento a 45 graus.
4. O veículo se posiciona de maneira diagonal à vaga, alinhando-se com a inclinação da mesma.
5. Executa a rotina de manobra ideal.
6. O sistema monitora o progresso da manobra de estacionamento para garantir a segurança e evitar colisões.

Pós-condições O veículo autônomo está estacionado na vaga inclinada a 45 graus, seguindo as marcações específicas para esse tipo de estacionamento.

3.3.3 Estacionar em vaga paralela

Breve descrição O veículo autônomo realiza a manobra de estacionamento em um local disponível para estacionamento paralelo.

Atores Passageiro

Pré-condições Sistemas de mapeamento, planejamento e decisão em execução, identificação dos marcadores específicos para vagas paralela e mapeamento da região próxima.

Fluxo principal

1. O passageiro seleciona a função de estacionar o veículo autônomo.
2. O veículo autônomo identifica um local disponível para estacionamento paralelo.
3. O sistema realiza a análise do espaço disponível para verificar se é adequado para a manobra de estacionamento.
4. O veículo se posiciona paralelo ao veículo da frente e bem próximo.
5. Executa a rotina de baliza.
6. O sistema monitora o progresso da manobra de estacionamento para garantir a segurança e evitar colisões.

Pós-condições O veículo autônomo está estacionado paralelamente à via no local identificado.

4 DESENVOLVIMENTO

O desenvolvimento do sistema seguirá as etapas exibidas no fluxograma da Figura 4.1.

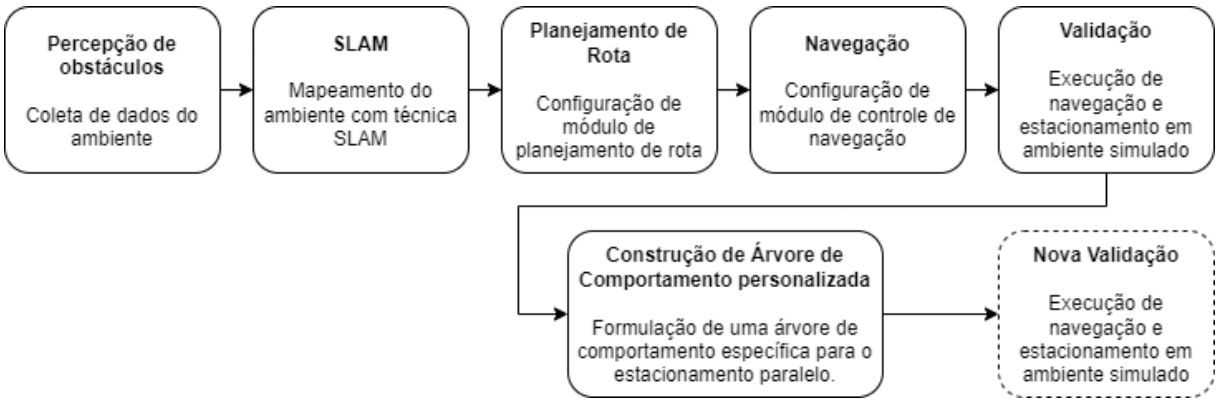


Figura 4.1: Sequência de etapas para confecção do sistema de planejamento e decisão do modelo simulado

4.1 Percepção de obstáculos

Utilizando o simulador *CARLA*, busca-se emular a configuração de câmeras do protótipo físico, gerando dados RGBD (do inglês, Red Green Blue Depth, ou Vermelho Verde Azul Profundidade) de imagens, tal como é feito pelas câmeras do protótipo físico. Por meio desses dados, é possível construir uma nuvem de pontos RGB no espaço em torno do veículo em tempo real. Esses dados são essenciais para a identificação de obstáculos e elaboração do mapa do ambiente, utilizado pela *Navigation2* no planejamento da rota do veículo. A Figura 4.2 exibe as nuvens de pontos geradas pelas câmeras RGBD do modelo simulado.

Figura 4.2: Dados RGBD de obstáculos da cena

4.2 SLAM

Para navegação plana em duas dimensões, a aquisição de dados de obstáculos deve ser em 2D. Tanto em simulação quanto no protótipo físico, são utilizados sensores que geram nuvens de pontos em três dimensões, como as câmeras RGBD e o LIDAR 3D (do inglês Light Detection and Ranging). Uma transformação utilizando uma biblioteca de código aberto [33] de nuvem de pontos para LIDAR 2D foi implementada, tomando como leitura o dado de obstáculo mais próximo do veículo dentro de sua faixa de altura. Para maior performance no hardware disponível, em simulação a nuvem de pontos utilizada para essa transformação provém do LIDAR 3D, que gera o mesmo tipo de dado que as câmeras RGBD, mas não necessitam de *nós* de tratamento de imagens em execução.

Com os dados da percepção e por meio dos recursos da biblioteca, é aplicada a técnica de SLAM para a construção e incrementação dinâmica do mapa do ambiente.

A biblioteca Navigation2 oferece ferramentas para tal finalidade, como o pacote *slam_toolbox*. Foi utilizada a implementação de SLAM *online*, ou seja, com dados de percepção em tempo real, e *assíncrono*, que evita atrasos de processamento ao ignorar e não processar todos os conjuntos de dados no *buffer*, priorizando os mais recentes.

4.3 Planejamento de Rota

A navegação autônoma por meio da biblioteca Nav2 apresenta um desafio notável quando se trata de veículos que seguem a Geometria de Ackermann, como carros convencionais. A maioria dos robôs comerciais que implementam a solução Nav2 são holonômicos, o que significa que podem mover-se em qualquer direção de forma independente. Veículos com direção padão Ackermann possuem uma limitação de movimento lateral, e é necessário traduzir essas limitações para os algoritmos de planejamento de rota e controle na configuração de uso dos recursos da *Nav2*, por meio da definição de um raio de curvatura factível mínimo para o modelo [34]. Empiricamente, o valor escolhido de raio mínimo de curvatura para o nosso modelo é de 3 metros.

Para realizar a navegação por meio da biblioteca Nav2, é preciso configurar ao menos um módulo de planejamento global, ou planejador global, do inglês *global planner*. Tais módulos são sub-árvores que se acoplam à árvore base de navegação da biblioteca. O módulo escolhido para implementação no modelo foi o chamado Smac-Hybrid A*, que é uma implementação do popular algoritmo de planejamento de rotas A* para ROS2, mas que considera as restrições cinemáticas do modelo em questão (*hybrid*) [35]. A* aplica propagação computacional direcionada ao destino de modo a encontrar mais rapidamente a rota mais curta até ele quando comparado à propagação simples de seu antecessor, o algoritmo de Dijikstra [36]. Tal algoritmo é compatível com a navegação padrão Ackermann e fornece configurações para limitar o número de rotas possíveis de acordo com as limitações cinemáticas do veículo.

O algoritmo atribui a cada seção do espaço mapeado um custo de navegação, proporcional à dificuldade de navegar no terreno. Regiões próximas a obstáculos possuem maior custo, enquanto regiões distantes possuem menor custo. É possível configurar pesos para alteração de custos designados a regiões que requerem alto ângulo de direção ou uso da marcha ré para priorizar rotas mais eficientes do ponto de vista de conforto. A este mapa de custo damos o nome de *global costmap*.

4.4 Navegação

Além do módulo de planejamento global, o sistema de navegação também necessita de um planejador local (do inglês *local planner*), ou controlador, que recebe a trajetória do planejador global e a executa, realizando ajustes de acordo com um *costmap* local, o qual, quando comparado com o *costmap* global, apresenta tamanho reduzido mas maior

taxa de atualização, para permitir reação a obstáculos dinâmicos. Este também deve considerar as limitações cinemáticas do modelo, de modo a enviar comandos factíveis de navegação aos sistemas de aceleração e direção.

O módulo escolhido é chamado de *Regulated Pure Pursuit* (RPP) [37] que emprega a simples perseguição de um ponto futuro da trajetória, considerando as limitações cinemáticas de curvatura do modelo e aplicando reduções de velocidade linear ao aproximar-se de curvas, obstáculos ou do fim da rota planejada. A faixa de velocidades lineares permitidas, o grau de agressividade dessa variação e a distância do ponto futuro perseguido com relação ao veículo são todos valores reguláveis para cada aplicação do módulo. Tal controlador não fornece adaptação de rota a obstáculos dinâmicos, tal qual outras opções como o *Model Predictive Path Integral Controller* (MPPI) [38]. No entanto, quando pareado com um planejador global capaz de ser executado em alta frequência, tal propriedade pode ser obtida.

4.5 Preparação do ambiente e validação

De modo a usar um ambiente simulado para o desenvolvimento e validação do projeto a ser desenvolvido, primeiro é preciso preparar e configurar as ferramentas a serem utilizadas. Isto em vista, selecionamos como mapa base o *Town02*, disponível no conjunto de mapas base do CARLA, para dar continuidade ao projeto em simulação. Para delimitar o espaço das vagas para o sistema de estacionamento, foi utilizado o modelo 3D de cone de construção, também disponível na biblioteca de recursos base do CARLA, que se assemelha aos cones que a rede neural de percepção do projeto legado em [27] está treinada a detectar. Com um *script* em *Python* foi possível adicionar e posicionar de forma adequada os cones no mapa, utilizando a API do simulador. A vaga delimitada é exibida na Figura 4.3

Figura 4.3: Cones posicionados em simulação para delimitar uma vaga

Tanto a navegação livre como as rotinas de manobras serão validadas em ambiente simulado. Espera-se mapear o mapa *Town02* do simulador *CARLA* e efetuar navegação e manobras de estacionamento de forma satisfatória e segura.

4.6 Árvore de Comportamento personalizada para estacionamento paralelo

Os resultados de navegação descritos na Seção 5.4 indicaram a necessidade de construir uma Árvore de Comportamento específica e personalizada para execução de uma tarefa precisa e com alta proximidade a obstáculos como o estacionamento em uma vaga paralela.

A elaboração da Árvore de Comportamento para a execução de tal tarefa foi um processo evolutivo e sistemático. Inicialmente, foi elaborada uma árvore para a rotina de estacionamento a partir de uma posição inicial do veículo definida em relação à vaga, representada pelos 4 cones. Essa árvore engloba etapas sequenciais e possíveis correções de posição, atuantes no controle direto do veículo, cada uma delas com suas respectivas condições de sucesso e falha. Em seguida, foi desenvolvida uma árvore de segurança que leva em consideração as condições de falha, monitoradas continuamente antes da árvore principal, subárvore desta. Por fim, mais próxima da raiz global, foi desenvolvida uma árvore que conduz o veículo à posição inicial da vaga utilizando a sub-árvore de navegação padrão da *Navigation2*, que utiliza dos planejadores e controladores configurados. Assim

que a vaga é identificada pelos 4 cones, o veículo é direcionado para a *pose* inicial correta. Esse processo progressivo e a hierarquia planejada devem assegurar uma execução robusta e confiável da tarefa de estacionamento.

5 RESULTADOS

5.1 Percepção de obstáculos

A Figura 5.1 mostra, em azul e roxo, os dados provenientes do LIDAR 3D e em branco, os dados filtrados, como um LIDAR 2D, com a faixa de percepção configurada a uma altura que ignora a guia da calçada. Com ajustes de configuração, também foi possível detectá-la como obstáculo, como mostra a Figura 5.2.

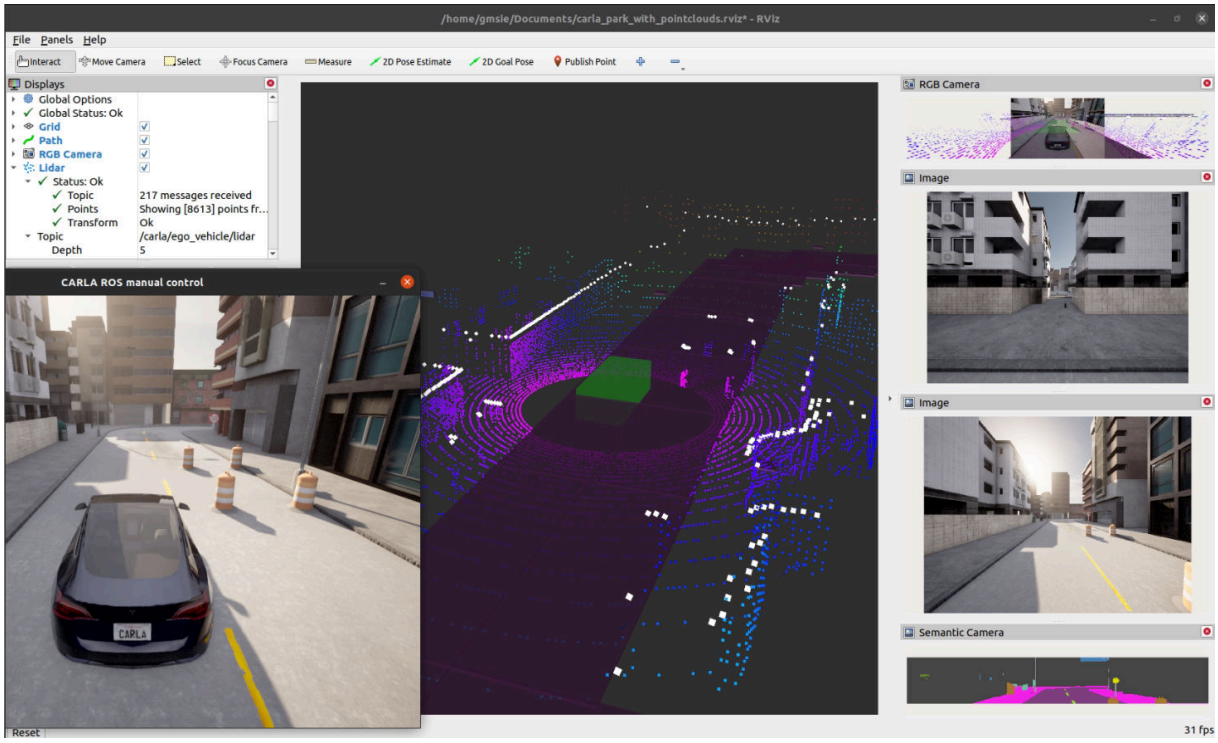


Figura 5.1: Transformação de dados de LIDAR 3D para LIDAR 2D

5.2 SLAM

O modo assíncrono de execução do algoritmo mostrou desempenho mais satisfatório que o modo síncrono, que apresentou latência consideravelmente superior e imprecisão na

localização do veículo devido a atrasos de processamento. A Figura 5.2 apresenta uma representação visual do algoritmo em ação, onde o veículo é controlado manualmente para realizar o mapeamento inicial do ambiente.

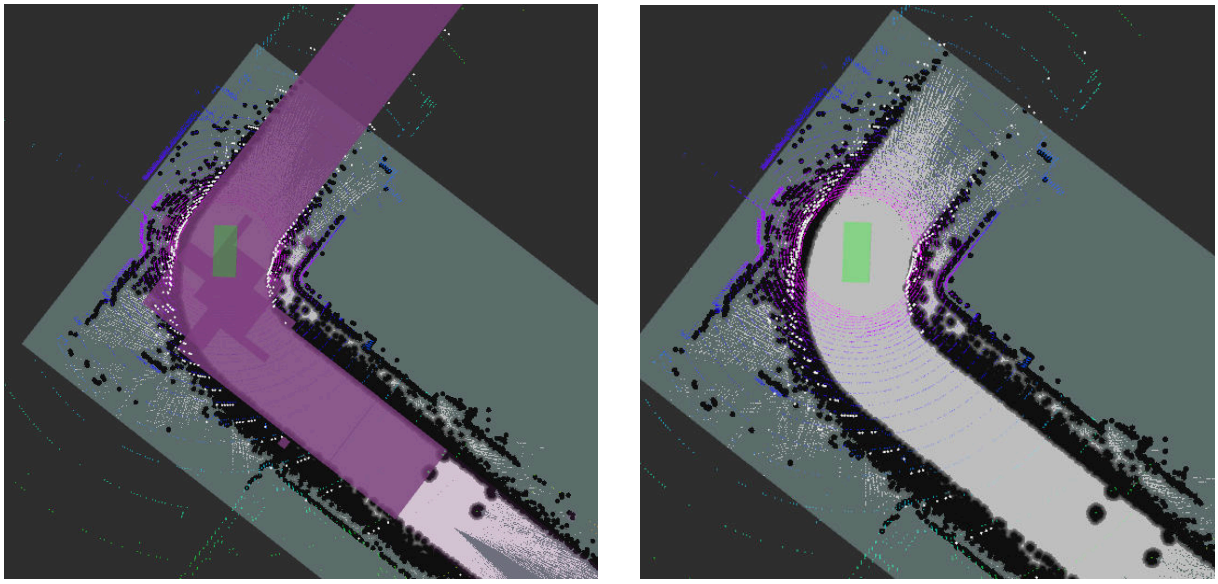


Figura 5.2: Algoritmo SLAM em funcionamento

Após percorrer uma maior extensão do ambiente, o mapa construído e extraído pode ser visto na Figura 5.3.

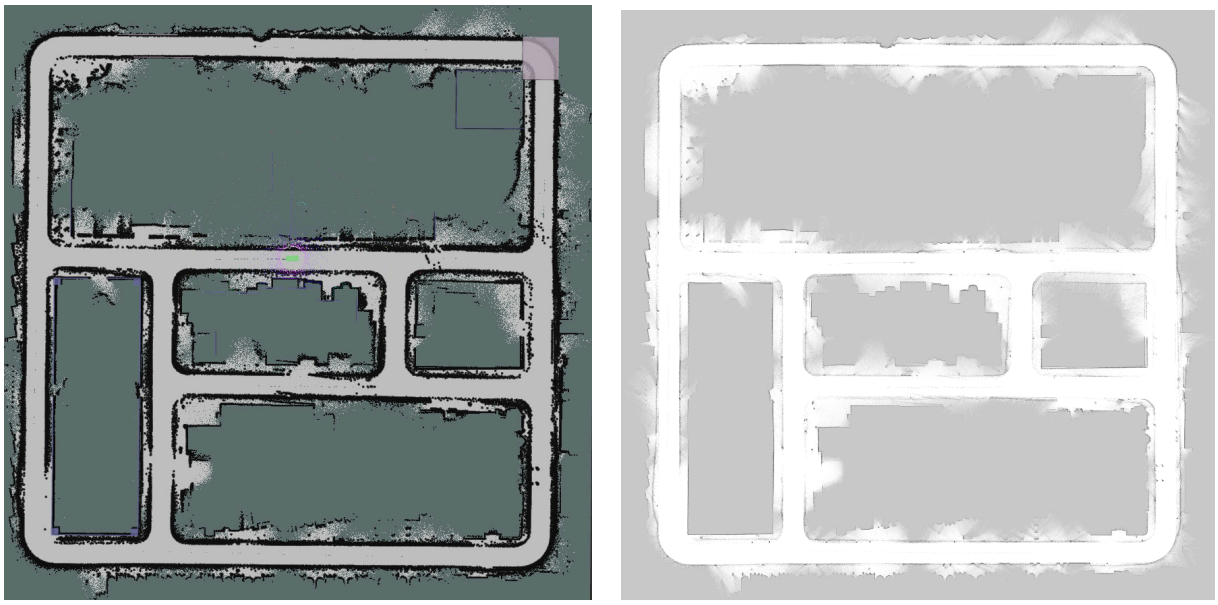


Figura 5.3: Mapa da Town02 base construído pelo algoritmo SLAM

5.3 Planejamento de rota

A animação da Figura 5.4[1] exibe o planejamento de rota realizado pelo algoritmo Smac-Hybrid A*, que leva em consideração as limitações de movimento do veículo para a geração de uma rota factível para o modelo em questão.

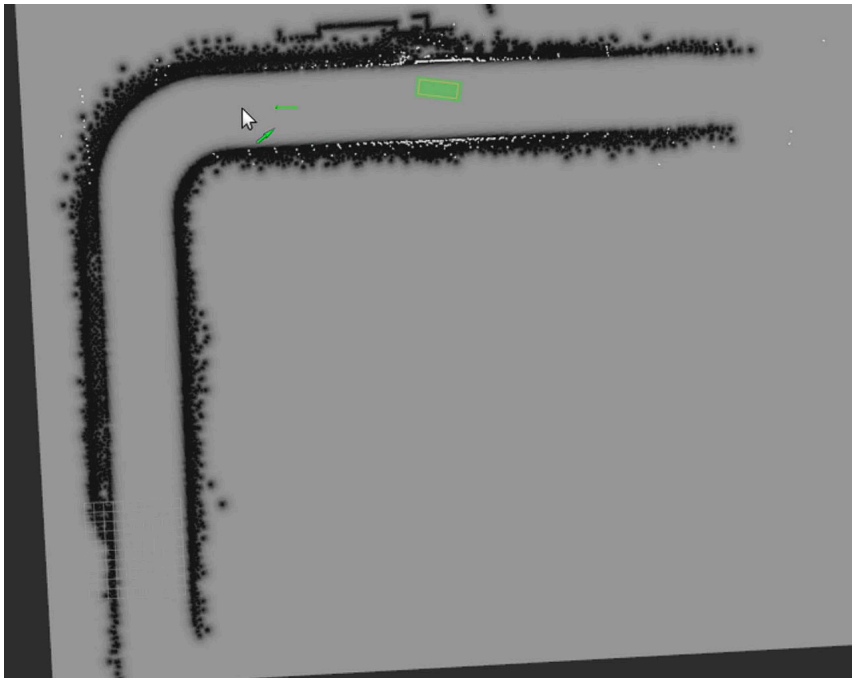


Figura 5.4: Planejamento de Rota por algoritmo Smac-Hybrid A*

5.4 Navegação

Foram registrados 5 cenários de navegação: A realização de uma conversão para a direita, o estacionamento do veículo em uma vaga inclinada a 45 graus, a tentativa de estacionamento em uma vaga paralela de frente, de ré e em vaga de tamanho reduzido. Esses cenários são ilustrados nas animações das Figuras 5.5, 5.6, 5.7, 5.8, e 5.9 respectivamente.

[1]Para visualizar a animação, utilize um leitor de PDF compatível, como o Adobe Acrobat Reader, para Windows, ou Okular, para Linux.

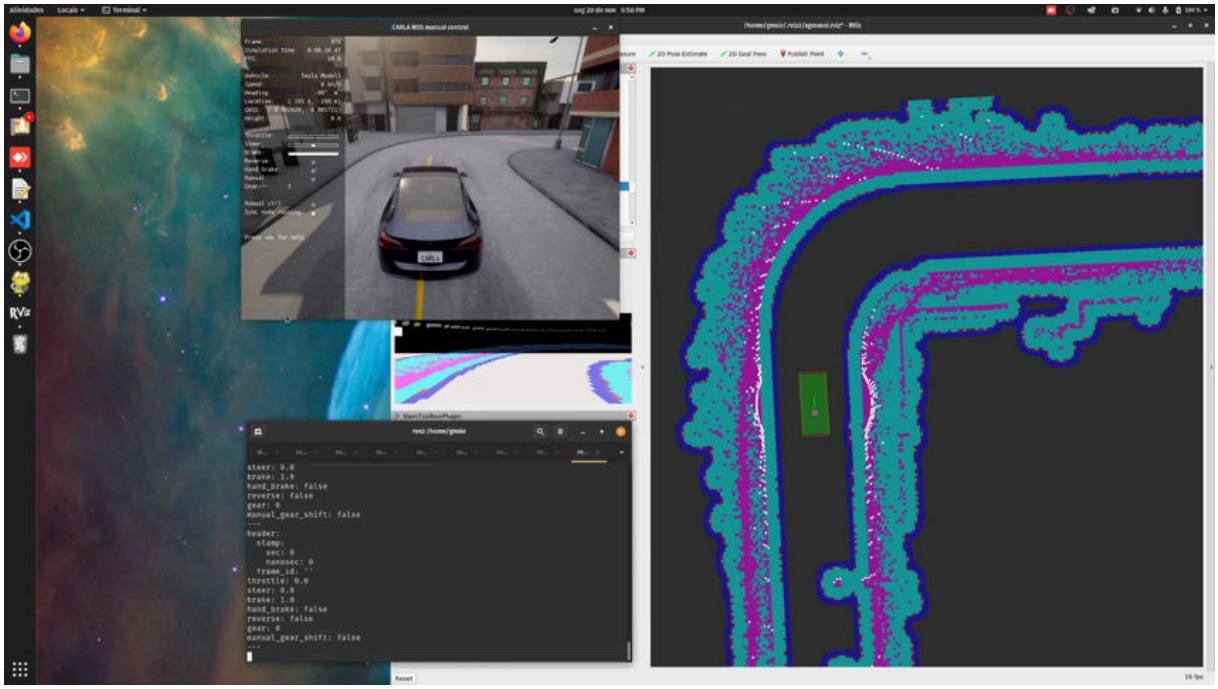


Figura 5.5: Execução de rota curvilínea com controlador RPP

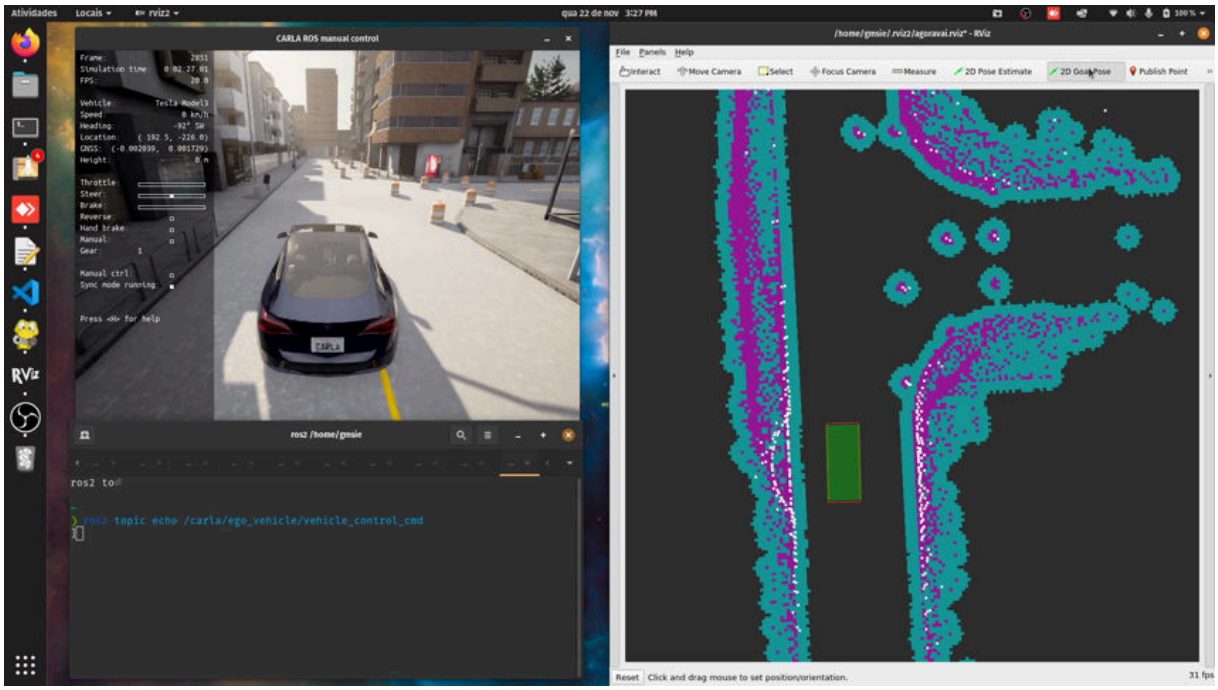


Figura 5.6: Execução de rota de estacionamento em 45º e controlador RPP

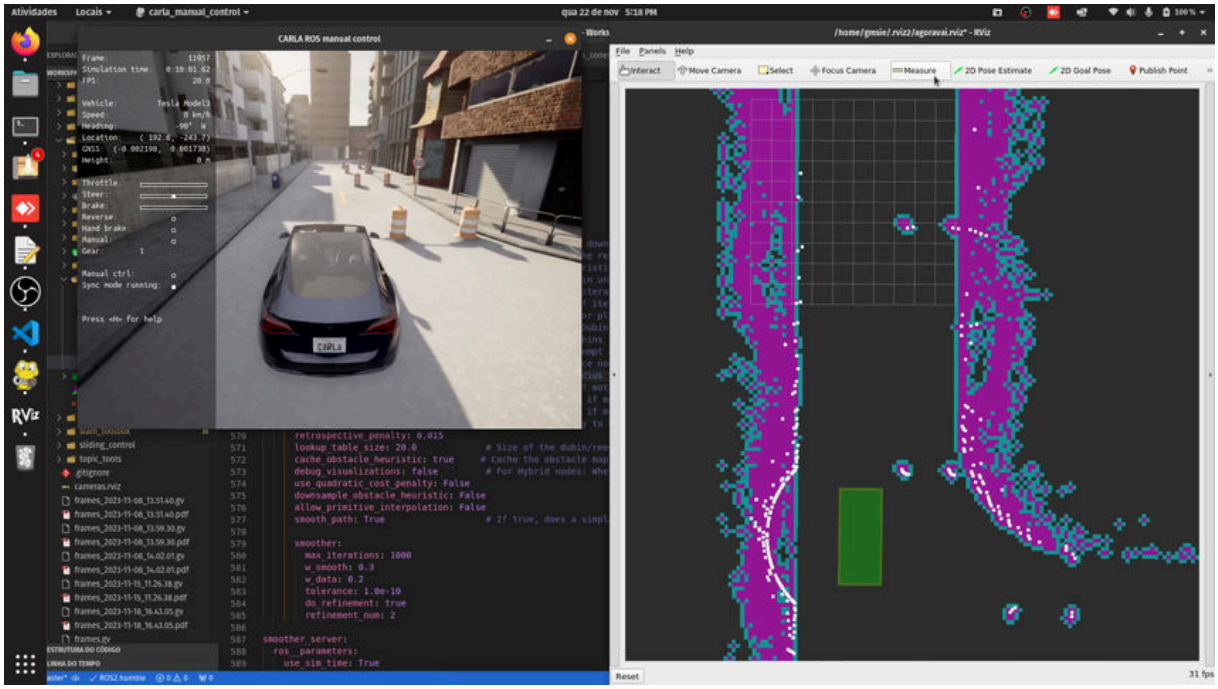


Figura 5.7: Execução de rota de estacionamento paralelo de frente com controlador RPP

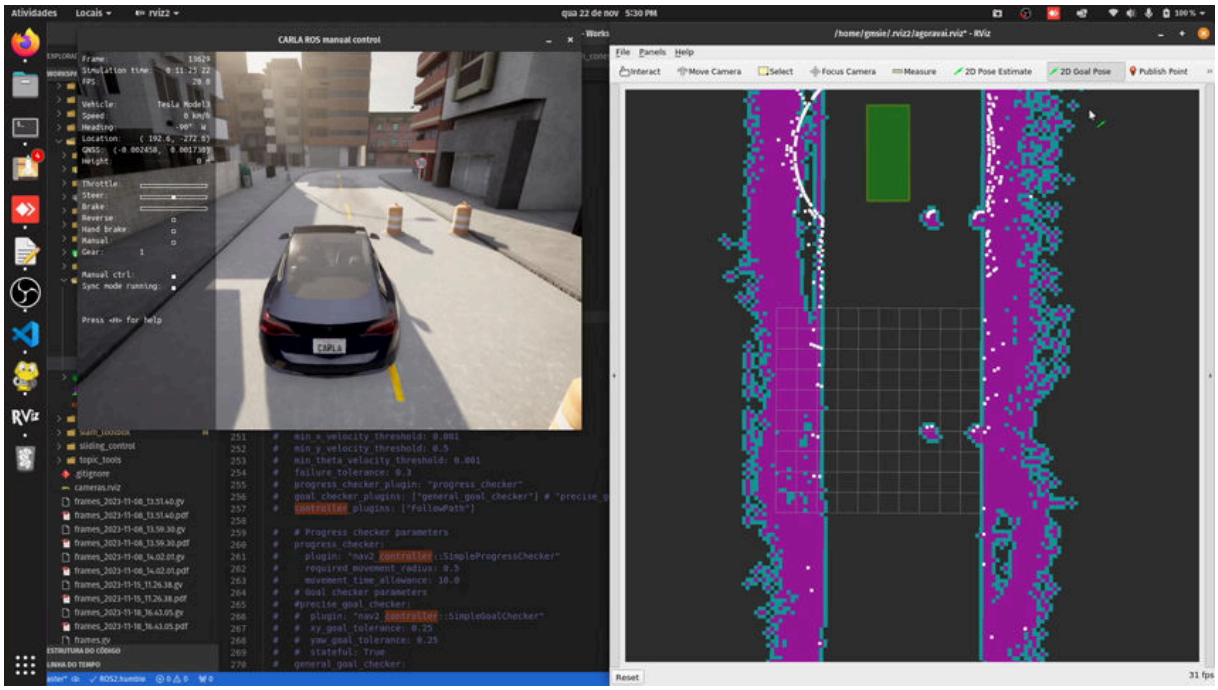


Figura 5.8: Execução de rota de estacionamento paralelo de ré com controlador RPP

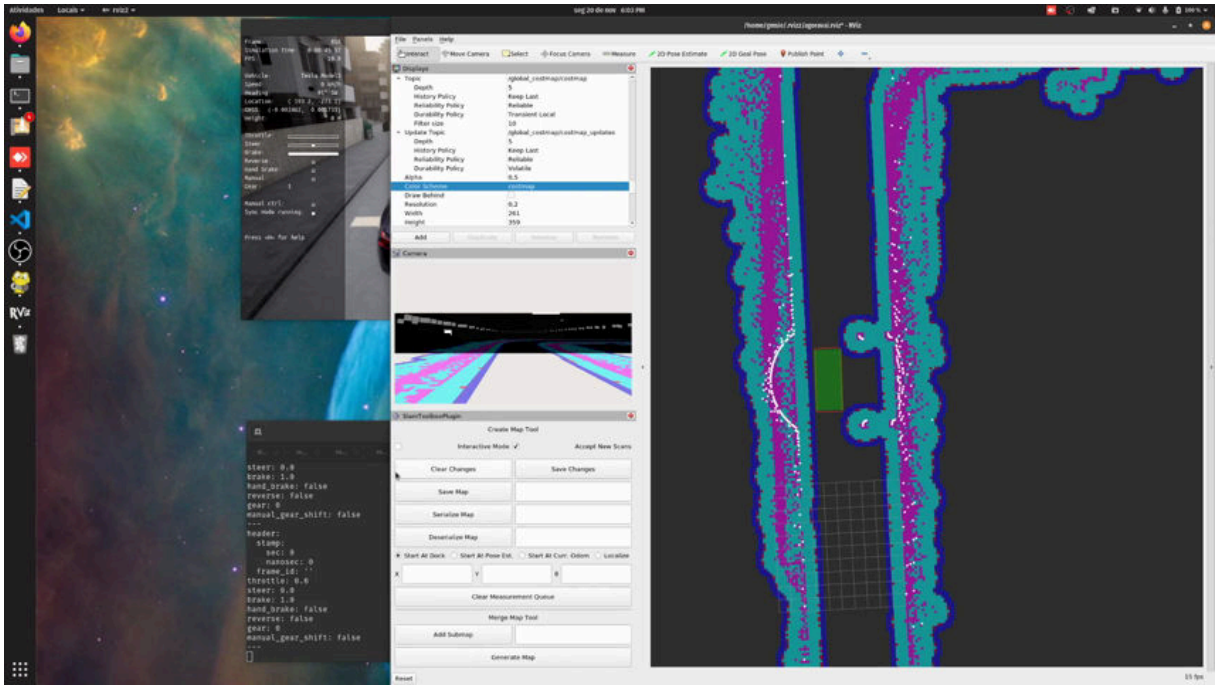


Figura 5.9: Criação de rota e tentativa de estacionamento paralelo com controlador RPP

As Figuras 5.7, 5.8, e 5.9 exemplificam a dificuldade tanto do planejador global Smac-Hybrid A* quanto do controlador RPP de executar manobras curtas com alta proximidade a obstáculos. Para realizar a tarefa, julga-se necessário empregar uma árvore de comportamento específica, a qual é detalhada na Seção 5.5.

5.5 Árvore de Comportamento personalizada para estacionamento paralelo

No topo da hierarquia global da Árvore de Comportamento para o estacionamento paralelo encontra-se uma árvore que consiste apenas em uma sequência contendo dois nós principais: a sub-árvore *NavigateToPose* da biblioteca *Navigation2*, que direciona o carro para a posição inicial, paralelo à vaga com a traseira do veículo alinhada à frente dela, e a sub-árvore desenvolvida, identificada como "RotinaComSeguranca2", como exibe a Figura 5.10.

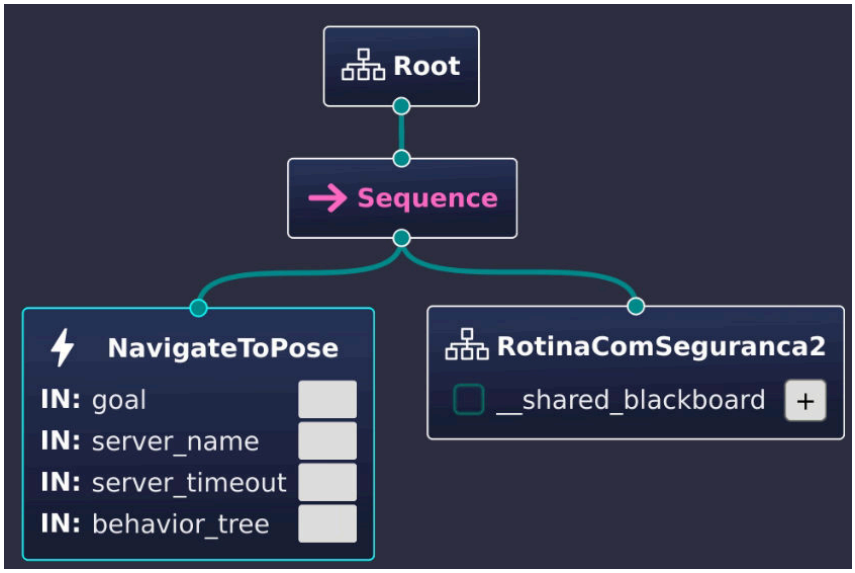


Figura 5.10: Rotina global de estacionamento

A sub-árvore de segurança é responsável por testar possíveis condições de falha globais durante toda a execução da rotina principal de estacionamento, indicada pela sub-árvore "RotinaEstacionamento"exibida na Figura 5.11. A segurança é garantida uma vez que se testa todas as possíveis falhas, representadas pelos nós condicionais filhos do nó de *fallback* antes de executar a rotina de estacionamento. Caso alguma condição retorne *SUCCESS*, que indicanda uma falha, a ação de parada é executada.

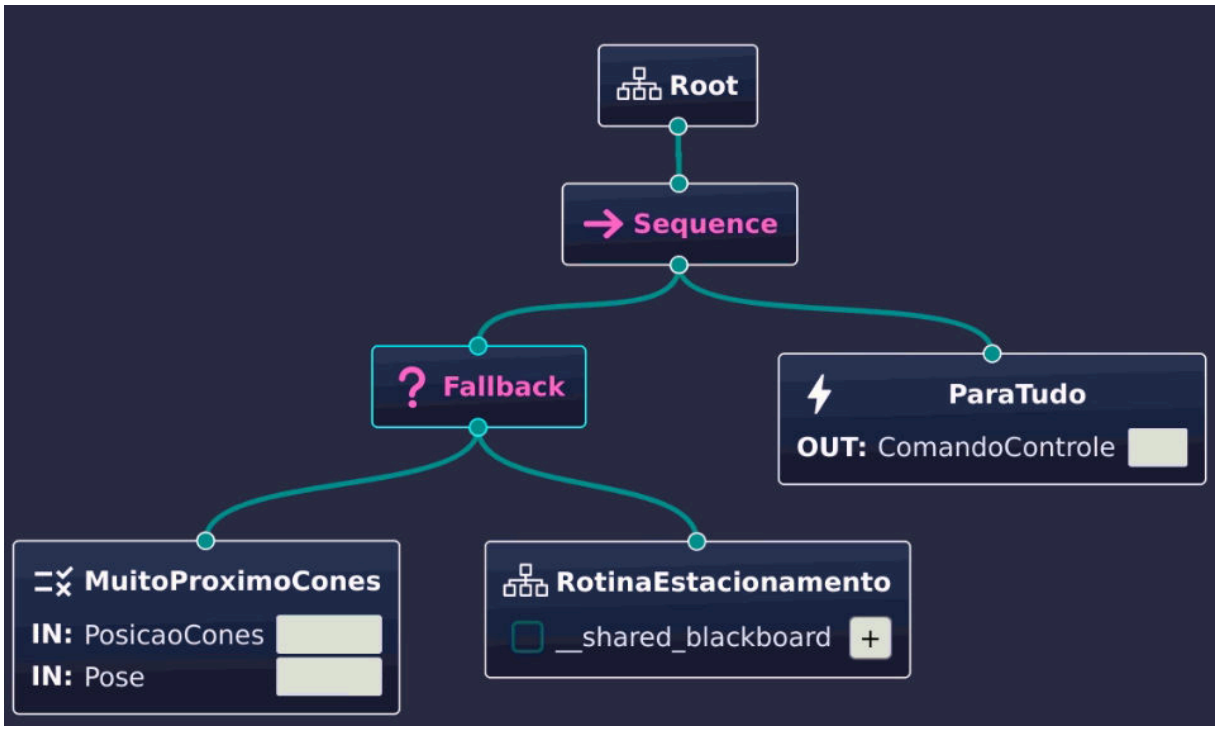
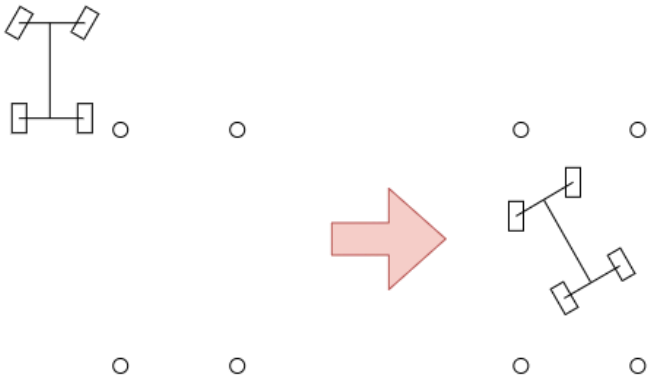
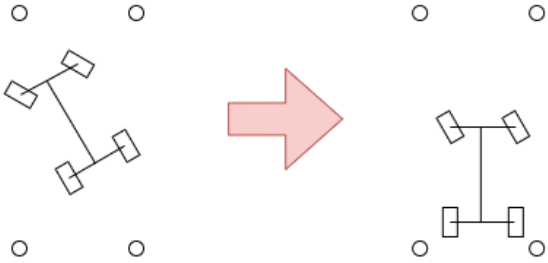


Figura 5.11: Árvore de segurança

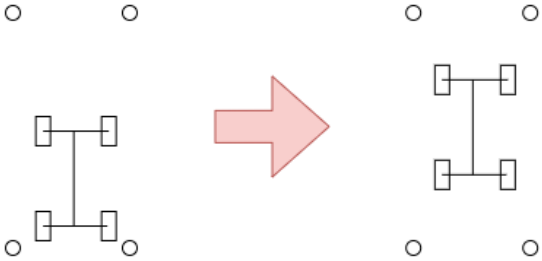
A subárvore "RotinaEstacionamento"inicia verificando se a posição veículo corresponde à posição desejada para início da rotina. A rotina é composta de quatro etapas, cada uma com sua condição de sucesso, ambas sob o mesmo nó de *fallback*. Para uma vaga localizada à direita da pista, uma vez que o veículo está posicionado no ponto de partida, a Etapa 1, indicada pela Figura 5.12a, consiste em realizar uma manobra de ré com o volante virado para a direita, enquanto o nó de condição monitora se o veículo chegou próximo ao centro da vaga. Uma vez satisfeita a condição, sai da sub-árvore de *fallback* do conjunto e se inicia a Etapa 2, representada pela Figura 5.12b, que mantém a ré, mas agora com o volante voltado para a esquerda. A condição de saída do *fallback* nessa etapa é se o veículo atinge a posição paralela à vaga. Se essa condição é atendida, o processo passa para a Etapa 3, ilustrada pela Figura 5.12c, na qual apenas há o ajuste do veículo em linha reta até o centro da vaga. Se caso, durante a segunda etapa, o veículo se aproxime da parte traseira da vaga antes de se orientar paralelamente à mesma, é preciso acionar a Etapa 4, mostrada na Figura 5.12d, na qual se faz o ajuste do veículo indo para frente com o volante para direita. Nessa etapa, a condição monitorada é novamente se o veículo ficou paralelo à vaga. Da mesma forma, se a condição for aceita, passa para Etapa 3 na qual há o ajuste ao centro da vaga. Caso contrário, se interrompe a manobra, visto que não é possível ou é muito arriscado estacionar no local.



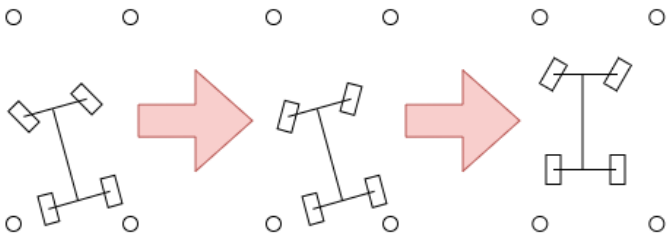
(a) Etapa 1



(b) Etapa 2



(c) Etapa 3



(d) Etapa 4

Figura 5.12: Ilustração de etapas da rotina de estacionamento

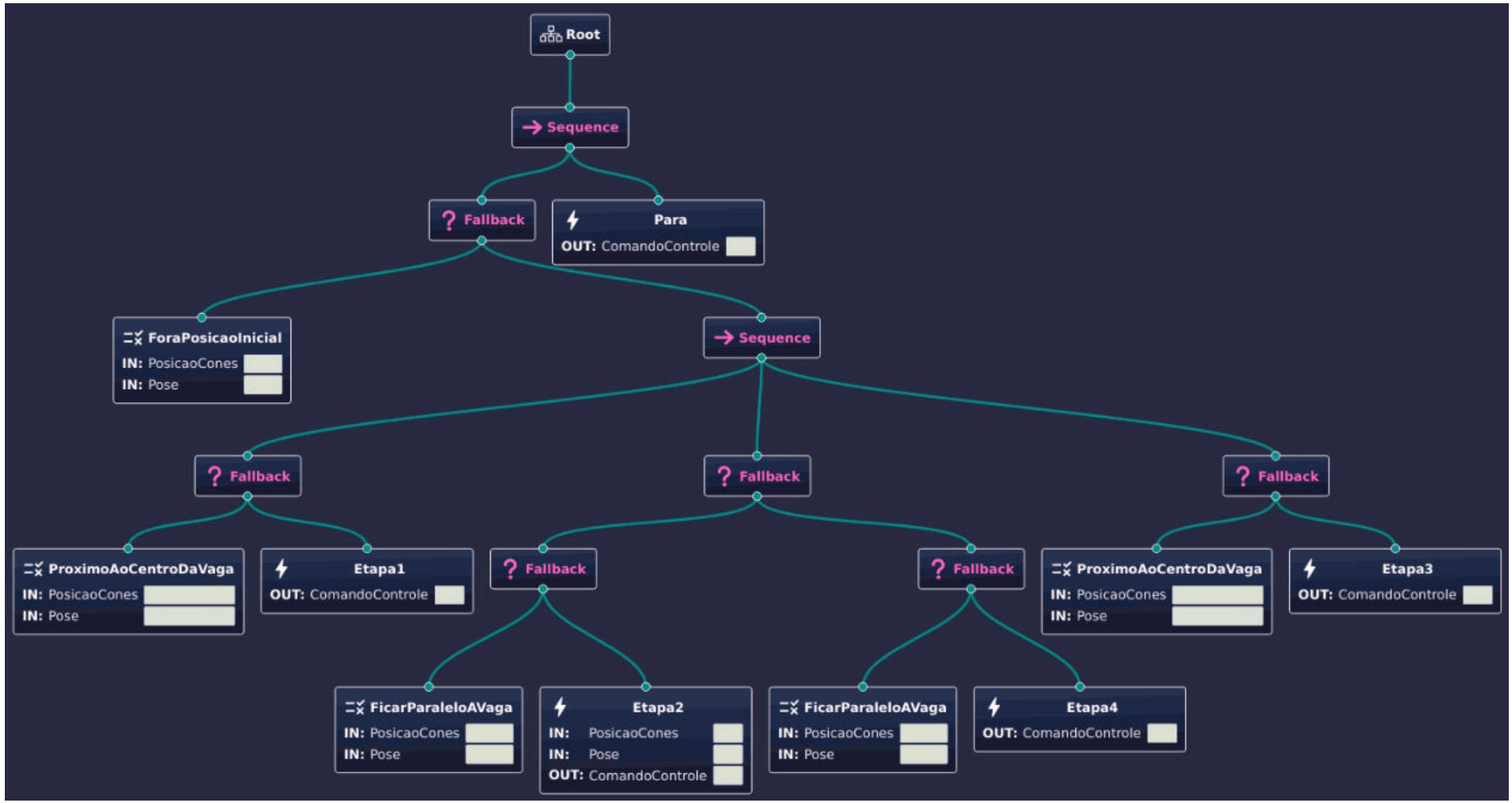


Figura 5.13: Rotina principal de estacionamento

6 DISCUSSÕES

6.1 Percepção de obstáculos

Na Figura 6.1 Nota-se uma limitação da percepção por LIDAR 3D do simulador devido à altura do veículo, que impede a detecção de obstáculos muito próximos dele. Tal imprecisão pode causar problemas no planejamento de rota local, a curto prazo, para obstáculos dinâmicos que ainda não possuem histórico registrado no mapa, mas pode ser solucionada no protótipo físico inclinando suas câmeras para baixo, de modo a detectar o ambiente o mais próximo possível do veículo.

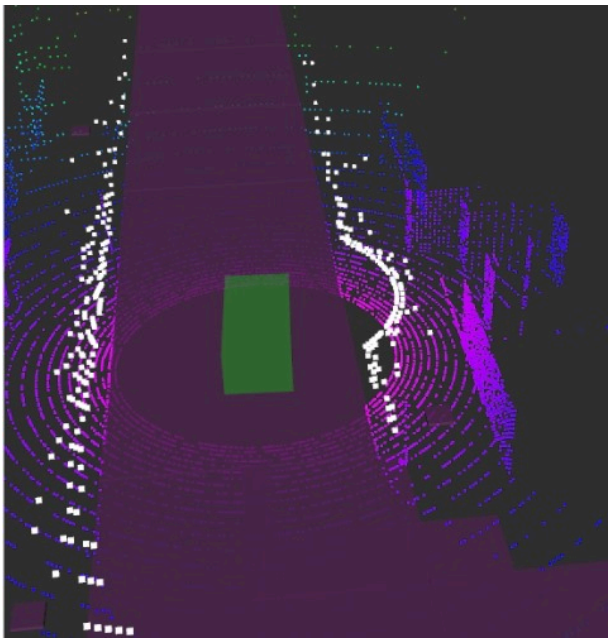


Figura 6.1: Detecção da guia da calçada

Com o objetivo de aproximar a simulação do protótipo físico da plataforma legada, é possível empregar as câmeras de profundidade do simulador *CARLA* em substituição ao LIDAR 3D como implementado. Nesse contexto, destaca-se a necessidade de fundir as nuvens de pontos provenientes das quatro câmeras de acordo com suas posições relativas ao modelo, e alimentar o dado resultante ao algoritmo de conversão para 2D já implementado,

além de maior poder de processamento gráfico do *hardware* utilizado.

6.2 SLAM

Nota-se que com o recurso de SLAM não é possível reconhecer as faixas de trânsito como obstáculos que devem ser evitados, como é necessário em uma situação de navegação real. Essa limitação é evidente para a implementação de uma navegação autônoma eficaz, mas foge do escopo do projeto e pode ser foco de futuros trabalhos que utilizarem a plataforma.

Outra limitação evidente do sistema é a realização do mapeamento em somente duas dimensões. Para ambientes relativamente planos, isso não se mostra um problema para a navegação, no entanto, caminhos com alterações abruptas de relevo são interpretados como obstruídos pelo algoritmo.

6.3 Planejamento de rota

O planejamento de rota implementado por meio do algoritmo Smac-Hybrid A* busca encontrar a trajetória com maior eficiência no período de tempo disponível para busca, utilizando de um mapeamento de custos de navegação por cada região do mapa disponível. Essa abordagem é efetiva em diversos ambientes, como o estacionamento, mas não se adéqua diretamente ao tráfego urbano. A navegação orientada por leis de trânsito requer que o veículo as respeite para harmonia e segurança nas cidades. Isso pode ser implementado ao elevar o custo de regiões como faixas de trânsito por meio de reconhecimento de padrões visuais, incorporando essa informação ao *costmap* para que seja considerada no planejamento de rotas.

Além disso, o planejador se mostrou indeciso ao tentar executar manobras com grande proximidade a obstáculos, muitas vezes planejando rotas com sentidos opostos de maneira alternada, de forma a impedir que o controlador obtenha progresso na navegação. Esse problema ocorre devido a diversos fatores, como: homogeneidade entre os custos atribuídos a regiões no entorno imediato do veículo, tornando diversas rotas similarmente prováveis de serem escolhidas; discordância entre a rota planejada e a executada, o que leva o planejador a recalcular uma rota a partir de um ponto que não estava incluso na anterior; e a alta frequência de execução do planejamento, que, embora importante para a adaptação a obstáculos dinâmicos, tal responsabilidade pode ser atribuída a um

planejador local mais complexo que o RPP utilizado.

6.4 Navegação

Conclui-se que algoritmo RPP é adequado quando o requisito da tarefa de navegação é a execução da trajetória global de maneira direta, sem a preocupação com obstáculos dinâmicos.

Para o estacionamento, demonstrou eficácia ao estacionar em vagas inclinadas, onde não há a necessidade de muitas manobras, sendo assim capaz de completar a tarefa sem colisões 5.6.

Já para o estacionamento paralelo, dois testes foram executados, um entrando na vaga de frente, Figura 5.7 e outro de ré, Figura 5.8. Devido à natureza da vaga paralela, cujo estacionamento envolve manobras frequentes e ajustes finos a posição do veículo, o resultado não foi satisfatório. O planejador enfrentou dificuldades em manter uma rota consistente, resultando em variações significativas dos comandos de controle gerados pelo RPP. Os experimentos culminaram em um estacionamento parcial, sem a conclusão bem-sucedida da manobra, mesmo com o envio de um novo comando de posição de destino sendo enviado para a navegação, de modo a provocar um segundo ajuste do veículo, como mostram as animações das Figuras 5.7 e 5.8.

Diante dessas dificuldades, uma possível solução é a criação uma Árvore de Comportamento personalizada para estacionamento em vagas paralelas, principalmente as de tamanho reduzido, que demandam uma abordagem com controle mais fino. Tal árvore foi esquematizada, mas não validada em tempo hábil para esse projeto.

6.5 Árvore de Comportamento personalizada para estacionamento paralelo

No âmbito do projeto da Árvore de Comportamento para o estacionamento paralelo, é possível agregar mais condições de segurança global sob o mesmo nó de *fallback* na árvore representada na Figura 5.11. Essa abordagem ampliaria a capacidade do sistema de reagir a uma variedade mais ampla de cenários inesperados, proporcionando uma resposta coordenada e eficiente diante de falhas potenciais.

Outra sugestão relevante seria a implementação de uma condição de saída da vaga para uma possível nova tentativa em caso de falha, em vez de uma simples interrupção

do processo. Essa modificação possibilitaria ao sistema realizar uma nova tentativa de estacionamento paralelo, conferindo uma abordagem mais dinâmica e adaptativa.

Adicionalmente, visando ampliar a aplicabilidade do sistema a diferentes cenários de estacionamento, seria apropriado contemplar rotinas alternativas para outros tipos de vagas. Isso envolveria a expansão da lógica da árvore para lidar com variações nas dimensões e orientações das vagas, proporcionando ao veículo a capacidade de se adaptar a diferentes ambientes de estacionamento. A incorporação de rotinas alternativas tornaria o sistema mais flexível e capaz de lidar com uma gama mais ampla de situações.

Essas considerações adicionais não apenas reforçariam a segurança e eficiência do sistema de estacionamento paralelo, mas também abririam perspectivas para futuras melhorias e expansões, conferindo-lhe maior adaptabilidade e resiliência face aos desafios do mundo real.

Vale ressaltar que, apesar da concepção de todo o procedimento para estacionamento em vagas paralelas, as limitações temporais impediram a conclusão e teste abrangente do mesmo. Ainda assim, com suas distintas etapas e estruturas de árvores de decisão, a rotina representa uma base sólida para a implementação do estacionamento em vagas paralelas e oferece uma fundação inicial para futuros desenvolvimentos em estacionamento em vagas de tamanho mais reduzido.

7 CONCLUSÃO E TRABALHOS FUTUROS

Na condução deste trabalho, foram realizadas análises detalhadas em diversas áreas do desenvolvimento de sistemas autônomos de navegação. Foram apresentados dados satisfatórios quanto ao mapeamento, localização e planejamento de rota para um veículo autônomo, dado o escopo de aplicação do projeto.

No que diz respeito à percepção, após refinamento de configuração, foi possível a detecção eficiente de obstáculos, como a guia da calçada, de modo a garantir a navegação dentro dos limites da rua para o veículo, por mais que o reconhecimento de sinalizações de trânsito esteja ausente. Além disso, o planejamento de rota, conduzido pelo algoritmo Smac-Hybrid A*, foi eficaz em ambientes como estacionamentos em vagas com tamanho ampliado, mas não foi efetivo em cenários com alta proximidade a obstáculos e trajetórias curtas.

A avaliação do algoritmo RPP para navegação indicou sua adequação para execução de trajetórias longas de forma eficaz, mas alertou para a imprecisão para execução de trajetórias curtas, o que afeta também o bom funcionamento do planejador global.

Por fim, a Árvore de Comportamento para estacionamento, por mais que modelada, não foi implementada e validada em tempo hábil e é sugerida como trabalho futuro.

Também sugere-se como trabalho futuro a embarcação do sistema no protótipo físico descrito na Seção 2.4, com mínimas adaptações a serem realizadas, como a fusão dos dados de nuvem de pontos das 4 câmeras RGBD do modelo para gerar a entrada do sistema de mapeamento.

Dentre os requisitos levantados, foi possível cumprir os que se referem à definição de trajetória, à realização de manobra de estacionamento para vagas a 45^o e segurança do veículo, mas foram somente parcialmente cumpridos os requisitos referentes à técnica SLAM com visão estéreo, cujos dados de entrada foram provenientes de um LIDAR 3D, e ao estacionamento em vaga paralela, que apresentou desempenho insatisfatório com a metodologia validada.

Em conclusão, este trabalho proporcionou uma visão abrangente do desenvolvimento de sistemas autônomos de navegação, destacando realizações e apontando desafios e oportunidades para futuras melhorias. As limitações identificadas fornecem direcionamentos claros para refinamentos e expansões em projetos subsequentes, visando aprimorar a eficiência, segurança e adaptabilidade de veículos autônomos em ambientes complexos.

REFERÊNCIAS

1 World Health Organization and others. Road safety. OECD, 2020.

2 MOHAMMED, A. A. et al. A review of traffic accidents and related practices worldwide. *The Open Transportation Journal*, v. 13, n. 1, 2019.

3 GONIEWICZ, K. et al. Road accident rates: strategies and programmes for improving road traffic safety. *European Journal of Trauma and Emergency Surgery*, Springer Science and Business Media LLC, v. 42, n. 4, p. 433–438, jul. 2015. Disponível em: ⟨https://doi.org/10.1007/s00068-015-0544-6⟩.

4 CLAUSSMANN, L. et al. A review of motion planning for highway autonomous driving. *IEEE Transactions on Intelligent Transportation Systems*, IEEE, v. 21, n. 5, p. 1826–1848, 2019.

5 IIHS. *Advanced driver assistance*. 2022. ⟨https://www.iihs.org/topics/advanced-driver-assistance⟩. Acesso em: 12 mar. 2023.

6 BANERJEE, S. S. et al. Hands off the wheel in autonomous vehicles?: A systems perspective on over a million miles of field data. In: IEEE. *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. [S.l.], 2018. p. 586–597.

7 SCHWARTING, W.; ALONSO-MORA, J.; RUS, D. Planning and decision-making for autonomous vehicles. *Annual Review of Control, Robotics, and Autonomous Systems*, Annual Reviews, v. 1, p. 187–210, 2018.

8 SCHWALL, M. et al. Waymo public road safety performance data. *arXiv preprint arXiv:2011.00038*, 2020.

9 WANG, J.; LIU, J.; KATO, N. Networking and communications in autonomous driving: A survey. *IEEE Communications Surveys & Tutorials*, IEEE, v. 21, n. 2, p. 1243–1274, 2018.

10 CHEN, T.; CHEN, F.; CHEN, C. Review on driverless traffic from management perspective. In: EDP SCIENCES. *MATEC Web of Conferences*. [S.l.], 2017. v. 124, p. 03002.

11 O'NEILL, B. Preventing passenger vehicle occupant injuries by vehicle design—a historical perspective from iihs. *Traffic injury prevention*, Taylor & Francis, v. 10, n. 2, p. 113–126, 2009.

12 RETTING, R. A.; FERGUSON, S. A.; MCCARTT, A. T. A review of evidence-based traffic engineering measures designed to reduce pedestrian–motor vehicle crashes. *American journal of public health*, American Public Health Association, v. 93, n. 9, p. 1456–1463, 2003.

13 O'NEILL, B.; MOHAN, D. Reducing motor vehicle crash deaths and injuries in newly motorising countries. *Bmj*, British Medical Journal Publishing Group, v. 324, n. 7346, p. 1142–1145, 2002.

14 THOMAS, E. et al. Perception of autonomous vehicles by the modern society: a survey. *IET Intelligent Transport Systems*, Wiley Online Library, v. 14, n. 10, p. 1228–1239, 2020.

15 LIU, Q. et al. Decision-making technology for autonomous vehicles: Learning-based methods, applications and future outlook. In: IEEE. *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. [S.l.], 2021. p. 30–37.

16 LIU, W. et al. A systematic survey of control techniques and applications: From autonomous vehicles to connected and automated vehicles. *arXiv preprint arXiv:2303.05665*, 2023.

17 Society of Automotive Engineers. *Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles," J3016_201609*. [S.l.]: SAE International, Warrendale, 2016.

18 ARADI, S. Survey of deep reinforcement learning for motion planning of autonomous vehicles. *IEEE Transactions on Intelligent Transportation Systems*, IEEE, v. 23, n. 2, p. 740–759, 2020.

19 COLLEDANCHISE, M.; ÖGREN, P. *Behavior Trees in Robotics and AI*. CRC Press, 2018. Disponível em: ⟨https://doi.org/10.1201/9780429489105⟩.

20 IOVINO, M. et al. A survey of behavior trees in robotics and ai. *Robotics and Autonomous Systems*, Elsevier, v. 154, p. 104096, 2022.

21 GHZOULI, R. et al. Behavior trees in action: a study of robotics applications. In: *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*. [S.l.: s.n.], 2020. p. 196–209.

22 COLLEDANCHISE, M.; FACONTI, D. *BehaviorTree.CPP Library Documentation*. Disponível em: ⟨https://www.behaviortree.dev⟩. Acesso em: 14 nov. 2023.

23 MACENSKI, S. *ROS2 Navigation 2 Behavior Tree Plugins*. Disponível em: ⟨https://github.com/ros-planning/navigation2/tree/main/nav2_behavior_tree/plugins⟩. Acesso em: 15 nov. 2023.

24 MACENSKI, S. *Navigation 2 Documentation*. Disponível em: ⟨https://navigation.ros.org/index.html⟩. Acesso em: 10 nov. 2023.

25 MACENSKI, S. et al. The marathon 2: A navigation system. In: IEEE. *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. [S.l.], 2020. p. 2718–2725.

26 MARUYAMA, Y.; KATO, S.; AZUMI, T. Exploring the performance of ros2. In: *Proceedings of the 13th International Conference on Embedded Software*. [S.l.: s.n.], 2016. p. 1–10.

27 D'ELIA, F.; MORALES, G. *Autonomous vehicle implementation in scale model with ROS 2*. 2022. Disponível em: ⟨https://pcs.usp.br/pcspf/wp-content/uploads/sites/8/2022/12/Monografia_PCS3560_SEM_2022_Grupo_S13.pdf⟩.

28 DOSOVITSKIY, A. et al. Carla: An open urban driving simulator. In: PMLR. *Conference on robot learning*. [S.l.], 2017. p. 1–16.

29 QUIGLEY, M. et al. Ros: an open-source robot operating system. In: KOBE, JAPAN. *ICRA workshop on open source software*. [S.l.], 2009. v. 3, n. 3.2, p. 5.

30 PARDO-CASTELLOTE, G. Omg data-distribution service: Architectural overview. In: IEEE. *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.* [S.l.], 2003. p. 200–206.

31 MACENSKI, S. et al. Robot operating system 2: Design, architecture, and uses in the wild. *Science Robotics*, American Association for the Advancement of Science, v. 7, n. 66, p. eabm6074, 2022.

32 CARLA ROS2 Bridge. Disponível em: ⟨https://github.com/carla-simulator/ros-bridge⟩. Acesso em: 20 mai. 2023.

33 LALANCETTE, C. *Pointcloud to Laserscan*. Disponível em: ⟨https://github.com/ros-perception/pointcloud_to_laserscan⟩. Acesso em: 16 ago. 2023.

34 ZHANG, H. et al. Energy efficient path planning for autonomous ground vehicles with ackermann steering. *Robotics and Autonomous Systems*, Elsevier, v. 162, p. 104366, 2023.

35 DOLGOV, D. et al. Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, v. 29, n. 5, p. 485–501, 2010. Disponível em: ⟨https://doi.org/10.1177/0278364909359210⟩.

36 DIJKSTRA, E. W. A note on two problems in connexion with graphs. *Numerische mathematik*, Springer, v. 1, n. 1, p. 269–271, 1959.

37 MACENSKI, S. et al. *Regulated Pure Pursuit for Robot Path Tracking*. 2023.

38 WILLIAMS, G. et al. Aggressive driving with model predictive path integral control. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. [S.l.: s.n.], 2016. p. 1433–1440.