UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA MECÂNICA

JHONAS PRADO MOURA

INVESTIGAÇÃO DO DESEMPENHO DO PLANEJADOR DE TRAJETÓRIAS MOTION PLANNING NETWORKS

Uberlândia, MG
2021

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA

JHONAS PRADO MOURA

INVESTIGAÇÃO DO DESEMPENHO DO PLANEJADOR DE TRAJETÓRIAS MOTION PLANNING NETWORKS

Monografia apresentada ao Curso de Engenharia Mecatrônica da Universidade Federal de Uberlândia como parte dos requisitos necessários para a obtenção do grau de Bacharel em Engenharia Mecatrônica.

Orientadora: Profª Drª Rita Maria da Silva Julia

Uberlândia, MG
2021

Jhonas Prado Moura

INVESTIGAÇÃO DO DESEMPENHO DO PLANEJADOR DE TRAJETÓRIAS MOTION PLANNING NETWORKS

Monografia apresentada ao Curso de Engenharia Mecatrônica da Universidade Federal de Uberlândia como parte dos requisitos necessários para a obtenção do título de Bacharel em Engenharia Mecatrônica.

Uberlândia, 21 de junho de 2021.

Profª Drª Rita Maria da Silva Julia
Universidade Federal de Uberlândia
Orientadora

Prof. Dr. Rogério Sales Gonçalves
Universidade Federal de Uberlândia
Examinador

Prof. Dr. Carlos Roberto Lopes
Universidade Federal de Uberlândia
Examinador

Agradecimentos

Agradeço inicialmente aos meus pais, Ramalho e Zirlene, e ao meu irmão, Thobias, pelo apoio que me deram por todos esses anos; sendo cruciais na minha formação de caráter e sempre tornando meus dias mais felizes. A minha namorada Júlia, que me apoiou incondicionalmente pelos percalços que ocorreram durante o desenvolvimento deste trabalho.

À Equipe de Desenvolvimento em Robótica Móvel por despertar em mim a paixão pelo desenvolvimento de *softwares* para robótica; além de disponibilizar a oportunidade de trabalhar nesta área e entrar em contato com as mentes primorosas que trabalham ativamente para avançá-la. Ao professor Rogério, pelo suporte e orientação durante minha participação na equipe.

À Universidade Federal de Uberlândia pelo curso de Engenharia Mecatrônica ministrado e pelas oportunidades a mim disponibilizadas.

Ao doutorando Matheus Prandini, que foi de extrema importância na realização deste trabalho, me guiando durante todo seu desenvolvimento. Por fim, agradeço à professora orientadora Rita, pelo suporte prestado durante o desenvolvimento deste trabalho, contribuindo imensamente com seu conhecimento e sua experiência.

Resumo

O presente trabalho busca investigar a aplicação de um planejador de trajetórias baseado em Aprendizado de Máquina na resolução de problemas em que é necessário planejar uma trajetória entre dois pontos de um ambiente, evitando possíveis colisões com obstáculos nele presentes. Os ambientes aqui utilizados são bidimensionais, estáticos e apresentam obstáculos com tamanhos iguais. A abordagem escolhida na resolução deste problema foi o planejador *Motion Planning Networks*. Neste algoritmo de planejamento, destacam-se os seguintes elementos: uma rede neural que sintetiza as informações provenientes do ambiente em uma representação compacta e eficiente; uma rede neural que gera gradativamente um conjunto de pontos candidatos à trajetória; um algoritmo otimizador que elimina pontos desnecessários deste conjunto; e, um replanejador que tenta corrigir eventuais falhas na trajetória planejada. A viabilidade desta abordagem foi avaliada por meio da análise dos seguintes parâmetros: taxa de sucesso em gerar uma trajetória válida e tempo de execução. Por meio de tal análise, concluiu-se que o algoritmo *Motion Planning Networks* é consistente em planejar trajetórias válidas para os referidos problemas, raramente apresentando falhas; bem como apresentou, frequentemente, tempos de execução abaixo de meio segundo. O tempo de execução do MPNet é significantemente menor do que o obtido utilizando o planejador RRT* para as mesmas trajetórias.

Palavras-chave: Planejamento de trajetórias. Motion Planning Network. Aprendizado de Máquina. Autoencoder. Contractive Autoencoder. Perceptron de Múltiplas Camadas. RRT*.

Abstract

This study aims to investigate the application of a Machine Learning-based motion planner for solving problems in which a path between two points must be planned, avoiding any collisions with obstacles present in the environment. The environments herein used are bidimensional, static and feature obstacles of the same size. The chosen approach for solving the said problem was the *Motion Planning Networks* algorithm. In this planning algorithm, the following elements are highlighted: a neural network that synthesizes the environment data in an embedded and efficient representation; a neural network that gradually processes a set of candidate points to the planned path; an optimizer algorithm, which removes unnecessary points from such set; and, a replanning algorithm which attempts to fix eventual mistakes in the planned path. The viability of this approach was evaluated through the analysis of the following parameters: success rate in planning a valid path and computation time. As a result of said analysis, it is concluded that the *Motion Planning Networks* algorithm is consistent in planning valid paths for said problems, rarely failing; besides, it regularly took less than half a second to plan a path. MPNet's mean computation time is significantly lower than the one obtained using the RRT* planner for the same paths.

Keywords: Motion Planning. *Motion Planning Networks*. Machine Learning. Autoencoder. Contractive Autoencoder. Multilayer Perceptron. RRT*.

Lista de ilustrações

Lista de tabelas

Lista de siglas

AE Autoencoder.

AM Aprendizado de Máquina.

CAE Contractive Autoencoder.

EMQ Erro Médio Quadrático.

ENet Rede de Codificação.

LSC Lazy States Contraction.

MPNet Motion Planning Networks.

PBA Planejador baseado em Amostragem.

PMC Perceptron de Múltiplas Camadas.

PNet Rede de Planejamento.

PReLU Parametrized Rectified Linear Unit.

PRM Probabilistic Roadmap.

RBM Restricted Boltzmann Machine.

RH Replanejamento Híbrido.

RN Replanejamento Neural.

RRT Rapidly-exploring Random Trees.

RRT* RRT Otimizado.

Sumário

1 Introdução

A robótica é definida como a conexão inteligente entre a percepção do ambiente e a ação executada por um agente com o objetivo de resolver o problema para o qual foi projetado (tendo como base a referida percepção) (BRADY, 1985). A percepção é caracterizada pelo sensoriamento, ou seja, os agentes são capazes de adquirir informações através do uso de sensores de toque, força, distância, entre outros. A capacidade de atuar é provida por braços mecânicos, rodas, e até mesmo pernas. Estudos recentes na robótica têm contribuído para o desenvolvimento de novas tecnologias como robôs colaborativos (HADIDI *et al.*, 2018), auxílio avançado de direção para veículos (OKUDA; KAJIWARA; TERASHIMA, 2014) e agentes com inteligência de enxame (KAUR; KUMAR, 2020). Essas tecnologias têm potencial de reduzir o número de acidentes em atividades como a manufatura (MATTHIAS *et al.*, 2011; ROBLA-GÓMEZ *et al.*, 2017), o transporte (MORANDO *et al.*, 2018), dentre outros.

No âmbito computacional, a robótica emprega diversos sistemas, como os de percepção (AVENDANO-VALENCIA; DIMEAS; ASPRAGATHOS, 2014) e de comportamento (MOURET; DONCIEUX, 2012). O processamento dos dados adquiridos pelo sensoriamento habilita o agente a perceber o ambiente. Já o comportamento é definido pelo conjunto de algoritmos responsáveis pela tomada de decisões e integração entre os subsistemas do agente.

O presente estudo se concentra na subárea da navegação, que orienta o agente na execução de um movimento por intermédio de dados provenientes da percepção e do comportamento. A navegação é composta por diversos algoritmos que possuem responsabilidades distintas, como interpretar e mapear o ambiente, situar o agente neste ambiente, localizar pontos de interesse e planejar trajetórias entre estados requisitados pelo algoritmo de comportamento.

Os algoritmos ligados à percepção devem analisar e adquirir informações sobre pontos de interesse no ambiente, como obstáculos, agentes e objetivos.

Os algoritmos relacionados ao comportamento devem interpretar o estado atual do agente, definir qual o próximo estado a ser atingido e requisitar ao sistema de planejamento uma trajetória que conduza do estado corrente ao desejado. O sistema de planejamento da trajetória, ao receber a requisição do algoritmo de comportamento, planeja uma trajetória entre os estados requisitados, evitando obstáculos previamente detectados.

Dessa forma, o planejador de trajetórias é o algoritmo responsável por definir a trajetória a ser percorrida pelo agente de um ponto inicial até um ponto objetivo, evitando possíveis obstáculos. Este ponto objetivo varia de acordo com o tipo do agente executor. Um veículo autônomo, por exemplo, pode ter como objetivo uma pose detalhando sua posição e orientação no mundo (FERGUSON; HOWARD; LIKHACHEV, 2008), enquanto que um robô manipulador industrial pode ter como objetivo uma determinada pose detalhando a rotação e translação de

seus atuadores (QURESHI; MIAO *et al.*, 2019).

Um planejador de trajetórias pode ser categorizado como global ou local. Como definido em Marín *et al.* (2018), um planejador global é aquele que usa as informações adquiridas até o momento referentes ao espaço de estados do problema para encontrar a trajetória a ser percorrida até o destino. Um planejador local é definido como o planejador responsável por recalcular a trajetória resultante do planejamento global somente em um raio ao redor do agente, buscando desviar de obstáculos não percebidos durante a execução do planejamento global.

No estado da arte de planejadores, destacam-se algoritmos de busca heurística, como A^* (KURZER, 2016) e Dijkstra (SNIEDOVICH, 2006), bem como os Planejadores baseados em Amostragem (PBAs), como *Rapidly-exploring Random Trees* (RRT) (LAVALLE, 2006) e RRT Otimizado (RRT*) (KARAMAN; FRAZZOLI, 2011). Tanto algoritmos baseados em heurística quanto baseados em amostragem possuem como fragilidade o elevado aumento do custo computacional com relação à dimensão do ambiente. Além disso, algoritmos iterativos como RRT* só garantem que a trajetória produzida seja próxima à ótima quando o número de amostras tende ao infinito. De modo a minimizar tais limitações, soluções baseadas em Aprendizado de Máquina (AM) têm sido objeto de pesquisa (YU; SU; LIAO, 2020; YU; HILL *et al.*, 2018; WULFMEIER *et al.*, 2017).

Dentre as abordagens baseadas em AM, destaca-se o planejador *Motion Planning Networks* (MPNet) (QURESHI; MIAO *et al.*, 2019). O MPNet é um algoritmo de planejamento composto pelos seguintes elementos:

- uma rede neural responsável por produzir uma representação sintetizada do ambiente;
- uma rede neural responsável por gerar gradativamente no ambiente um dos pontos candidatos a compor uma trajetória, pontos, estes, denominados estados críticos;
- um algoritmo que gradativamente proponha uma conexão entre os estados críticos gerados visando a compor uma candidata à trajetória (ou seja, uma sequência de estados críticos que leve do estado inicial ao estado final);
- um otimizador de trajetórias que, uma vez gerada uma sequência completa candidata à trajetória, tenta otimizá-la;
- e, por último, um sistema de replanejamento que tente resgatar uma trajetória válida a partir dos pontos usados em fracassadas candidatas à trajetória (que mesmo o otimizador não conseguiu reparar).

As redes neurais de geração de uma representação compactada do ambiente e de geração de estados críticos são denominadas, respectivamente, Rede de Codificação (ENet) e Rede de Planejamento (PNet), sendo que seus parâmetros e suas configurações originais foram propostos em Qureshi, Miao *et al.* (2019).

A ENet tem a responsabilidade de processar uma representação sintetizada a partir da representação original do ambiente, reduzindo a sua dimensão e, consequentemente, o tempo de execução e o custo de memória da etapa de geração de estados críticos. É interessante ressaltar que a representação simplificada obtida pode ser generalizada para representar diversos ambientes com natureza similar aos presentes na sua base de dados de treinamento. Para isso, a abordagem originalmente utilizada foi o *Autoencoder* (AE) (BANK; KOENIGSTEIN; GIRYES, 2020), uma rede neural que processa uma representação original de modo a gerar uma representação codificada e sintetizada, que pode ser utilizada para redução de dimensão (ICHTER; PAVONE, 2018), no pré-treinamento de redes neurais (SAGHEER; KOTB, 2019), no reconhecimento de imagens (PENG *et al.*, 2017) e nos sistemas de recomendação (ZHANG; LIU; JIN, 2020).

A PNet consiste em um Perceptron de Múltiplas Camadas (PMC) (GARDNER; DORLING, 1998) responsável por indicar iterativamente um estado crítico como candidato a pertencer à trajetória resultante. Para tal, a referida rede utiliza as seguintes informações: representação codificada do ambiente; estado atual do agente; e estado destino. Essas informações, então, são processadas pela PNet, de modo que resulte em um novo estado crítico candidato a pertencer à trajetória.

As redes neurais que compõem a arquitetura da abordagem MPNet podem ser treinadas por meio das seguintes metodologias (QURESHI; MIAO *et al.*, 2019): método com aprendizado em lotes *offline*, método com aprendizado contínuo e método com aprendizado contínuo ativo.

O algoritmo de conexão entre estados críticos tem como finalidade efetuar conexões entre tais estados na medida em que eles são produzidos pela PNet. Uma conexão é considerada válida quando não colide com obstáculos. Tão logo seja produzida uma sequência completa de conexões que seja candidata à trajetória, o otimizador de trajetórias atua no sentido de aprimorá-la ao máximo, o que não significa, necessariamente, que ele será bem sucedido na tarefa de produzir uma trajetória válida (ou seja, que leve da origem ao destino sem colisões com obstáculos).

O MPNet efetua o replanejamento quando são detectadas colisões com obstáculos em um ou mais trechos da trajetória otimizada. Para tal, Qureshi, Miao *et al.* (2019) apresenta duas técnicas: Replanejamento Neural (RN) e Replanejamento Híbrido (RH). O RN busca uma trajetória alternativa para conectar cada par de estados consecutivos, referentes aos trechos em que foram detectadas tais colisões. Isso se repete até que seja obtida uma trajetória geral válida ou, então, que seja atingido um determinado limite de iterações. O RH requisita a outro planejador (como, por exemplo, o RRT*), denominado *planejador oráculo*, a geração de uma trajetória que conecte os pares de estados consecutivos para os quais o RN não tenha sido capaz de gerar uma trajetória válida.

Em Qureshi, Miao *et al.* (2019), vários tipos de ambientes foram utilizados como estudo de caso para a aplicação do MPNet. Um deles foi o "*Simples 2D*", que consiste em um ambiente bidimensional, completamente observável, determinístico, estático, conhecido, com agente único e com obstáculos de dimensões iguais. Esse tipo de ambiente pode ser gerado algoritmicamente

de maneira simples, possibilitando a coleta de um grande volume de dados, o que faz com que a metodologia de treinamento sugerida para ele seja o método de aprendizado em lotes *offline* (QURESHI; MIAO *et al.*, 2019).

1.1 Motivação

O planejamento autônomo consiste no processo de criar especificações de comportamento algoritmicamente (MCDERMOTT, 2003), sendo responsável pela realização de estratégias ou sequências de ações de agentes inteligentes. Sua importância pode ser notada em uma grande variedade de problemas, tais como em problemas de locomoção (MCNAUGHTON *et al.*, 2011), problemas de comunicação (YAN; MOSTOFI, 2013) e problemas de manipulação (ALAMI; LAUMOND; SIMÉON, 1995). O planejamento de trajetórias é essencial para a navegação de agentes móveis (PARK; HUH, 2016), estando presente em várias tecnologias emergentes, como veículos autônomos (OKUDA; KAJIWARA; TERASHIMA, 2014) e robôs colaborativos (HADIDI *et al.*, 2018).

Com base na relevância do planejamento de trajetórias nas tecnologias modernas, a motivação do presente trabalho é de implementar e aplicar o MPNet a ambientes Simples 2D utilizando a configuração original de parâmetros para a ENet e a PNet proposta em Qureshi, Miao *et al.* (2019).

Além disso, outro fator de motivação para esta pesquisa é a realização de uma análise comparativa entre os algoritmos MPNet e RRT*, o que será feito por meio da comparação do tempo que cada uma destas abordagens leva para encontrar a trajetória final.

1.2 Objetivos

Considerando os pontos apresentados na seção 1.1, o objetivo geral do presente trabalho é implementar[1] e aplicar o algoritmo MPNet com as configurações sugeridas em Qureshi, Miao *et al.* (2019) para a ENet e a PNet. A técnica de replanejamento utilizada nesta aplicação é o RN.

A performance do sistema será avaliada por meio da análise dos seguintes parâmetros: taxa de sucesso na geração de trajetórias válidas e tempo de processamento necessário para o processamento de cada trajetória. Para a análise do segundo parâmetro, é realizada uma comparação com o tempo de processamento do planejador RRT* para as mesmas trajetórias propostas.

Especificamente neste trabalho, implementa-se o MPNet treinado com aprendizado em lotes *offline*, por esta ser a abordagem sugerida em Qureshi, Miao *et al.* (2019) para os ambientes escolhidos como estudo de caso. Este método envolve o treinamento das redes ENet

[1] Os algoritmos implementados na execução do presente trabalho são disponibilizados em: `https://github.com/jhonasiv/MPNet`.

e PNet utilizando uma base de dados previamente adquirida. Para o treinamento da ENet, a presente pesquisa utiliza cenários distintos gerados algoritmicamente, enquanto que para realizar o treinamento da PNet são geradas trajetórias por meio do planejador RRT*.

Desta forma, de modo a perfazer o objetivo geral aqui proposto, os seguintes objetivos específicos devem ser perseguidos:

1. Desenvolvimento de método para geração de cenários. Tal método tem como propósito gerar uma grande quantidade de cenários quadrados com mesmas dimensões, cada uma com o mesmo número de obstáculos de igual tamanho, garantindo que nenhum obstáculo tenha o mesmo ponto como centro. Os cenários gerados são utilizados no cumprimento dos seguintes objetivos específicos: 2; 3a; e 4.

2. Geração de trajetórias a serem usadas como amostras de treinamento. O objetivo aqui é gerar trajetórias com os cenários produzidos no cumprimento do objetivo específico 1, utilizando o algoritmo planejador baseado em amostragem RRT*. Tais trajetórias serão utilizadas para cumprir o objetivo específico 3b.

3. Implementação do planejador MPNet.

    a) Implementação e treinamento da ENet utilizando a configuração sugerida em Qureshi, Miao *et al.* (2019). A entrada da rede corresponde a cada cenário gerado no cumprimento do objetivo específico 1 e sua saída equivale a uma matriz codificada de dimensões reduzidas, que representa o cenário de entrada de um modo sintetizado. O treinamento desta rede não requer dados rotulados, pois consiste em um AE que utiliza os dados de entrada como referência na fase de treinamento (em estratégia análoga à técnica de treinamento *Reconstrução de Dados* (HINTON; OSINDERO; TEH, 2006)).

    b) Implementação e treinamento da PNet utilizando a configuração sugerida em Qureshi, Miao *et al.* (2019). Sua entrada consiste na matriz codificada do cenário a ser analisado (adquirida através da ENet treinada no cumprimento do objetivo específico 3a) no estado atual do agente e no estado destino. Sua saída equivale a um estado crítico candidato a pertencer à trajetória. Além disso, cada estado crítico presente nas trajetórias geradas no cumprimento do objetivo específico 2 é utilizado na criação da base de dados de treinamento desta rede neural.

    c) Implementação dos algoritmos de conexão de estados críticos, otimização da trajetória e de replanejamento presentes na MPNet.

    d) Integração entre os algoritmos anteriormente implementados, formando o planejador MPNet.

4. Realização dos experimentos que possibilitam a aquisição de dados de desempenho do planejador aqui estudado, bem como a avaliação da sua taxa de sucesso na geração de

trajetórias válidas e do tempo gasto no processamento da trajetória em comparação com o algoritmo RRT*.

1.3 Organização do Trabalho

Esta monografia é composta por cinco capítulos além da introdução. Suas finalidades são descritas a seguir:

Fundamentação Teórica - descreve os fundamentos teóricos necessários para a compreensão e reprodução deste trabalho;

Trabalhos Relacionados - realiza uma breve descrição de alguns métodos utilizados no estado da arte de planejadores de trajetória;

Geração das bases de dados e implementação do MPNet - explica os procedimentos utilizados para a aquisição de dados, treinamento da ENet e da PNet, e implementação do planejador MPNet;

Análise dos Resultados - dispõe os resultados obtidos, analisando os parâmetros indicados na seção 1.2;

Conclusão e Trabalhos Futuros - conclui o projeto analisando o valor dos resultados obtidos e apresentando propostas para serem estudadas futuramente;

2 Fundamentação Teórica

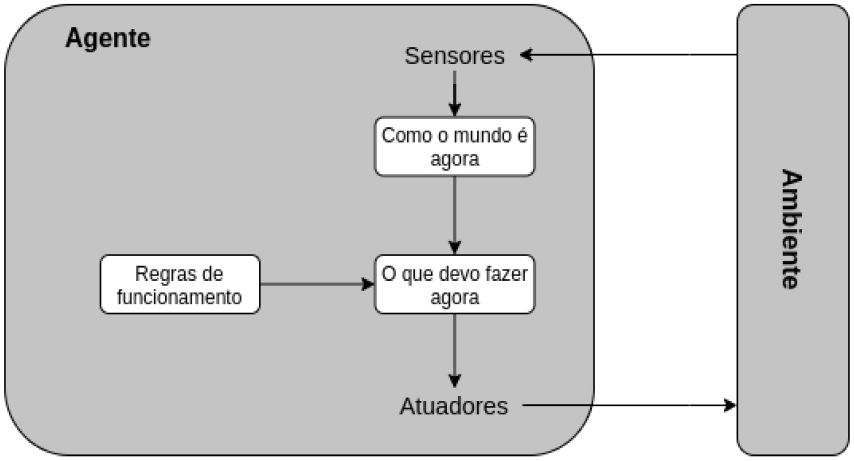
Este capítulo apresenta uma síntese dos conceitos e das técnicas que embasam o presente trabalho, tendo como propósito discorrer sobre fundamentos que auxiliam na compreensão da maneira como o problema tratado foi abordado no contexto desta monografia.

2.1 Agentes

Um agente é tudo o que pode ser considerado capaz de perceber seu ambiente por meio de sensores e de agir sobre esse ambiente por intermédio de atuadores (RUSSELL; NORVIG, 2009). A Figura 2.1 ilustra essa relação. Para um agente robótico, são exemplos de sensores: câmeras, giroscópios e acelerômetros. Motores são exemplos de atuadores.

No contexto de agentes, o termo percepção, definido por Russell e Norvig (2009), é usado ao se referenciar as entradas perceptivas do agente em um dado instante. Já uma sequência de percepções do agente é a história de tudo o que o agente percebeu em um determinado intervalo de tempo. A maneira como o agente se comporta no ambiente pode depender de toda sequência de percepções recebidas até o instante corrente. O comportamento do agente é descrito pela função que mapeia qualquer sequência de percepções específica a uma ação.

Figura 2.1 – Relação entre agente e ambiente



Fonte: Autor[1]

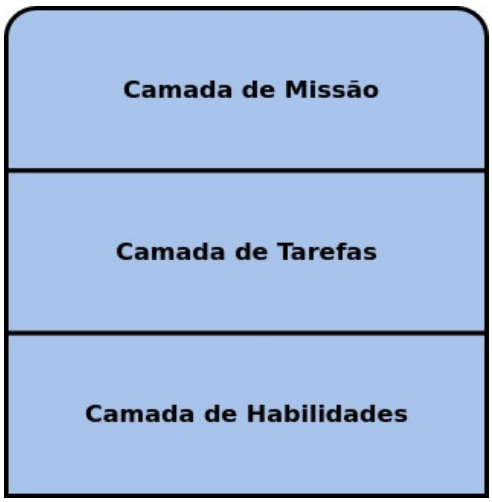
No contexto do presente trabalho, englobando a área de robótica, o modelo utilizado para estruturar o comportamento do agente é o modelo referencial funcional (PUTZ; ELFVING, 1993). Este modelo, ilustrado na Figura 2.2, define uma representação unificada de funções essenciais

[1] Adaptada a partir da tradução de figura obtida em Russell e Norvig (2009).

para o controle do agente, dividida nas seguintes camadas: habilidades, tarefas e missão. Cada camada soluciona problemas relacionados a um nível de abstração. Por exemplo, considerando a Figura 2.2 e a área da robótica, a camada de habilidades soluciona problemas relacionados ao nível de *hardware*, chamado de baixo nível. Por outro lado, a camada de missão soluciona os problemas relativos às tarefas a serem realizadas pelo agente, sendo que cada tarefa é composta por um conjunto de habilidades. As camadas são descritas a seguir:

- **Camada de missão**: define quais tarefas devem ser executadas e sua ordem de execução a fim de solucionar o problema. Conforme descrito por Hurmuzlu e Nwokah (2017), missões são instruções orientadas ao processo que não especificam explicitamente a maneira que uma tarefa deve ser executada, isso fica a cargo da camada de tarefas;
- **Camada de tarefas**: detalha as ações que devem ser executadas para cada tarefa definida na camada de missão. Deste modo, cada tarefa possui um conjunto de ações. É nessa camada que são definidos comportamentos para problemas isolados, como abrir uma porta, navegar pelo ambiente ou se recuperar de uma queda;
- **Camada de habilidades**: transforma o conjunto de ações, definido na camada de tarefas, em comandos de controle dos atuadores do agente. O conjunto de habilidades disponíveis para o agente equivale ao conjunto de ações que ele pode executar. Algumas das habilidades comumente presentes são a de se locomover pelo ambiente (por exemplo mover-se para direita, esquerda, cima e baixo), bem como de manipular objetos, como levantar um bloco usando uma garra;

Figura 2.2 – Relação hierárquica entre camadas do modelo referencial funcional



Fonte: Autor.

Este trabalho foca em solucionar um problema de planejamento de trajetórias, que é uma funcionalidade modelada pela camada de tarefas. Assim sendo, não é necessário um detalhamento

sobre como os atuadores do agente executam as instruções por ela determinadas. Mesmo que o modo que os atuadores realizam as instruções não seja relevante para o problema, as ações que o agente é capaz de realizar o são. Neste trabalho, o agente corresponde a uma partícula capaz de se locomover por um ambiente bidimensional, podendo mover-se em qualquer direção (vertical, horizontal e diagonal), enquanto que a percepção do ambiente é abstraída; ou seja, o agente tem acesso às informações provenientes do ambiente através de uma matriz de pontos que descreve a posição dos obstáculos nele, não havendo necessidade de sensoriamento.

Além disso, outro ponto que deve ser definido é o ambiente de tarefas, que Russell e Norvig (2009) definem como sendo os problemas para os quais os agentes são a solução. Com o intuito de modelar agentes eficientes, é essencial que seja feita uma descrição precisa do ambiente. Para classificar um ambiente, são usadas as seguintes propriedades:

- **Completamente observável versus parcialmente observável**: um ambiente completamente observável é aquele em que os sensores de um agente permitem acesso preciso a todas as informações do estado do ambiente que sejam relevantes para a tomada de decisão a cada instante. Já um ambiente parcialmente observável é todo aquele em que isso não acontece.

- **Agente único versus multiagente**: um ambiente de agente único é aquele em que não existem outros agentes interagindo nele. Intuitivamente, um ambiente de multiagentes é aquele em que existe mais de um agente interagindo nele. Essa interação pode ser tanto cooperativa quanto competitiva.

- **Determinístico versus estocástico**: quando o próximo estado do ambiente é completamente determinado pelo estado atual e pela ação executada pelo agente, esse ambiente é classificado como determinístico. Caso contrário, ele é estocástico.

- **Episódico versus sequencial**: quando a experiência do agente é dividida em episódios atômicos, o ambiente é classificado como episódico. Cada episódio consiste na percepção do agente seguida por uma ação. Além disso, o episódio seguinte não pode depender de episódios anteriores. Em ambientes sequenciais, a decisão atual pode afetar todas as decisões futuras.

- **Estático versus dinâmico**: um ambiente dinâmico é aquele que pode sofrer alterações enquanto o agente delibera. Caso contrário, ele é um ambiente estático.

- **Discreto versus contínuo**: a distinção entre discreto e contínuo aplica-se ao estado do ambiente em relação ao modo como o tempo é tratado, e ainda às percepções e ações do agente.

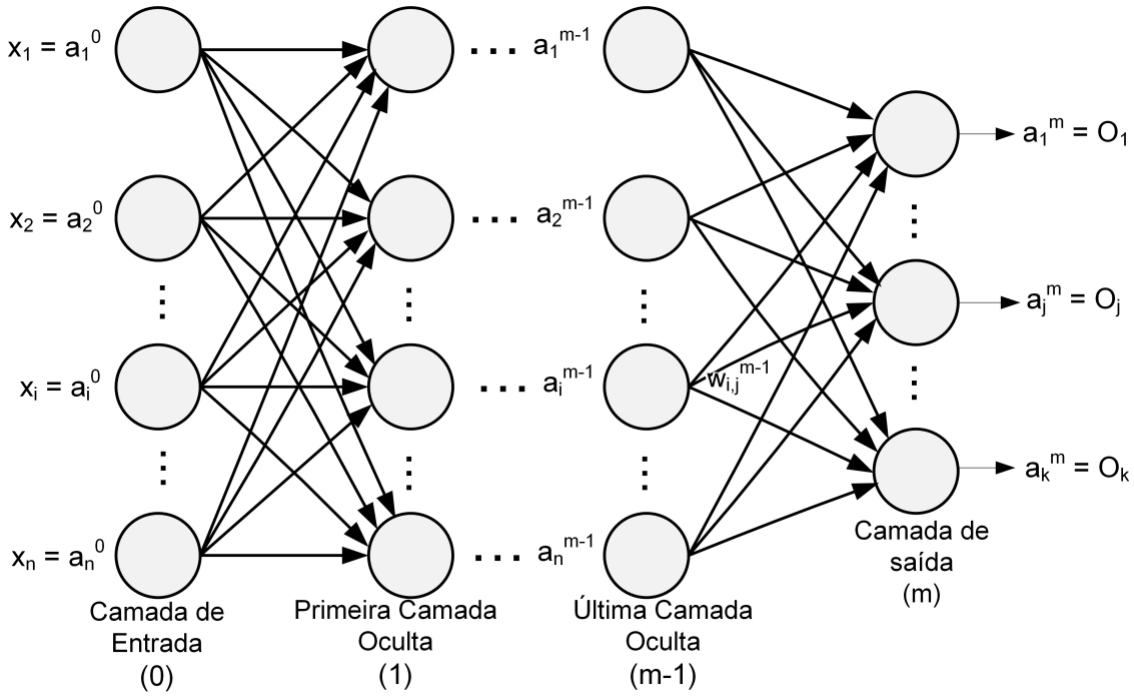
- **Conhecido versus desconhecido**: esta distinção se refere ao estado de conhecimento do agente em relação às leis que regem o ambiente. Em um ambiente conhecido, são fornecidos

os resultados produzidos pela execução de todas as ações. No entanto, em um ambiente desconhecido, o agente deve aprender quais as consequências das suas ações, para que assim consiga tomar boas decisões.

2.2 Perceptron de múltiplas camadas

Os Perceptron de Múltiplas Camadas (ou PMCs) são redes neurais artificiais que utilizam a arquitetura *feedforward* (ou seja, a propagação dos sinais de entrada ocorre em um único sentido: partindo da camada de entrada para a camada de saída) que contém mais de uma camada de neurônios processadores (SILVA; SPATTI; FLAUZINO, 2010). Estas camadas processadoras, que estão dispostas entre a camada de entrada e a camada de saída, são denominadas *camadas ocultas*, conforme ilustrado na Figura 2.3.

Figura 2.3 – Representação de um PMC



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

Resumidamente, um PMC é treinado da seguinte maneira: comparam-se as respostas produzidas nas camadas de saída da rede neural com as respectivas respostas desejadas disponibilizadas através de dados rotulados (FARIA *et al.*, 2020). Mais especificamente, o processo de ajuste de pesos sinápticos é norteado pelos desvios (erros) entre as respostas produzidas e as respostas desejadas pelos neurônios de saída, durante cada época. Para a execução deste processo, utiliza-se a estratégia de *backpropagation* (RUMELHART; HINTON; WILLIAMS, 1986) no reajuste, de modo que os erros produzidos nos neurônios de saída para cada amostra

desencadearão o seguinte fluxo: inicialmente, são utilizados no reajuste dos pesos das conexões entre a última camada oculta e a camada de saída; em seguida, são ajustados os pesos das conexões entre a penúltima e a última camadas ocultas. Essa dinâmica de retropropagação dos ajustes se repete até que ocorra o reajuste dos pesos entre a camada de entrada e a primeira camada oculta.

Particularmente no presente estudo, o PMC é empregado na geração de estados críticos da trajetória, que é descrita na seção 2.3.

2.3 Motion Planning Networks

O MPNet (QURESHI; MIAO *et al.*, 2019) é um algoritmo de planejamento de trajetórias composto pelos seguintes elementos: uma rede neural responsável por produzir uma representação compactada do ambiente; uma rede neural responsável por gerar gradativamente no ambiente um dos estados críticos candidatos a compor uma trajetória; um algoritmo que gradativamente proponha uma conexão entre os estados críticos gerados visando formar uma candidata à trajetória; um otimizador de trajetórias que atua no aprimoramento da candidata à trajetória produzida (conjuntos de estados críticos conectados), visando eliminar estados desnecessários, o que não garante a produção de uma trajetória válida (que leve da origem ao destino sem colisões com obstáculos); e, por fim, um sistema de replanejamento que, a partir dos estados críticos presentes em uma candidata à trajetória fracassada, efetua o replanejamento de trechos nos quais não foi possível conectar o par de estados consecutivos.

As redes neurais responsáveis pela geração de uma representação compactada do ambiente e de geração gradativa de estados críticos candidatos a pertencer à trajetória correspondem, respectivamente, à Rede de Codificação (ou ENet) e à Rede de Planejamento (ou PNet). O seu treinamento ocorre de acordo com a metodologia de treinamento em lotes offline.

De modo a gerar o conjunto de estados críticos candidatos a pertencer à trajetória, esta abordagem usa de uma estratégia de planejamento iterativo bidirecional (QURESHI; AYAZ, 2015).

As próximas subseções têm como propósito detalhar os elementos que compõem o MPNet.

2.3.1 Codificação do Ambiente

Considerando um ambiente A, percebido por meio dos sensores do agente, a ENet é responsável por transformar as informações do espaço de estados ocupado por obstáculos $A_{obs} \subset A$, que descreve o cenário, em uma representação codificada Z, de modo a reduzir o gasto de memória para processar o cenário durante o planejamento da trajetória. Para efetuar tal tarefa, a ENet consiste em um AE denominado *Contractive Autoencoder* (CAE) (RIFAI *et al.*, 2011). O CAE foi escolhido devido à sua capacidade de produzir uma codificação (conjunto

de características) adequada para um cenário, ainda que haja ruídos na representação que ele recebe deste cenário, o que é fundamental para manter sua funcionalidade mesmo em ambientes de elevadas dimensões.

Autoencoder

O AE é definido por Bank, Koenigstein e Giryes (2020) como um tipo específico de rede neural que codifica a entrada em uma representação comprimida e significativa. O AE é usado para reduzir a dimensão dos dados de forma a manter somente as características mais relevantes. É composto por dois módulos, o *codificador* e o *decodificador*, que incluem uma ou mais camadas totalmente conectadas. Para efetuar tal codificação dos dados de entrada, o AE conta com a *camada codificadora*, que consiste na última camada do módulo codificador. Esta camada se comporta como um gargalo, forçando os dados a serem comprimidos, produzindo, assim, sua representação codificada. O módulo decodificador do AE é utilizado para efetuar os reajustes dos parâmetros de sua rede neural durante o treinamento. Tal reajuste é baseado na técnica de Reconstrução dos Dados (HINTON; OSINDERO; TEH, 2006). Na referida técnica, o algoritmo de treinamento tenta reconstituir o dado de entrada da seguinte forma: ele propaga o sinal produzido pelo módulo codificador na entrada do módulo decodificador; o valor da discrepância entre a saída do módulo decodificador (representação reconstruída) e a entrada do módulo codificador (representação original), que corresponde à taxa de perda (que, conforme a Equação 2.3, é calculada a partir do Erro Médio Quadrático (EMQ), expressado pela Equação 2.4), é, então, usado como base para calcular os valores dos reajustes a serem feitos nos parâmetros de ambos os módulos do AE. A Figura 2.4 ilustra a arquitetura de um AE com uma camada escondida.

Considerando a etapa de treinamento de um AE simples com somente uma camada escondida, cada codificação h gerada pelo codificador $f(x)$ é introduzida ao decodificador $g(h)$, que a reconstrói em uma representação y. A relação entre os módulos é descrita pelas Equações 2.1 e 2.2,

$$f(x) = s_f(Wx + b_h) \equiv h \tag{2.1}$$

$$g(h) = s_g(Uh + b_y) \equiv y, \tag{2.2}$$

onde:

x representa o vetor dos dados de entrada;

s_f e s_g são as funções de ativação;

W e U correspondem aos conjuntos de pesos sinápticos, respectivamente, do codificador e do decodificador;

b_h e b_y correspondem aos conjuntos de vieses, respectivamente, do codificador e do decodificador;

Seu treinamento consiste em encontrar um conjunto de parâmetros $\theta = \{W, b_h, U, b_y\}$ que minimize a taxa de perda de reconstrução correspondente ao resultado da função de perda calculada pela Equação 2.3, a qual avalia a discrepância entre as representações reconstruída e original. Tal resultado corresponde ao valor do reajuste a ser aplicado nos parâmetros da rede neural. Em seguida, realiza-se o processo de *backpropagation*, propagando a referida taxa pelos conjuntos de pesos sinápticos, como descrito na seção 2.2.

$$\mathcal{J}_{AE}(\theta) = \sum_{x \in D_n} L(x, g(f(x))), \tag{2.3}$$

onde D_n corresponde ao conjunto de dados de entrada.

Usualmente, a função L toma forma da função do EMQ, cuja definição é

$$L_{mse}(x, y) = \frac{\|x - y\|^2}{N}, \tag{2.4}$$

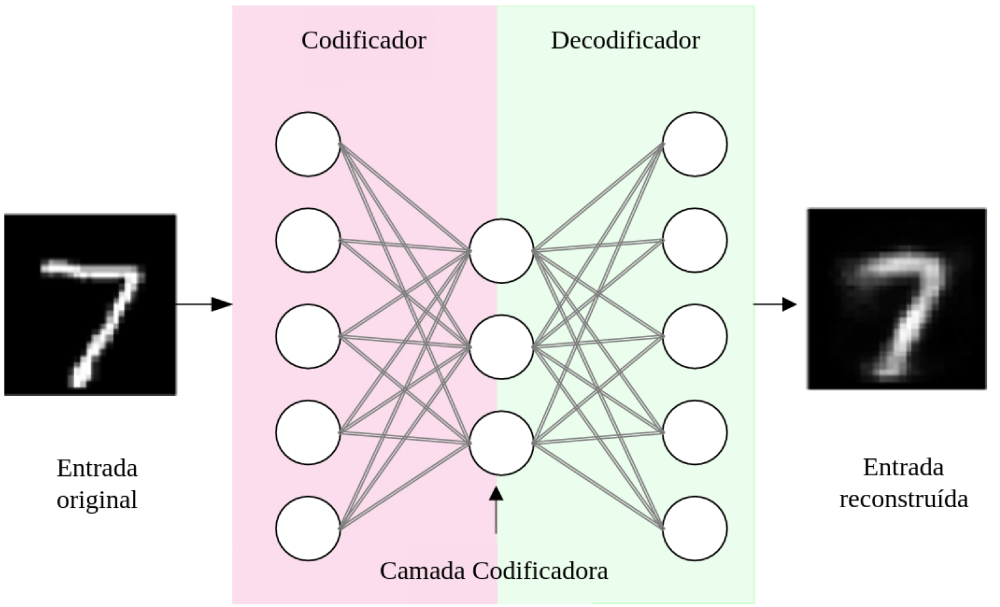
onde:

N é o número total de amostras;

x corresponde à resposta desejada;

y corresponde à resposta obtida;

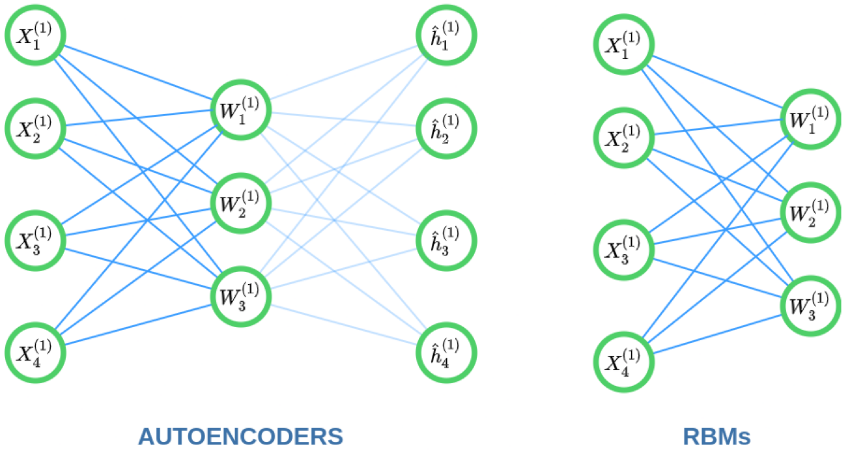
Figura 2.4 – Ilustração de um AE com uma camada escondida



Fonte: Autor[2].

Convém salientar que o processo de treinamento do AE, que é baseado na técnica de reconstrução de dados, é semelhante ao usado na *Restricted Boltzmann Machine* (RBM) (SMOLENSKY, 1986). A diferença essencial em tal processo é que, nesta última, há apenas o módulo de codificação, sendo a reconstrução dos dados feita a partir de uma retropropagação do sinal de saída (representação codificada do dado) em direção à entrada da rede. Neste caso, o erro será calculado a partir da diferença entre o resultado de tal retropropagação e o dado de entrada apresentado.

Figura 2.5 – Arquiteturas de um RBM e de um AE



Fonte: (DEB, 2020)

Um dos principais dilemas que envolvem os AEs é o dilema de viés-variância. Por um lado, deseja-se que a arquitetura seja capaz de reconstruir os dados de entrada eficientemente, isto é, minimizar o erro entre a reconstrução e a entrada, tornando-as similares. Por outro lado, também se deseja que a representação codificada generalize para uma representação significativa.

Na literatura, existem diversas arquiteturas distintas de AE que lidam com tais desafios. Dentre elas, o CAE (RIFAI *et al.*, 2011) é relevante para este trabalho.

CAE

O CAE introduz uma função de perdas, responsável por calcular o reajuste de pesos sinápticos da rede, denominada *função de contração*. A função de contração efetua um balanceamento entre os seguintes fatores: fator de reconstrução e fator de regularização. O fator de reconstrução se refere ao EMQ do AE apresentado pela Equação 2.4. O fator de regularização é calculado com base na medida da *sensitividade*. Esta última, por sua vez, corresponde ao quadrado da norma de Frobenius da Jacobiana $J_f(x)$, que é calculada na camada codificadora da seguinte forma: inicialmente a entrada $x \in \mathbb{R}^{d_x}$ é mapeada na representação escondida $h \in \mathbb{R}^{d_h}$ pela função $f(x)$; em seguida, o valor da sensitividade é obtido pela somatória do quadrado das derivadas

[2] Traduzida a partir de uma figura obtida de Pawar e Attar (2020).

parciais das saídas do codificador, que compõem a representação codificada, conforme definido pela Equação 2.5.

$$\|J_f(x)\|_F^2 = \sum_{ij} \left(\frac{\partial h_j(x)}{\partial x_i} \right)^2, \tag{2.5}$$

onde:

i corresponde a um iterador do conjunto de dados da entrada x;

j corresponde a um iterador do conjunto de elementos da representação codificada h;

Na prática, o fator de regularização, por meio da sensitividade, traz as seguintes contribuições ao processo de reajustes do CAE: ele evita que a ocorrência de ruídos nos dados de entrada apresentados à rede neural produza uma distorção na codificação retornada; e ele não permite que variações insignificantes entre os dados de entrada apresentados à rede produzam codificações distintas. Entretanto, o fator de regularização pode produzir o indevido efeito colateral de gerar codificações iguais para dados de entrada que, apesar de muito semelhantes, apresentam pequenas e relevantes diferenças entre si. A função de contração, então, compensa esse efeito colateral por meio do fator de reconstrução (ou EMQ), conforme mostrado na Equação 2.6.

$$\mathcal{J}_{CAE}(\theta) = \sum_{x \in D_n} \left(L_{mse}(x, g(f(x))) + \lambda \|J_f(x)\|_F^2 \right), \tag{2.6}$$

onde:

λ corresponde a uma taxa de ajuste com valores variando entre 1×10^{-2} e 1×10^{-4};

$\lambda \|J_f(x)\|_F^2$ consiste no fator de regularização;

$L_{mse}(x, g(f(x)))$ consiste no fator de reconstrução;

D_n equivale ao conjunto de dados de entrada;

θ corresponde ao conjunto de parâmetros da rede CAE;

2.3.2 PNet

A PNet consiste em um PMC, que tem como objetivo gerar, na medida em que solicitado pelo sistema planejador MPNet, um novo estado crítico.

Seu treinamento é realizado através da estratégia de previsão do próximo passo. Para tal, sua entrada corresponde a um conjunto formado pela representação codificada do cenário Z, pelo estado atual do agente s_t e pelo estado destino s_T. Sua saída corresponde ao estado $\hat{s}_{t+1}$, que se torna um estado candidato a compor a trajetória. Para realizar o reajuste de pesos sinápticos, a

função de perda utilizada é a EMQ, de modo que tal operação é realizada com o estado candidato $\hat{s}_{t+1}$ e o estado rotulado s_{t+1}, conforme descrito na Equação 2.7,

$$\mathcal{J}_{PNet}(\theta) = \frac{1}{N_p} \sum_{j}^{\hat{N}} \sum_{i}^{T-1} \left\| \hat{s}_{j,i+1} - s_{j,i+1} \right\|^2 , \tag{2.7}$$

onde:

i e j correspondem a iteradores, respectivamente, do conjunto de T estados críticos de uma trajetória e do conjunto composto por $\hat{N}$ trajetórias (σ).

T corresponde ao comprimento de uma trajetória σ_j;

$\hat{N}$ corresponde ao número total de trajetórias;

N_p equivale ao número total de amostras de treinamento ($N_p = \hat{N} \times T$) da PNet;

θ corresponde a um conjunto de parâmetros composto pelos conjuntos de pesos sinápticos e vieses da rede;

2.3.3 Treinamento utilizando método de aprendizado em lotes *offline*

Nesta abordagem, a ENet e a PNet podem ser treinadas separadamente ou em conjunto. Para realizar seu treinamento em conjunto, o gradiente da função de perda da PNet é propagado através de ambas as redes. O presente trabalho realizou o treinamento das redes separadamente, de modo a analisar seus resultados isoladamente.

A efetuação desta metodologia de treinamento das redes neurais presentes no MPNet utilizando esse tipo de aprendizado requer que o conjunto total de dados de treinamento esteja disponível antes do treinamento ser iniciado. Para gerar tal conjunto de dados, utilizam-se representações do ambiente, conjuntos de pares compostos pelos estados atual e de destino para cada ambiente, além de uma trajetória gerada por um algoritmo especialista para cada um destes pares, que será usada no processamento da base de dados de treinamento da PNet. Após a conclusão do treinamento, os parâmetros treinados de cada rede neural não são mais modificados. A seção 4.2 detalha o procedimento utilizado na geração dos dados utilizados nas bases de dados de treinamento.

2.3.4 Estratégia de planejamento bidirecional do MPNet

O planejamento bidirecional tem como objetivo gerar um conjunto de estados críticos candidatos a pertencer à trajetória. Utiliza-se a PNet na geração destes estados críticos.

A estratégia de planejamento bidirecional envolve o planejamento alternado de duas trajetórias, σ_1 e σ_2. Considerando que foi requisitado o planejamento de uma trajetória entre o

estado atual s_1 e o estado destino s_{obj}, a trajetória σ_1 é planejada de modo que a posição atual do agente seja s_1 e seu destino seja s_{obj}, enquanto que, σ_2 é planejada com o sentido oposto, ou seja, sua posição atual equivale a s_{obj} e seu destino a s_1. O estado crítico que descreve a posição atual no planejamento da trajetória σ_t, onde $t \in \{1, 2\}$, é denominado $s_{\sigma_t}^{atual}$. Cada iteração do planejador bidirecional para uma trajetória σ_t requisita à PNet a geração de um novo estado crítico, que passa a representar a posição atual do agente para esta trajetória ($s_{\sigma_t}^{atual}$). Ao final de cada iteração ocorre a conexão do par de estados críticos $s_{\sigma_1}^{atual}$ e $s_{\sigma_2}^{atual}$. Caso tal conexão seja válida, conclui-se o planejamento bidirecional, resultando em um conjunto de estados críticos gerado pela concatenação de σ_1 e de σ_2 com a ordem dos estados críticos invertida (isto é necessário para que o estado destino seja o último estado da trajetória). Do contrário, alterna-se a trajetória a ser planejada, utilizando $s_{\sigma_t}^{atual}$ como novo estado atual e $s_{\sigma \neq \sigma_t}^{atual}$ como novo estado destino. Este processo é detalhado pelo pseudocódigo apresentado na subseção 4.3.3.

2.3.5 Algoritmo de conexão de estados críticos

O algoritmo de conexão de estados de críticos tem como finalidade conectar um par de estados críticos, verificando a validade de tal conexão. A conexão corresponde a uma linha reta traçada entre tal par de estados críticos. A validação desta conexão ocorre por meio da verificação da existência de colisão da linha traçada com algum obstáculo, de modo que a ocorrência desta colisão resulta em uma conexão inválida. A Figura 2.6b mostra a conexão de estados críticos efetuada a partir da sequência otimizada de estados candidatos a pertencer à trajetória.

Além da aplicação mencionada na estratégia de planejamento bidirecional, o algoritmo de conexão de estados críticos também é utilizado pelo algoritmo otimizador, auxiliando-o a encontrar pares de estados críticos que resultem na otimização da trajetória. Por fim, ele é usado na checagem da validade da trajetória otimizada, posto que uma trajetória é válida somente se todas as conexões entre seus estados críticos consecutivos também o são.

2.3.6 Otimizador de trajetórias

O otimizador de trajetórias utiliza o algoritmo *Lazy States Contraction* (LSC) (HAUSER; NG-THOW-HING, 2010) para reduzir a quantidade de estados críticos que compõem uma trajetória, assim otimizando-a. A seguir, detalha-se o processo realizado na otimização da trajetória. Devido à dificuldade em se fazer uma primeira apresentação em linguagem natural deste algoritmo que seja clara e suficiente (tal como foi feito para os anteriores), optou-se por introduzi-lo nesta seção diretamente por meio de pseudocódigo.

O algoritmo otimizador de trajetórias consiste em um processo iterativo controlado por meio de dois iteradores, denominados i_1 e i_2. Considerando uma sequência de estados críticos candidatos à trajetória σ: $(s_1, s_2, \ldots, s_{n-1}, s_n)$, inicializam-se os valores de i_1 e i_2, a princípio, com os estados s_1 e s_n, respectivamente. Em seguida, executa-se, iterativamente, o seguinte

procedimento:

Algoritmo 2.1: Algoritmo de otimização de um sequência de estados críticos candidata à trajetória

Entrada : A sequência de estados candidata à trajetória σ: $(s_1, \ldots, s_{n-1}, s_n)$
Saída : A sequência de estados otimizada resultante σ

```
1  Function Otimizador(σ):
2      while i_1 ≠ s_n do
3          Tenta-se conectar i_1 com i_2
4          if conexão válida then
5              Elimina-se da sequência dos estados críticos todos os estados compreendidos
                entre i_1 e i_2
6              i_1 ← i_2
7              i_2 ← s_n
8          else
9              Recua-se i_2 para o estado anterior ao representado por ele atualmente
10         end
11         if i_2 = i_1 then
12             Avança-se i_1 para o estado posterior ao representado por ele atualmente
13             Recua-se i_2 para s_n
14         end
15     end
16     σ ← σ \ σ'
17     return σ
```

2.3.7 Replanejador de trajetórias

O replanejamento de trajetórias é realizado quando a trajetória obtida após a otimização não é válida, o que é decidido pelo algoritmo de conexão de estados críticos. Em Qureshi, Miao *et al.* (2019) são propostas duas técnicas para efetuar o replanejamento: o Replanejamento Neural (RN) e o Replanejamento Híbrido (RH).

O RN é um algoritmo que executa o replanejamento até encontrar uma trajetória válida ou atingir o limite N_{rp} de tentativas. O replanejamento inicia com a remoção dos estados críticos inválidos[3] da trajetória. Então, itera-se sobre o conjunto de pares de estados críticos consecutivos da trajetória resultante, de modo que os pares que resultarem em uma conexão válida são preservados, e os que resultarem em uma conexão inválida são introduzidos no planejador bidirecional como estados atual e destino. O conjunto de novos estados críticos gerados pelo

[3] Um estado crítico é considerado como válido se não ocupar um estado que é ocupado por um obstáculo.

planejador bidirecional são incluídos na trajetória entre seus respectivos estados atual e destino. Um pseudocódigo que descreve o replanejamento é apresentado na subseção 4.3.5.

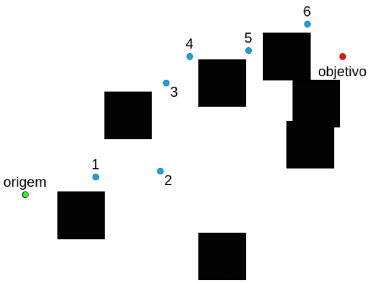
O RH envolve a adição de um planejador oráculo que é acionado caso o RN resulte em falha. O planejador oráculo pode ser qualquer planejador de trajetórias capaz de solucionar o problema proposto.

2.3.8 Atuação conjunta de todos os elementos do MPNet

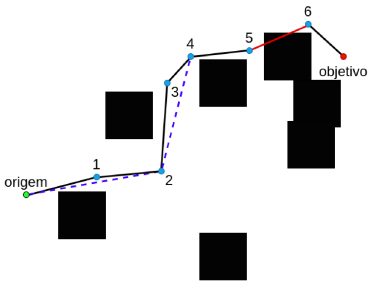
O planejador MPNet é formado pela atuação conjunta dos elementos descritos nas subseções anteriores. O processo executado na geração de uma trajetória que conecte dois estados em um ambiente com obstáculos estáticos ocorre da seguinte maneira: é gerada uma sequência de estados críticos candidata à trajetória por meio da estratégia de planejamento bidirecional; essa sequência de estados é otimizada, eliminando estados desnecessários; é verificado se a sequência de estados otimizada resulta em uma trajetória válida após a conexão dos estados críticos consecutivos nela presentes; caso a trajetória gerada seja válida, ela se torna a trajetória resultante e o processo é finalizado; caso ela seja inválida, é efetuado o replanejamento, algoritmo que busca resgatar trechos da trajetória que geraram uma conexão inválida; novamente se verifica a validade da trajetória, resultando em uma trajetória final, caso seja válida, seguida da finalização do planejamento, e, caso seja inválida, em uma nova tentativa de replanejamento (são realizadas no máximo N_{rp} tentativas); caso o limite de tentativas seja esgotado sem que uma trajetória válida seja encontrada, o MPNet pode recorrer ao RH, se o uso desta técnica não estiver disponível para a aplicação, o planejamento da trajetória falha. Este processo é descrito por um pseudocódigo apresentado na subseção 4.3.6.

A Figura 2.6 ilustra o processo completo de geração de uma trajetória, desde a geração dos estados críticos até a conclusão do replanejamento.

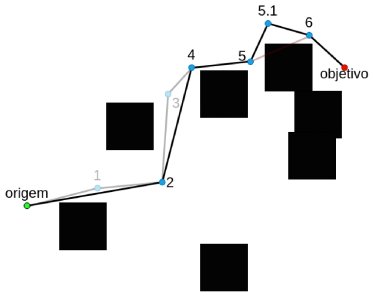
Figura 2.6 – Procedimento de geração de trajetórias seguido pelo MPNet



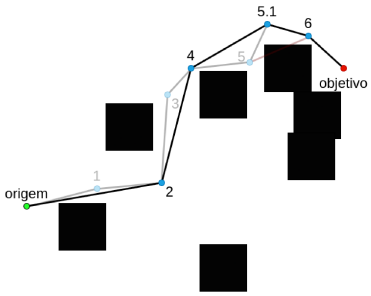
(a) Início do planejamento com estados críticos definidos.



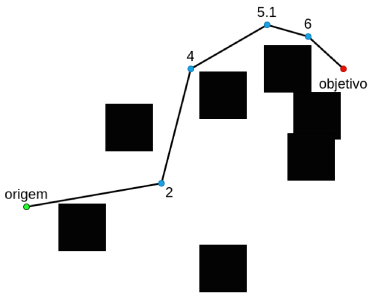
(b) São encontradas as otimizações possíveis na trajetória, descritas pelas linhas roxas tracejadas, e identifica-se uma colisão por meio da conexão dos estados candidatos, descrita por uma linha vermelha.



(c) Com a trajetória otimizada, é processado um novo estado crítico a partir dos pontos de colisão e este é conectado a trajetória gerada anteriormente.



(d) Realiza-se uma nova otimização da trajetória replanejada.



(e) Trajetória final otimizada.

Fonte: Autor

Neste exemplo, a Figura 2.6a mostra o conjunto de estados críticos candidatos a pertencer à trajetória obtidos utilizando a estratégia de planejamento bidirecional. Tal processo é iniciado por meio do planejamento da trajetória σ_1 a partir da introdução dos estados *origem* e *objetivo* à PNet, respectivamente como estado atual e estado destino. O processamento realizado pela PNet resulta no estado 1 ($s_{\sigma_1}^{atual}$). É efetuada a conexão do estado 1 ao estado *objetivo* ($s_{\sigma_2}^{atual}$), e conclui-se que tal conexão não é válida. Assim sendo, uma nova iteração é iniciada, a trajetória σ_2 vira a trajetória a ser planejada, seu estado atual passa a ser o estado *objetivo* e o estado destino se torna o estado 1. Tais estados são introduzidos à PNet, que gera o estado 6 ($s_{\sigma_2}^{atual}$). Novamente a conexão entre os estados $s_{\sigma_2}^{atual}$ e $s_{\sigma_1}^{atual}$, que correspondem, respectivamente, aos estados 6 e 1, é inválida. Portanto, a trajetória a ser planejada volta a ser σ_1, para qual o estado atual equivale ao estado 1 e o destino equivale ao estado 6. Esse processo se repete até que o estado 4 é gerado, neste caso sua conexão com o estado 3 é válida, e portanto o planejamento bidirecional é finalizado.

Em seguida, é efetuada a otimização da sequência de estados críticos candidata à trajetória, ilustrada na Figura 2.6b. No início de tal processo, os iteradores i_1 e i_2 são inicializados, respectivamente, nos estados *origem* e *objetivo*. Então, são efetuadas conexões entre tais iteradores, recuando i_2 em caso de conexões inválidas, até que seja obtida uma conexão válida, o que ocorre quando i_2 coincide com o estado 2. Neste caso, o estado 1 é eliminado da sequência de estados críticos candidata à trajetória. O processo se repete, de modo que i_1 represente o estado 2 e i_2 retorna ao estado *objetivo*. Nesta iteração, o estado 3 é eliminado da sequência. Novamente os iteradores são atualizados, com i_1 representando o estado 4 e i_2 retornando ao estado *objetivo*. Esta iteração não resulta em estados eliminados. Os iteradores voltam a ser atualizados, com i_1 representando o estado 5, que não é conectável com nenhum dos estados posteriores. Deste modo, conclui-se que tal sequência de estados críticos candidata à trajetória não é válida, porém o processo de otimização não é interrompido, e resulta em uma iteração que conecta o estado 6 (que i_1 representa) ao estado *objetivo* (que i_2 representa). Na iteração seguinte, i_1 representa o estado *objetivo*, o que encerra o processo de otimização.

O algoritmo de conexão de estados críticos, então, conecta a sequência de estados críticos resultantes da otimização, de modo que são conectados cada par de estados críticos consecutivos. A Figura 2.6b ilustra uma trajetória inválida, devido à conexão com um obstáculo entre os estados 5 e 6.

Como a candidata à trajetória gerada pelo algoritmo otimizador não é válida, o replanejamento é necessário. Tal processo identifica que não há uma conexão válida entre os estados 5 e 6, conforme ilustrado pela Figura 2.6b. Então, os demais estados da trajetória são mantidos e executa-se a geração de estados críticos candidatos a pertencer à trajetória entre os estados 5 e 6. Tal processo, ilustrado pela Figura 2.6c resulta no estado 5.1, que, por sua vez, é conectável tanto com o estado 5 quanto com o estado 6.

Por fim, o otimizador de trajetórias é executado novamente, desta vez incluindo o estado 5.1, resultando na trajetória válida ilustrada na Figura 2.6d. Neste exemplo, a trajetória final resultante é ilustrada pela Figura 2.6e.

3 Trabalhos Relacionados

Este capítulo aborda os principais trabalhos correlatos que influenciam na execução da presente pesquisa no campo de planejamento de trajetórias. Para tanto, são descritas abordagens que solucionam problemas similares ao do presente trabalho ou que utilizam abordagens similares em problemas diferentes. Tais abordagens são divididas em PBAs e planejadores baseados em AM.

Planejadores baseados em Amostragem

Os PBAs se dividem em planejadores de consulta múltipla, como o *Probabilistic Roadmap* (PRM) (LATOMBE; KAVRAKI, 1998), e em planejadores de consulta única, como o RRT (LAVALLE, 1998).

Métodos baseados no PRM realizam um pré-processamento denominado de fase de construção, na qual uma árvore é expandida por meio da geração de uma grande quantidade de nós aleatoriamente distribuídos pelo ambiente, o que tem alto custo em seu desempenho. É possível substituir métodos de consulta múltipla por várias requisições a métodos de consulta única, permitindo que planejadores de consulta única se tornem ferramentas promissoras para o planejamento de trajetórias.

O RRT expande uma árvore, que tem como raiz o estado inicial da trajetória, por meio da geração de nós distribuídos aleatoriamente pelo espaço de busca. Depois que um nó é gerado, ocorre a tentativa de conectá-lo ao nó vizinho mais próximo, o novo nó é, então, adicionado à árvore se a referida conexão não resultar em colisão com obstáculos. O planejamento é encerrado quando ocorre a conexão com o estado destino da trajetória ou quando o limite de nós é alcançado. Apesar do RRT encontrar uma solução de maneira eficiente, não há garantia de que a trajetória encontrada seja ótima. Para solucionar tal limitação, LaValle (2006) desenvolveu o RRT*, que introduz o cálculo de distância (custo) entre cada nó e a raiz da árvore, bem como o procedimento de reconexão de nós, que ocorre quando a conexão de um nó gerado com um de seus vizinhos resultaria em uma trajetória com menor custo. Deste modo, o RRT* garante que uma trajetória ótima seja encontrada quando o número de nós gerados tenda ao infinito. Entretanto, seu desempenho se torna mais lento à medida que a dimensão do ambiente aumenta. Vários métodos foram desenvolvidos buscando mitigar esse fator, como a amostragem enviesada (GAMMELL; SRINIVASA; BARFOOT, 2015), a avaliação preguiçosa de arestas (HAGHTALAB *et al.*, 2017) e a geração bidirecional de árvores (QURESHI; AYAZ, 2015).

PBAs com amostragem enviesada realizam adaptativamente a amostragem do espaço de estados para superar limitações causadas pela exploração aleatória dos métodos anteriores. O *Informed-RRT** (GAMMELL; SRINIVASA; BARFOOT, 2014) define uma região elipsoidal

utilizando uma trajetória inicial encontrada pelo RRT*, de modo que a região de amostragem seja limitada, reduzindo o espaço de estados a ser investigado. A área da referida região elipsoidal é reduzida até que seja encontrada a trajetória ótima. No entanto, por depender do RRT* na geração da trajetória inicial influencia no seu desempenho quando encontrá-la é desafiador. A partir desta abordagem, Gammell, Srinivasa e Barfoot (2015) propõem o *Batch Informed Trees*, um algoritmo de busca incremental de grafos que usa uma região elipsoidal que se altera dinamicamente para gerar lotes de amostragens e assim computar possíveis trajetórias. Apesar de apresentarem melhorias na velocidade de processamento em relação ao RRT* quando buscada uma trajetória próxima à ótima, ambos os métodos continuam ineficientes em ambientes de grandes dimensões.

Métodos de avaliação preguiçosa de arestas demonstram melhorias significativas na velocidade de processamento de PBAs ao avaliá-las somente ao longo de soluções em potencial. Entretanto, tal abordagem é dependente do algoritmo seletor de arestas e exibem um desempenho limitado em alguns ambientes (HOU; SRINIVASA, 2018). Métodos de geração bidirecional de árvores apresentam melhorias quando o ambiente possui passagens apertadas, porém as limitações dos PBAs são mantidas (QURESHI; AYAZ, 2015).

Planejadores baseados em Aprendizado de Máquina

Planejadores baseados em AM têm sido utilizados tanto como soluções independentes quanto como auxiliares para PBAs. Uma abordagem com *Conditional Variational Autoencoders* (ICHTER; HARRISON; PAVONE, 2017) foi proposta com o objetivo de contextualizar as informações do ambiente, gerando amostras codificadas para PBAs. Bhardwaj, Choudhury e Scherer (2017) propõem um método que aprende uma política determinística para guiar a expansão da árvore do planejador para regiões que potencialmente contém uma solução.

Wen *et al.* (2020) propõem um algoritmo com a capacidade de desviar tanto de obstáculos fixos quanto obstáculos móveis em ambientes desconhecidos. Para tanto, os autores utilizam o método *FastSLAM* para criar um mapa bidimensional do ambiente, redes residuais no reconhecimento de obstáculos e, por fim, o algoritmo *Dueling Deep Q-Network* para realizar o planejamento do desvio de obstáculos. Yu, Su e Liao (2020) propõem a junção de redes neurais e de aprendizado por reforço hierárquico, assim obtendo melhorias no desempenho do planejador e na sutileza da trajetória quando comparado a outras soluções que usam *Q-Learning*. Tamar *et al.* (2017) apresentam as *Value-Iteration Networks*, uma abordagem que considera que o algoritmo de planejamento *Value-iteration* (BERTSEKAS, 2005) pode ser representado por uma rede neural convolucional. Então, ao anexar um módulo de planejamento em uma rede neural classificadora, obtém-se um modelo capaz de aprender os parâmetros de planejamento e de entregar previsões válidas.

Qureshi, Miao *et al.* (2019) propõem e aplicam o MPNet, abordagem na qual o presente estudo se baseia, detalhada na seção 2.3. Tal abordagem pode planejar trajetórias em ambientes variados, como ambientes bidimensionais e tridimensionais, tanto com obstáculos que possuem a

mesma forma e tamanho quanto com formas e tamanhos variados, sendo aplicável para diversos tipos de agentes, como corpos rígidos e braços mecânicos. As vantagens de tal abordagem são a velocidade para realizar o planejamento e a flexibilidade tanto em acoplar outros planejadores para aprimorar seu desempenho quanto em ser acoplado a outros planejadores.

4 Geração das bases de dados e implementação do MPNet

Este capítulo detalha os procedimentos realizados para implementar os objetivos específicos do presente trabalho (descritos na seção 1.2). A seguir, está disposto um resumo contendo a finalidade de cada seção presente neste capítulo:

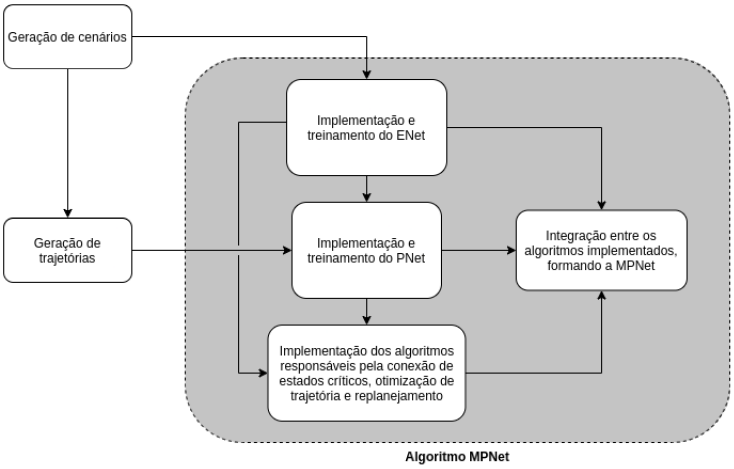
- Relação entre os objetivos específicos: apresenta a relação entre cada objetivo específico;
- Aquisição e processamento de dados para treinamento das redes neurais: descreve o procedimento utilizado na geração de dados de cenários e de trajetórias e no processamento de tais dados para serem utilizados no treinamento da ENet e da PNet;
- Detalhamento dos algoritmos que compõem o MPNet: descreve a implementação dos algoritmos que compõe o planejador MPNet e a sua integração;

4.1 Relação entre os objetivos específicos

Os objetivos específicos, descritos na seção 1.2, representam etapas necessárias para a implementação e execução da abordagem aqui proposta. Entre eles existem relações que determinam suas dependências e, portanto, sua ordem de implementação. Estas relações são ilustradas pela Figura 4.1.

Primeiramente é necessário implementar o gerador de cenários. Estes cenários serão utilizados na geração de trajetórias, no treinamento da ENet e na avaliação do MPNet. O próximo passo é a geração de trajetórias, que servirão de referência no treinamento da PNet.

Figura 4.1 – Relações entre objetivos específicos



Fonte: Autor[1]

Em seguida, a ENet é implementada e treinada utilizando uma base de dados criada a partir dos cenários gerados. Com a ENet treinada e as trajetórias geradas, cria-se a base de dados necessária para o treinamento da PNet, possibilitando sua implementação e treinamento. Na sequência, são implementados os algoritmos que compõem o MPNet, que são o planejador bidirecional, os algoritmos de conexão de estados críticos, de otimização de trajetória e o replanejador. Por fim é realizada a integração dos algoritmos anteriormente implementados, formando o MPNet. Essas etapas são detalhadas nas seções 4.2 e 4.3.

4.2 Aquisição e processamento de dados para treinamento das redes neurais

Esta seção visa detalhar o procedimento utilizado na aquisição de dados. Sua importância se origina da necessidade de delimitar as informações do cenário utilizadas pelo agente e os parâmetros que o algoritmo é capaz de processar, bem como para auxiliar na reprodução da presente pesquisa. É dividida em quatro subseções:

- Construção dos cenários para coleta de dados;
- Criação da base de dados utilizada no treinamento da ENet;
- Coleta de dados de trajetórias geradas pelo planejador especialista;
- Criação das bases de dados utilizadas no treinamento da PNet;

Estas subseções tratam, respectivamente: da geração de cenários; da criação das bases de dados a serem utilizadas no treinamento da ENet a partir dos cenários gerados; da geração de

[1] As setas descrevem etapas originárias das quais dependem as etapas destinatárias.

trajetórias; e, por fim, da criação das bases de dados a serem utilizadas no treinamento da PNet a partir das trajetórias geradas e dos cenários codificados pela ENet;

4.2.1 Construção dos cenários para coleta de dados

Para que ocorra convergência no treinamento da ENet, os cenários gerados não devem variar de maneira drástica entre si. Assim sendo, o algoritmo gerador implementado segue um padrão no posicionamento dos obstáculos.

Neste trabalho, cada cenário A_a corresponde a um quadrado de lado $D = 40\,\mathrm{m}$ contendo sete obstáculos quadrados de lado $L = 5\,\mathrm{m}$. Cada um desses obstáculos será alocado no cenário de forma que seu centro ocupe uma posição cartesiana definida por um par (i, j), denominado $estado$, onde $i, j \in \mathbb{R}\colon \left[-\frac{D}{2}, \frac{D}{2}\right]$. Cada obstáculo será alocado de forma que seu centro ocupe um estado s distinto dos centros dos demais obstáculos, no entanto, é possível que partes destes obstáculos se superponham, ou seja, mais de um obstáculo pode ocupar o mesmo estado. Os estados preenchidos por obstáculos determinam o conjunto $A_{obs} \subset A_a$, enquanto que o espaço livre de obstáculos é determinado por $A_{free} = A_a \setminus A_{obs}$.

Assim sendo, um conjunto de 20 estados $O_C\colon (s_1, s_2, \ldots, s_{20})$ é definido, correspondendo aos possíveis centros dos obstáculos. Em seguida, realiza-se uma combinação sem repetições destes 20 estados em grupos de sete elementos, resultando em um conjunto com $\binom{20}{7}$ combinações possíveis. A construção de um cenário A_a se inicia com a escolha aleatória de um destes grupos a fim de definir os centros dos sete obstáculos presentes nos cenários. Após definida a posição central de cada obstáculo, os estados referentes à área ocupada por cada obstáculo são preenchidos, de modo que não haja rotação e respeitando sua dimensão.

4.2.2 Criação da base de dados utilizada no treinamento da ENet

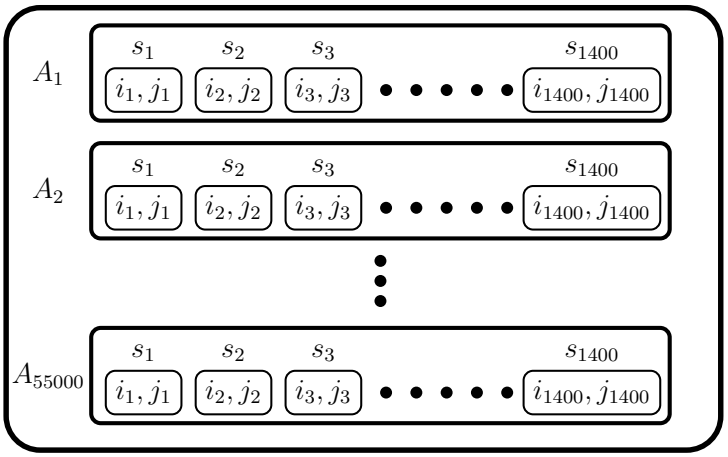
O treinamento da ENet requer uma grande quantidade de cenários para que ele seja capaz de generalizar para cenários similares desconhecidos. Portanto, foram construídos 55000 cenários, dentre eles 11000 cenários são usados como dados de validação, isto é, são usados apenas na fase de testes da rede treinada, não sendo usados no treinamento propriamente dito. Como a arquitetura utilizada na ENet consiste em um CAE, as respostas desejadas são equivalentes aos dados de entrada. A Figura 4.2 apresenta uma amostra dos cenários gerados pelo presente trabalho, onde os pontos azuis representam os estados ocupados por obstáculos.

Figura 4.2 – Amostras de cenários gerados para a base de dados da ENet

Fonte: Autor

A base de dados da ENet é composta por um conjunto de obstáculos A_{obs} extraído de cada cenário disponível. A fim de reduzir o custo computacional no carregamento dos dados e o número de parâmetros a serem treinados pela rede neural, são selecionados aleatoriamente 200 estados de cada obstáculo, de modo que cada exemplo da base de dados da ENet seja composto por 1400 estados, resultando em 2800 unidades de dados (uma vez que cada estado é representado por um par de coordenadas cartesianas). Não é realizada a normalização destes dados, o que permite a utilização de cenários com dimensões variadas e desconhecidas. Define-se a codificação gerada pelo CAE como $Z_a : (c_1, ..., c_N)$, onde N é determinado pelo número de unidades de saída da camada de codificação. A Figura 4.3 apresenta o formato dos dados de entrada da ENet, onde cada cenário A_a é composto por um conjunto de 1400 estados s, e cada estado é definido pelas coordenadas (i, j).

Figura 4.3 – Base de dados processada para o treinamento da ENet



Fonte: Autor

4.2.3 Coleta de dados de trajetórias geradas pelo planejador especialista

O objetivo da PNet é processar os dados de entrada gerando um estado crítico candidato a pertencer à trajetória que conecta o estado atual e o estado destino. Para perfazê-lo, é necessário que um planejador especialista gere trajetórias completas, que então são utilizadas na criação da base de dados utilizada no treinamento da PNet. Os cenários gerados na etapa descrita pela subseção 4.2.1 são aproveitados.

Neste trabalho, utilizou-se o algoritmo RRT* como planejador especialista. Este planejador explora o ambiente gerando aleatoriamente novos estados, de modo que são priorizados aqueles que minimizem o custo da trajetória, até que seja encontrada uma trajetória válida. Então, o RRT* busca novos estados com o objetivo de otimizar a trajetória até atingir o limite de iterações determinado. Particularmente, foi utilizado um limite de 10000 iterações para o planejamento de cada trajetória. Este valor foi utilizado por ser o suficiente para se planejar trajetórias próximas à ótima sem acarretar em um aumento do tempo de processamento (utilizando tal valor, o tempo médio de planejamento de uma trajetória foi em torno de três segundos).

Para o planejador RRT*, foi empregado o algoritmo disponibilizado por Qureshi, Miao *et al.* (2019)[2]. A partir os cenários gerados previamente, são gerados aleatoriamente pares de estados pertencentes a A_{free}, que serão usados na definição de estado atual e de estado destino em cada trajetória.

4.2.4 Criação das bases de dados utilizadas no treinamento da PNet

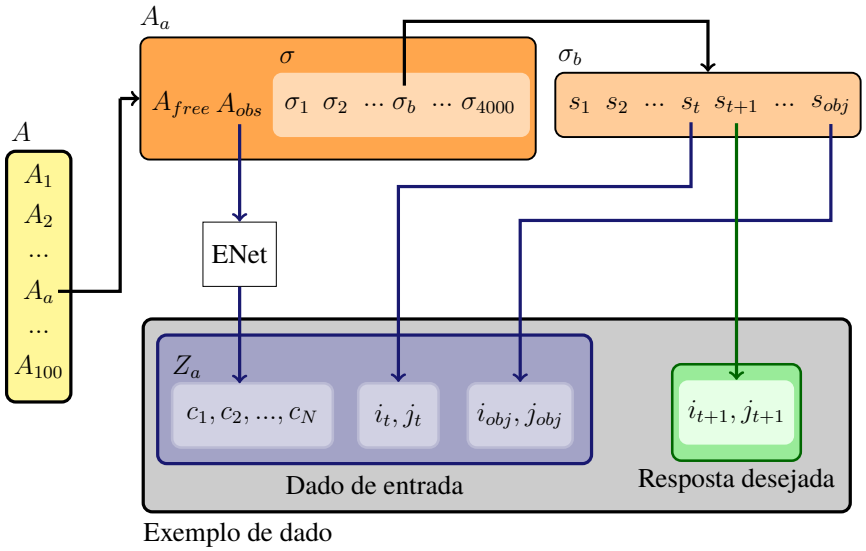
A criação das bases de dados utilizadas no treinamento da PNet depende das trajetórias de referência geradas pelo planejador especialista e da codificação do cenário gerada pela ENet

[2] Materiais suplementares, parâmetros de implementação, códigos e vídeos do projeto disponibilizados por Qureshi, Miao *et al.* (2019) em: `https://sites.google.com/view/mpnet/home`

após o seu treinamento. Define-se uma trajetória de referência como $\sigma\colon (s_1, ..., s_{obj})$, onde s_1 representa o estado atual e s_{obj} representa o estado destino. Supondo que σ seja composta por n elementos, define-se $t \in \mathbb{Z}\colon [1, ..., n]$, de modo que a notação s_t descreva o t-ésimo estado de σ.

Considerando uma trajetória σ_b produzida para um cenário A_a, um dado de entrada da PNet é formado por uma representação codificada Z_a, gerada pela ENet a partir de A_a, e pelos estados s_t e s_{obj} pertencentes à trajetória σ_b. A resposta desejada (rótulo) para tal dado de entrada é formada pelo estado s_{t+1} de σ_b. A Figura 4.4 ilustra essa relação.

Figura 4.4 – Processamento de um dado de entrada da PNet



Fonte: Autor

A base de dados de treinamento da PNet é gerada a partir de 100 cenários contendo 4000 trajetórias cada. Para a base de dados de validação, são gerados dados de entrada a partir de 200 trajetórias em cada cenário utilizado na base de dados de treinamento, e a partir de 10 cenários novos contendo 2000 trajetórias cada. Assim como na subseção 4.3.1, os dados de entrada não são normalizados.

4.3 Detalhamento dos algoritmos que compõem o MPNet

Nesta seção, detalha-se a implementação do planejador MPNet. Para tal, são descritos os algoritmos que compõem o referido planejador, dipostos na seção 2.3.

Conforme ilustrado pela Figura 4.1, a primeira etapa a ser realizada é a implementação e o treinamento da ENet, necessário para gerar a base de dados para o treinamento da PNet, que é implementado e treinado em seguida. Com ambas as redes treinadas, implementa-se os algoritmos de planejamento bidirecional, conexão de estados críticos, otimização de trajetórias e

replanejamento. Por fim, tais algoritmos são integrados, formando o planejador de trajetórias MPNet.

4.3.1 Arquitetura da ENet e seu treinamento

Após a geração dos cenários e a criação da base da dados de treinamento da ENet, é necessário efetuar a implementação de sua arquitetura, seguida de seu treinamento. Para tanto, o tipo de AE sugerido originalmente em Qureshi, Miao *et al.* (2019) é o CAE, descrito na subseção 2.3.1. Todas as camadas utilizadas nesta arquitetura, as quais são apresentadas na Tabela 4.1, são camadas totalmente conectadas, também chamadas de camadas densas. Além disso, são utilizadas duas funções de ativação diferentes ao longo da arquitetura. A função de ativação $f(x) = x$, denominada Linear, é utilizada na camada de entrada do codificador, na camada codificadora e na camada de saída do decodificador. Nas demais camadas, a função de ativação utilizada é a *Parametrized Rectified Linear Unit* (PReLU) (HE *et al.*, 2015), definida pela Equação 4.1.

$$f(x_i) = \begin{cases} x_i, & \text{se } x_i > 0 \\ a_i x_i & \text{se } x_i \leq 0 \end{cases} \tag{4.1}$$

A Tabela 4.1 dispõe a configuração de parâmetros da arquitetura implementada e apresenta o número de unidades e a função de ativação de cada camada.

Tabela 4.1 – Configuração de parâmetros utilizada na ENet

Módulo	Camada	Unidades	Função de Ativação
Codificador	Entrada	2800	Linear
	Densa1	512	PReLU
	Densa2	256	PReLU
	Densa3	128	PReLU
	Codificadora	28	Linear
Decodificador	Densa4	128	PReLU
	Densa5	256	PReLU
	Densa6	512	PReLU
	Reconstrutora	2800	Linear

O treinamento desta rede foi efetuado até que fosse alcançada sua convergência. Isso foi feito por meio do acompanhamento da variação da taxa de perda de validação, assumindo-se que a convergência é alcançada quando não tiver havido variação maior do que 1×10^{-3} por 40 épocas na referida taxa de perda. Utilizou-se o valor de 1×10^{-2} para a taxa de aprendizagem e o Adagrad (DUCHI; HAZAN; SINGER, 2011) como algoritmo otimizador.

4.3.2 Arquitetura da PNet e seu treinamento

A implementação e o treinamento da PNet são efetuados a partir da base de dados processada após o treinamento da ENet, conforme descrito pela subseção 4.2.4. A arquitetura utilizada nesta rede corresponde a um PMC, assim como foi disposto na subseção 2.3.2. De modo a aprimorar a capacidade desta arquitetura de generalizar para dados não vistos durante seu treinamento, são utilizadas camadas de *Dropout* (SRIVASTAVA *et al.*, 2014) após algumas de suas camadas totalmente conectadas. Esta abordagem faz com que uma parcela aleatória dos neurônios presentes na camada anterior seja ignorada, resultando na priorização de informações mais importantes durante o treinamento. A parcela de dados ignorada é definida pela taxa de *Dropout*, que varia entre 0 e 1. A Tabela 4.2 descreve a configuração da arquitetura utilizada nesta pesquisa para a PNet. A existência de taxa de *Dropout* em uma camada disposta na referida tabela implica na utilização desta taxa em uma camada de *Dropout* que a sucede.

Tabela 4.2 – Configuração de parâmetros utilizada na PNet

Camada	Unidades	Função de Ativação	Taxa de *Dropout*
Densa1	1280	PReLU	0.5
Densa2	1024	PReLU	0.5
Densa3	896	PReLU	0.5
Densa4	768	PReLU	0.5
Densa5	512	PReLU	0.5
Densa6	384	PReLU	0.5
Densa7	256	PReLU	0.5
Densa8	256	PReLU	0.5
Densa9	128	PReLU	0.5
Densa10	64	PReLU	0.5
Densa11	32	PReLU	—
Densa12	2	Linear	—

No treinamento desta arquitetura, utilizou-se a função de perda EMQ, o algoritmo otimizador Adagrad e uma taxa de aprendizagem de 1×10^{-4}. Assim como no treinamento da ENet, o treinamento desta rede ocorreu até atingir a convergência. Para tal, o limiar da variação mínima da taxa de perda de validação utilizado foi de 1×10^{-3} e o número de épocas em que a variação tem para ultrapassar tal limiar foi de 20 épocas.

4.3.3 Planejamento bidirecional

O planejamento bidirecional, descrito na subseção 2.3.4, utiliza a PNet para gerar alternadamente duas trajetórias, σ_1 e σ_2. Conforme disposto no algoritmo 4.1, a trajetória σ_1 começa seu processamento partindo do estado atual s_1 enquanto a trajetória σ_2 começa partindo do estado destino s_{obj}. Estes também são os estados críticos mais recentes de cada trajetória (respectivamente $s_{\sigma_1}^{atual}$ e $s_{\sigma_2}^{atual}$). Inicialmente, a trajetória corrente σ_t se refere à σ_1. No início de uma

iteração, é gerado um novo estado crítico $s_{\sigma_t}^{atual}$ por meio da PNet, que é incluído na trajetória corrente. Realiza-se uma tentativa de conectar os estados $s_{\sigma_1}^{atual}$ e $s_{\sigma_2}^{atual}$, sendo que: se tal conexão for válida, a ordem da trajetória σ_2 é invertida, de modo que o último estado seja o estado destino, e, então, ambas as trajetórias são concatenadas, resultando na trajetória σ e finalizando este processo; do contrário, alterna-se a trajetória à qual σ_t se refere. Tal procedimento se repete por até N iterações, resultando em falha na geração da trajetória caso este limite seja alcançado.

Algoritmo 4.1: Algoritmo de planejamento bidirecional.

Entrada : A codificação Z_a do cenário
O estado atual s_1
O estado destino s_{obj}
O limite de iterações a serem realizadas

Saída : A trajetória replanejada σ ou um conjunto vazio

Function `PlanejadorBidirecional(`Z_a, s_1, s_{obj}, N`):`
  $\sigma_1 \leftarrow \{s_1\}, \sigma_2 \leftarrow \{s_{obj}\}$, $s_{\sigma_1}^{atual} \leftarrow s_1$, $s_{\sigma_2}^{atual} \leftarrow s_{obj}$
  Inicialmente, a trajetória corrente σ_t se refere a trajetória σ_1
  for $i \leftarrow 0$ N **do**
    $s_{\sigma_t}^{atual} \leftarrow \text{PNET}(Z_a, s_{\sigma_t}^{atual}, s_{\sigma \neq \sigma_t}^{atual})$
    $\sigma_t \leftarrow \sigma_t \cup \sigma_{\sigma_t}^{atual}$
    Efetua-se a tentativa de conexão entre $\sigma_{\sigma_t}^{atual}$ e $\sigma_{\sigma \neq \sigma_t}^{atual}$
    if *conexão resultante é inválida* **then**
      σ_t passa a se referir à outra trajetória
    else
      Inverte-se a ordem dos estados críticos que compõem σ_2
      $\sigma \leftarrow \sigma_1 \cup \sigma_2$
      return σ
    end
  end
  return $\emptyset$

4.3.4 Algoritmos de otimização da trajetória e conexão de estados críticos

A otimização da trajetória é feita através do algoritmo LSC, conforme descrito na subseção 2.3.6. Esta abordagem tem como o objetivo eliminar estados críticos que são desnecessários para a trajetórias, efetuando assim a sua otimização.

A conexão de estados críticos é realizada traçando uma linha reta entre eles e verificando se tal linha colide com algum obstáculo, conforme descrito na subseção 2.3.5. Tal algoritmo é utilizado após a sequência de estados críticos candidata à trajetória ter sido otimizada, de modo que é verificada se a conexão de pares de estados críticos consecutivos desta sequência

corresponde a uma trajetória válida. Além disso, este algoritmo é requisitado durante a execução da estratégia de planejamento bidirecional, da otimização da trajetória e do replanejamento.

4.3.5 Replanejador de trajetórias

O replanejamento de trajetórias é realizado sempre que um par de estados críticos consecutivos na trajetória não seja conectável. Conforme descrito na subseção 2.3.7, inicialmente são removidos os estados críticos inválidos da trajetória. Em seguida, itera-se por ela, adicionando os pares de estados consecutivos que forem conectáveis à trajetória σ_{novo} e processando uma trajetória intermediária σ' que conecte um par de estados consecutivo não conectável. Caso a trajetória σ' gerada seja válida, os estados que a compõem são incluídos na trajetória σ_{novo}, e a iteração continua. Do contrário, o replanejamento é interrompido e resulta em falha. O RH é realizado quando requisitado, sendo que é necessário definir o planejador oráculo a ser utilizado. O algoritmo 4.2 detalha tal procedimento.

Algoritmo 4.2: Algoritmo de replanejamento.

Entrada : A trajetória σ
A codificação Z_a do cenário
O espaço de estados ocupados por obstáculos A_{obs}
Se o replanejamento usará um planejador oráculo
O número de iterações a serem realizadas no planejamento bidirecional para cada par de estados críticos não conectável na trajetória σ

Saída : A trajetória replanejada σ_{novo} ou um conjunto vazio

1 **Function** `Replanejamento(`σ, Z_a, A_{obs}, *planejamento oráculo,* N`):`
2 $\sigma_{novo} \leftarrow \emptyset$
3 Os estados críticos inválidos são removidos, resultando em um conjunto de n estados críticos, denominado $\hat{\sigma}$
4 **for** $i \leftarrow 0$ **to** $n-1$ **do**
5 Efetua-se a tentativa de conexão entre os estados $\hat{\sigma}_i$ e $\hat{\sigma}_{i+1}$
6 **if** *conexão resultante é válida* **then**
7 $\sigma_{novo} \leftarrow \sigma_{novo} \cup \{\hat{\sigma}_i, \hat{\sigma}_{i+1}\}$
8 **else**
9 **if** *planejamento oráculo* **then**
10 $\sigma' \leftarrow$ PlanejadorOraculo$(\hat{\sigma}_i, \hat{\sigma}_{i+1}, A_{obs})$
11 **else**
12 $\sigma' \leftarrow$ PlanejadorBidirecional$(Z_a, \hat{\sigma}_i, \hat{\sigma}_{i+1}, N)$
13 **end**
14 **if** σ' **then**
15 $\sigma_{novo} \leftarrow \sigma_{novo} \cup \sigma'$
16 **else**
17 **return** $\emptyset$
18 **end**
19 **end**
20 **end**
21 **return** σ_{novo}

4.3.6 Integração de todos os elementos do MPNet

A integração de todos os elementos do MPNet, descrita na subseção 2.3.8, é realizada conforme disposto no algoritmo 4.3, aproveitando os algoritmos descritos anteriormente. A linha 20 do referido algoritmo representa a possibilidade de efetuar o planejamento com RH. Caso tal método de replanejamento fosse utilizado, os parâmetros avaliativos de desempenho e de tempo de execução do MPNet estariam sujeitos ao planejador oráculo escolhido. Salienta-se que o presente trabalho não faz uso desta solução, efetuando a avaliação do MPNet sem a influência de um planejador oráculo (um planejador diferente do MPNet utilizado para corrigir trechos que

o RN não tenha sido capaz de corrigir).

Particularmente no presente trabalho, são realizadas no máximo 12 tentativas de replanejamento ($N_{rp} = 12$), os limites no número N de iterações para o planejamento bidirecional executado antes da otimização de trajetória e para o replanejamento são iguais a 100 e 50, respectivamente.

Algoritmo 4.3: Algoritmo integrado da MPNet.

Entrada : O estado atual s_1
O estado destino s_{obj}
O espaço de estados ocupados por obstáculos A_{obs}
Se será utilizado o RH

Saída : A trajetória final σ ou um conjunto vazio

Function `MPNet(`s_1, s_{obj}, A_{obs}, RH`)`**:**
  // Constantes utilizadas no presente trabalho.
  $N_{rp} \leftarrow 12$, $N \leftarrow 100$
  $Z_a \leftarrow \text{ENet}(A_{obs})$
  $\sigma \leftarrow \text{PlanejadorBidirecional}(Z_a, s_1, s_{obj}, N)$
  $\sigma \leftarrow \text{LSC}(\sigma)$
  Efetua-se a validação da trajetória
  if *trajetória válida* **then**
    return σ
  else
    // Limite de iterações do planejador bidirecional no replanejamento
    for $i \leftarrow 0$ **to** N_{rp} **do**
      $\sigma \leftarrow \text{Replanejamento}(\sigma, Z_a, A_{obs}, \text{false}, 50)$
      $\sigma \leftarrow \text{LSC}(\sigma)$
      Efetua-se a validação da trajetória
      if *trajetória válida* **then**
        return σ
      end
    end
    if *RH* **then**
      // Usando o replanejamento híbrido
      $\sigma \leftarrow \text{Replanejamento}(\sigma, Z_a, A_{obs}, \text{true}, 0)$
      $\sigma \leftarrow \text{LSC}(\sigma)$
      Efetua-se a validação da trajetória
      if *trajetória válida* **then**
        return σ
      end
    end
  end
  return $\emptyset$

5 Análise dos Resultados

Neste capítulo são dispostos os resultados obtidos nos experimentos com o planejador MPNet com RN em ambientes Simples 2D, o qual teve sua implementação detalhada na seção 4.3. São analisados os resultados obtidos no treinamento da ENet e da PNet, bem como a taxa de sucesso do MPNet com RN em gerar trajetórias válidas e seus respectivos tempos de execução. Deste modo, este capítulo é dividido em duas seções: apresentação dos resultados obtidos no treinamento da ENet e da PNet; e análise dos resultados obtidos referentes ao desempenho do MPNet.

Na seção referente aos resultados obtidos no treinamento da ENet e da PNet, são apresentados os valores da taxa de perda de validação e do número de épocas atingidas no momento de convergência para cada rede neural treinada, bem como se analisa a representação reconstruída pela ENet. A seção dos resultados relacionados ao desempenho do MPNet é composta pela análise dos seguintes parâmetros avaliativos: taxa de sucesso na geração de trajetórias válidas e tempo de execução do algoritmo. Além disso, investiga-se o funcionamento de tal planejador, por meio de uma análise detalhada dos resultados associados ao planejamento de alguns cenários selecionados.

5.1 Treinamento da ENet e da PNet

Esta seção apresenta os resultados referentes ao treinamento da ENet e da PNet, sendo avaliados a taxa de perda de validação e o número de épocas ocorridas no momento da convergência. Além disso, é disposta a representação reconstruída gerada pela ENet (que consiste em um CAE) após seu treinamento, a fim de realizar uma análise visual de seu desempenho e propor possíveis aprimoramentos.

Os modelos foram implementados e treinados através das bibliotecas *PyTorch* 1.8 (PASZKE *et al.*, 2019) e *PyTorch Lightning* (FALCON *et al.*, 2019). A linguagem de programação escolhida para seu desenvolvimento foi o *Python* 3.8. O sistema utilizado para treinamento e teste foi uma instância de máquina virtual disponibilizada pela *Google Cloud Compute Engine* (GOOGLE, 2021), baseada na máquina predefinida *e2-highcpu-32*, que possui as seguintes configurações: 32 vCPUs[1], 32 GB de memória RAM, 40 GB de armazenamento.

O treinamento do CAE foi realizado até que ocorresse a convergência do modelo, conforme descrito na subseção 4.3.1. Tal treinamento foi repetido 40 vezes buscando coletar dados sobre a taxa de perda de validação obtida, assim como a quantidade de épocas atingidas. Cada

[1] Uma vCPU consiste em uma *Hyper-thread* em um dos processadores disponíveis para a máquina, que no caso são: Intel Xeon Scalable Processor (Skylake) ou Intel Xeon E5 V4 (Broadwell E5).

treinamento da ENet demorou aproximadamente 5 horas e 30 minutos. A Tabela 5.1 lista a média e desvio padrão dos valores obtidos.

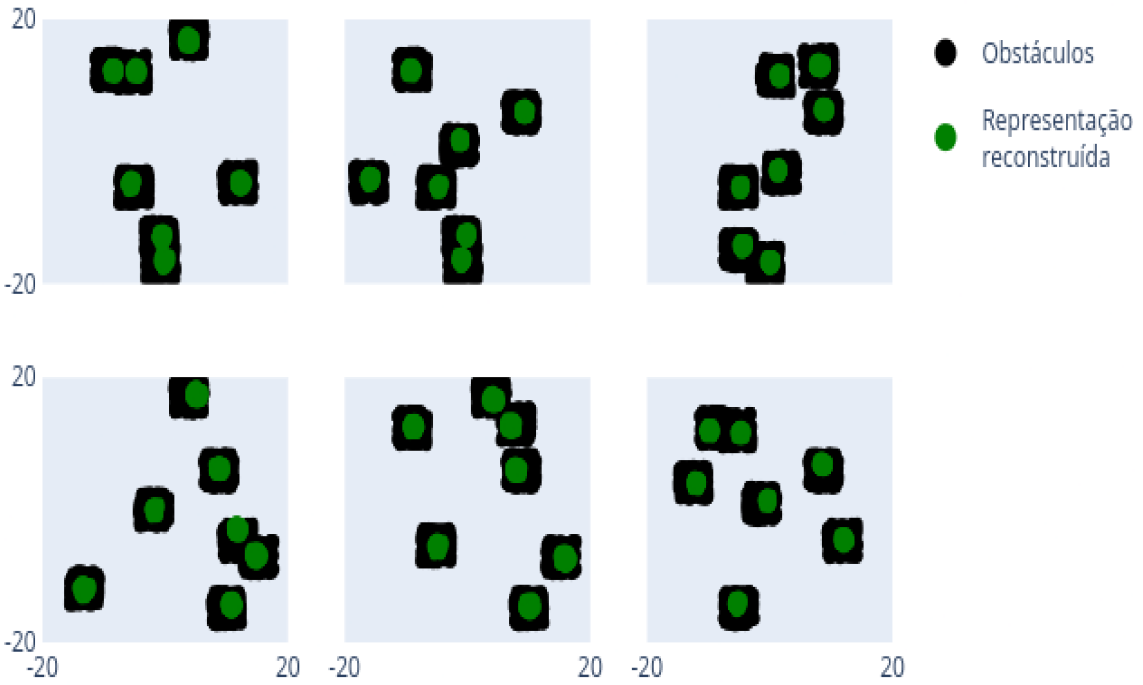
Tabela 5.1 – Resultados obtidos no treinamento da ENet

Parâmetro	Média (Desvio padrão)
Taxa de perda de validação	4.14 (1.08)
Número de épocas	67.50 (11.62)

O valor obtido do desvio padrão se deve à inicialização aleatória do conjunto de pesos sinápticos do modelo, o que pode fazer com que o modelo fique preso a um mínimo local em algumas iterações, influenciando tanto na taxa de perda obtida quanto no número de épocas.

A fim de investigar a qualidade do modelo após o treinamento, utiliza-se a representação reconstruída de cenários não presentes na base de dados de treinamento. Tal representação permite que sejam identificadas possíveis falhas no modelo, bem como avaliar a sua capacidade de generalizar eficientemente mesmo quando são introduzidos cenários que não estavam presentes na base de dados de treinamento. A Figura 5.1 ilustra as representações reconstruídas obtidas para seis destes cenários.

Figura 5.1 – Representação reconstruída gerada pelo modelo do CAE treinado



Fonte: Autor

Observa-se que os pontos gerados na representação reconstruída tendem a se concentrar

em estados próximos ao centro dos obstáculos. Uma possível causa desse efeito é a dimensão da representação codificada, que pode estar sendo insuficiente para que o CAE consiga gerar uma reconstrução que defina totalmente o cenário. Isso pode comprometer um pouco o processo de geração de estados críticos e de trajetórias, pelo seguinte motivo: a PNet, tendo sido treinada a partir de vários cenários em que ele não percebia, com exatidão, os limites dos espaços ocupados pelos obstáculos, pode, após treinada, gerar com mais frequência estados críticos pertencentes a um desses espaços, o que vai comprometer, também, o processo de avaliação de estados válidos durante o planejamento bidirecional. Uma das consequências deste efeito é a demanda mais frequente de um acionamento do algoritmo de replanejamento.

Diferentemente do treinamento do CAE, o treinamento do PMC (ou PNet) foi executado somente uma vez. Isto ocorreu devido à quantidade de parâmetros a serem treinados no modelo apresentado na subseção 4.3.2, bem como ao tamanho da base de dados de treinamento, o que torna o processo lento e inviabiliza a coleta de mais iterações. O treinamento da PNet demorou aproximadamente 13 horas para ser efetuado. A Tabela 5.2 descreve os valores da taxa de perda de validação e do número de épocas obtidos no treinamento do PMC.

Tabela 5.2 – Resultados obtidos no treinamento da PNet

Parâmetro	Valor obtido
Taxa de perda de validação	6.52
Número de épocas	88

Nota-se que a taxa de perda de validação obtida no treinamento do PMC tem um valor alto considerando as dimensões do cenário ($1600\,\mathrm{m}^2$), o que significa que os estados críticos obtidos pelo modelo se distanciaram dos estados presentes nos dados rotulados. Tal resultado pode ser justificado pela alta taxa de *Dropout* (0.5) utilizada em conjunto com o baixo número de unidades advindas da representação codificada no dado de entrada (28 unidades) do PMC, por meio da falha da PNet em identificar propriamente as extremidades de cada obstáculo, o que pode ser intensificado pela atuação das camadas de *Dropout*. As consequências disto podem ser positivas ou negativas. Trajetórias podem se tornar inválidas ou distantes da trajetória ótima, assim como é possível que ocorra o aumento da taxa de sucesso devido à distância da trajetória de referência, que pode incluir estados muito próximos aos obstáculos, permitindo que o modelo aprenda a se distanciar mais dos mesmos. A seção 5.2 detalha os resultados obtidos tal arquitetura, exemplificando casos que podem ter sido influenciados por este efeito.

Como o PMC retorna somente o próximo ponto pertencente a uma trajetória, é mais conveniente abordar os seus resultados na subseção 5.2.3, que apresenta e avalia o funcionamento do planejador bidirecional, etapa em que esta rede é utilizada.

5.2 Desempenho do planejador MPNet com RN

Esta seção analisa o desempenho do planejador MPNet com RN. Para tal, foram coletados resultados de 10000 trajetórias processadas pelo MPNet e pelo planejador RRT*. Tais trajetórias originam-se de 100 cenários usados no treinamento da PNet (cenários conhecidos). A partir de cada um deles, foram selecionados 50 pares de estados atual e destino provenientes de trajetórias que não fizeram parte da base de dados de treinamento da PNet. Além disso, também foram selecionados 500 outros pares provenientes de cada cenário de um conjunto de 10 que não foram usados no mesmo treinamento (cenários inéditos).

5.2.1 Taxa de sucesso na geração de trajetórias válidas

Com o objetivo de avaliar a taxa de sucesso do MPNet na geração de trajetórias válidas, foram coletados dados referentes ao resultado do processamento realizado por ele, sendo que os resultados possíveis são: sucesso, replanejamento bem sucedido, falha e replanejamento mal sucedido.

Tabela 5.3 – Taxa de sucesso na geração de trajetórias válidas do planejador MPNet

Resultado	Cenários conhecidos (%)	Cenários inéditos (%)	Total (%)
Sucesso	3403 (68.06)	3380 (67.60)	6783 (67.83)
Falha	9 (0.18)	15 (0.30)	24 (0.24)
Replanejamento bem suc.	1490 (29.80)	1496 (29.92)	2986 (29.86)
Replanejamento mal suc.	98 (1.96)	109 (2.18)	207 (2.07)

Tabela 5.4 – Taxa de sucesso e falha geral para cenários conhecidos e inéditos do planejador MPNet

Resultado final	Cenários conhecidos (%)	Cenários inéditos (%)	Total (%)
Sucesso geral	4893 (97.86)	4876 (97.52)	9769 (97.69)
Falha geral	107 (2.14)	124 (2.48)	231 (2.31)

De acordo com a Tabela 5.4, foi obtida uma taxa de sucesso geral de 97.69% utilizando o planejador MPNet, que é próxima da taxa de sucesso de obtida em Qureshi, Miao *et al.* (2019) (que foi de 99.30% em cenários vistos e de 98.30% em cenários novos).

Observando a Tabela 5.3, é possível notar que o planejador teve um número maior de falhas quando deparado com cenários inéditos. Além disso, foi necessário realizar o replanejamento em um número maior de trajetórias para tais cenários. Entretanto, a diferença na taxa de sucesso entre os cenários conhecidos e os inéditos não foi significativa (aproximadamente 0.35%), o que permite a conclusão de que o planejador generaliza eficientemente para cenários e trajetórias que não estavam presentes na base de dados de treinamento.

De modo a investigar o comportamento do MPNet com RN em relação ao comportamento do RRT*, selecionaram-se alguns exemplos para cada tipo de resultado encontrado.

5.2.2 Análise de amostras de trajetórias geradas pelo MPNet que não resultaram em replanejamento

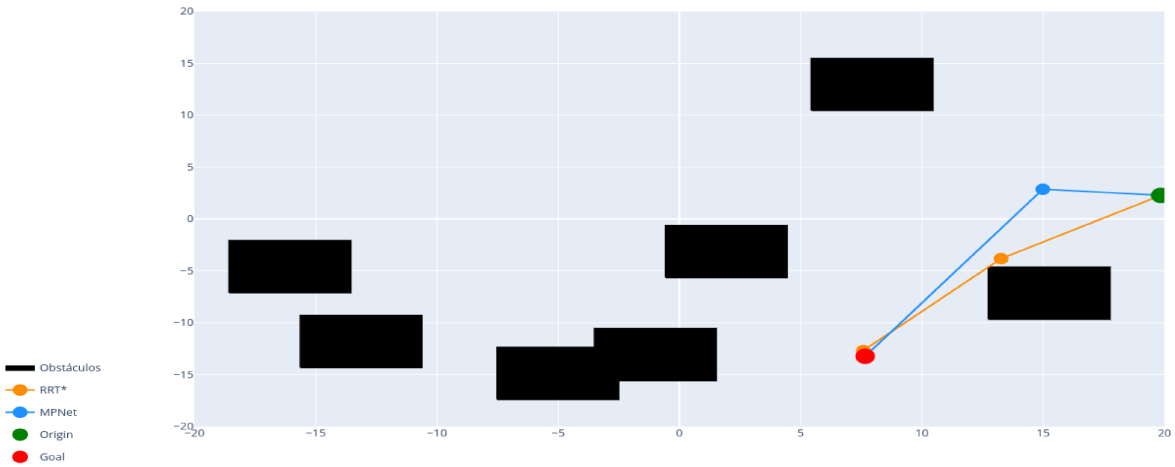
Nesta subseção são analisadas amostras de trajetórias geradas pelo MPNet que resultaram em sucesso ou falha, sem que fossem submetidas à etapa de replanejamento. Deste modo, podemos investigar as razões para uma trajetória não precisar do replanejamento e para uma trajetória falhar sem possibilidade de replanejamento.

Análise de trajetórias que resultaram em sucesso

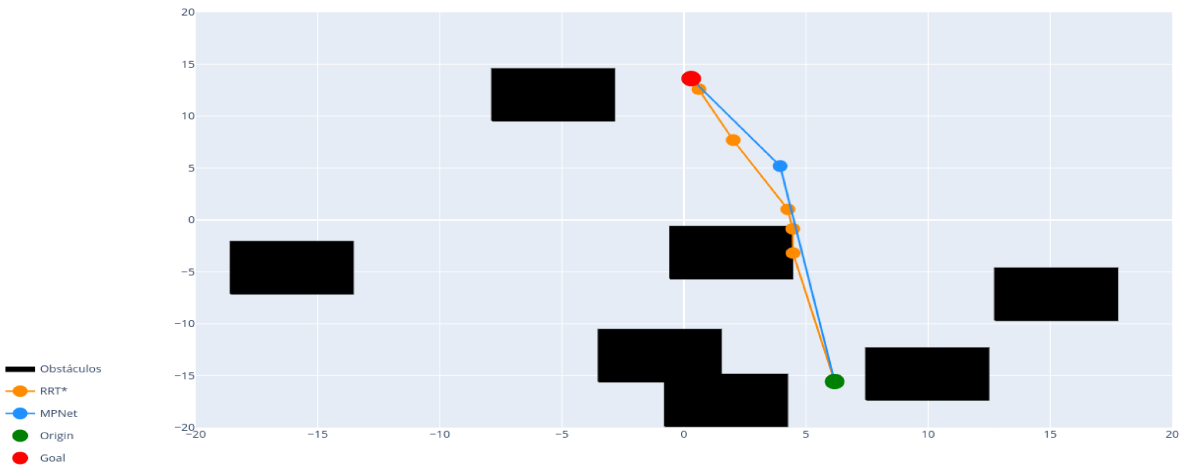
Considera-se que uma trajetória resultou em sucesso, sem necessidade de replanejamento, nas situações em que foi possível conectar os estados críticos mais recentes gerados pelo planejador bidirecional e, além disso, a trajetória otimizada produzida a partir deles é válida.

A Figura 5.2 apresenta exemplos de planejamentos realizados pelo MPNet que resultaram em sucesso sem necessidade de replanejamento, bem como a trajetória encontrada pelo RRT*.

Figura 5.2 – Exemplos de planejamentos que resultaram em sucesso sem necessidade de replanejamento



(a) Cenário conhecido



(b) Cenário inédito

Fonte: Autor

Observando as trajetórias geradas pelo RRT* na Figura 5.2, percebe-se que se aproximam consideravelmente dos obstáculos, enquanto que as trajetórias geradas pelo MPNet se distanciam deles. Esse comportamento do RRT* pode ter influenciado o treinamento do PMC (visto que os dados de treinamento utilizados se originam de trajetórias geradas pelo RRT*), dificultando sua convergência para os dados rotulados próximos aos obstáculos, considerando que as codificações dos cenários (também utilizadas no treinamento da PNet) resultaram em reconstruções imperfeitas, e assim causando divergências entre as trajetórias geradas pelo RRT* e pelo MPNet.

Por fim, analisando visualmente a trajetória gerada pelo MPNet descrita na Figura 5.2a, é possível perceber que ela é menos ótima do que a trajetória gerada pelo RRT*, que se aproxima de uma linha reta entre o ponto de origem e o ponto de destino. Este comportamento pode ser um sintoma da alta taxa de perda de validação obtida no treinamento do PMC, descrita pela

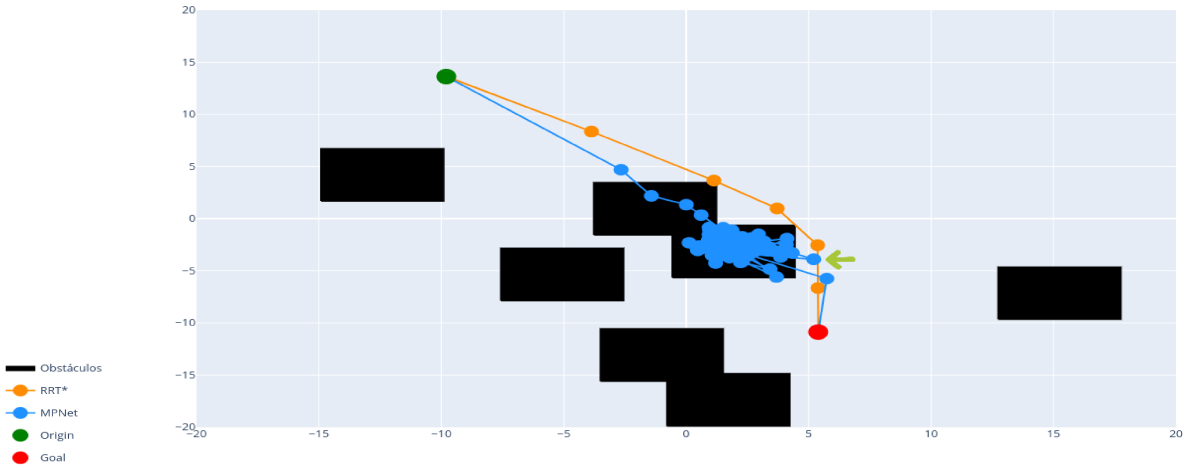
Tabela 5.2, visto que pode ser observado um distanciamento entre os estados gerados pela PNet e os obstáculos.

Análise de trajetórias que resultaram em falha

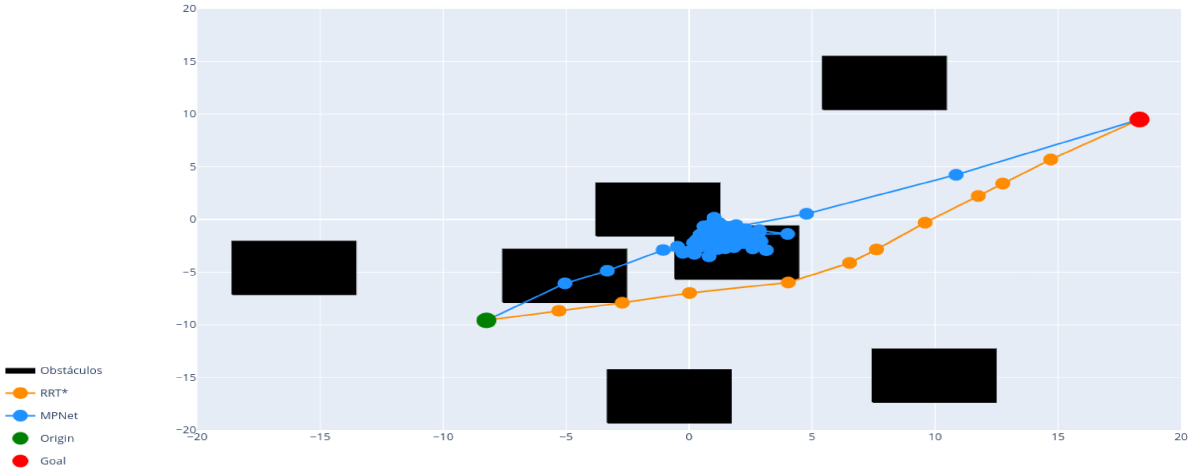
Considera-se que uma trajetória resultou em falha quando não foi possível conectar os estados críticos mais recentes gerados no planejamento bidirecional, tendo sido atingido o número limite de $N = 100$ iterações estabelecido.

A Figura 5.3 apresenta exemplos de planejamentos realizados pelo planejador MPNet que resultaram em falha sem possibilidade de replanejamento, bem como a trajetória gerada pelo planejador RRT*.

Figura 5.3 – Exemplos de planejamento que resultaram em falha sem possibilidade de replanejamento



(a) Cenário conhecido



(b) Cenário inédito

Fonte: Autor

Observando a Figura 5.3, nota-se um padrão em ambas as imagens, o planejador bidirecional não foi capaz de desviar dos obstáculos. Na Figura 5.3a, nota-se que o estado crítico indicado pela seta verde é válido, mesmo tendo sido gerado depois de estados críticos inválidos. Apesar disso, seu par era inválido e a PNet foi incapaz de gerar outros estados válidos a partir deles. Este erro pode ser corrigido efetuando alterações nos modelos da ENet ou da PNet buscando melhorar a capacidade de evitar obstáculos do MPNet, como a utilização de um AE que codifique e reconstrua o cenário com maior eficiência.

5.2.3 Análise de amostras de trajetórias geradas pelo MPNet em que o replanejamento foi realizado

Considera-se que uma trajetória precisa ser replanejada sempre que tiver sido possível conectar os estados críticos mais recentes gerados pelo planejador bidirecional, contudo a trajetória resultante do LSC foi inválida. Nesta subseção, são analisadas as amostras de trajetórias geradas pelo MPNet em que ocorreu a tentativa de replanejamento. Para tal, foram selecionados dois exemplos em que o replanejamento resultou em uma trajetória válida, bem como dois exemplos em que o replanejamento resultou em uma trajetória inválida. Em ambos os casos um dos exemplos é de um cenário conhecido e, o outro, de um cenário inédito.

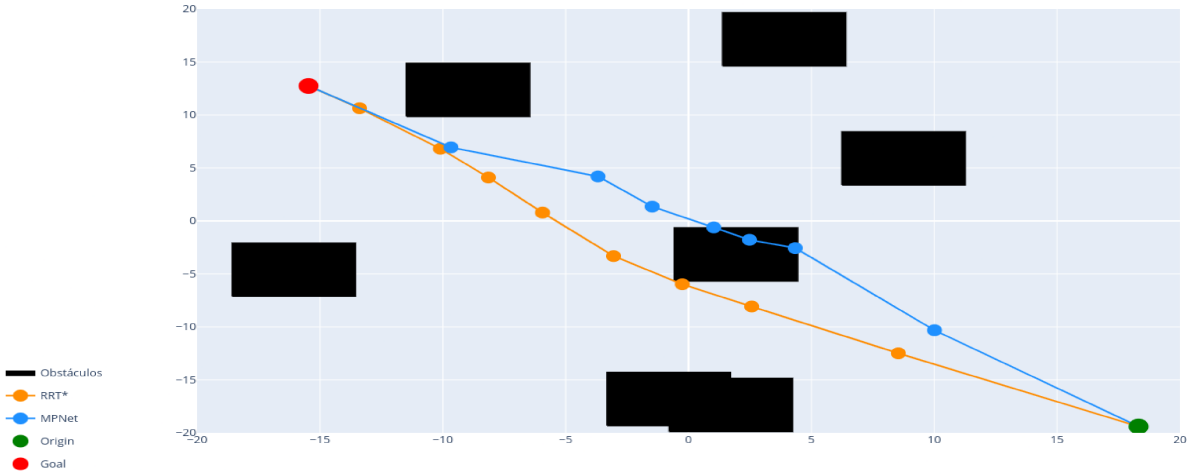
Para cada exemplo, são ilustradas as trajetórias resultantes do planejamento bidirecional, otimização da trajetória, o replanejamento e por fim a trajetória resultante. Assim, é possível analisar visualmente o resultado obtido em cada etapa, permitindo um maior aprofundamento sobre o comportamento do algoritmo.

Análise dos resultados do planejamento bidirecional

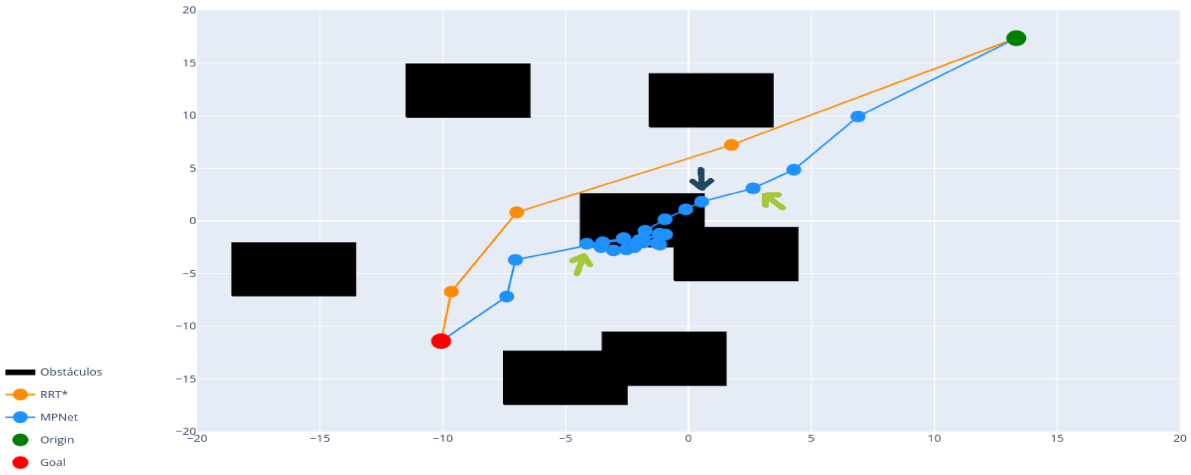
O planejamento bidirecional é responsável pela geração do conjunto de estados críticos candidatos à trajetória, conforme explicado na seção 2.3. Para fins ilustrativos, os pares de estados críticos consecutivos que pertencem ao conjunto gerado no planejamento bidirecional foram conectados, formando uma trajetória.

As Figuras 5.4 e 5.5 exemplificam os resultados obtidos no planejamento bidirecional de amostras que resultaram em replanejamentos bem e mal sucedidos. A trajetória resultante do planejamento realizado pelo RRT* é incluída para fins de comparação.

Figura 5.4 – Etapa do planejamento bidirecional de amostras que resultaram em planejamento bem sucedido



(a) Cenário conhecido

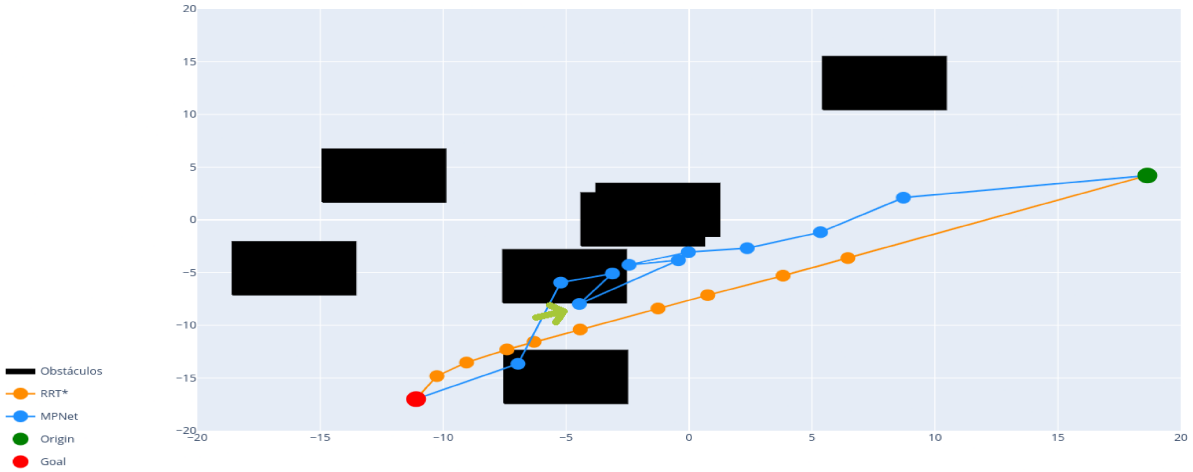


(b) Cenário inédito

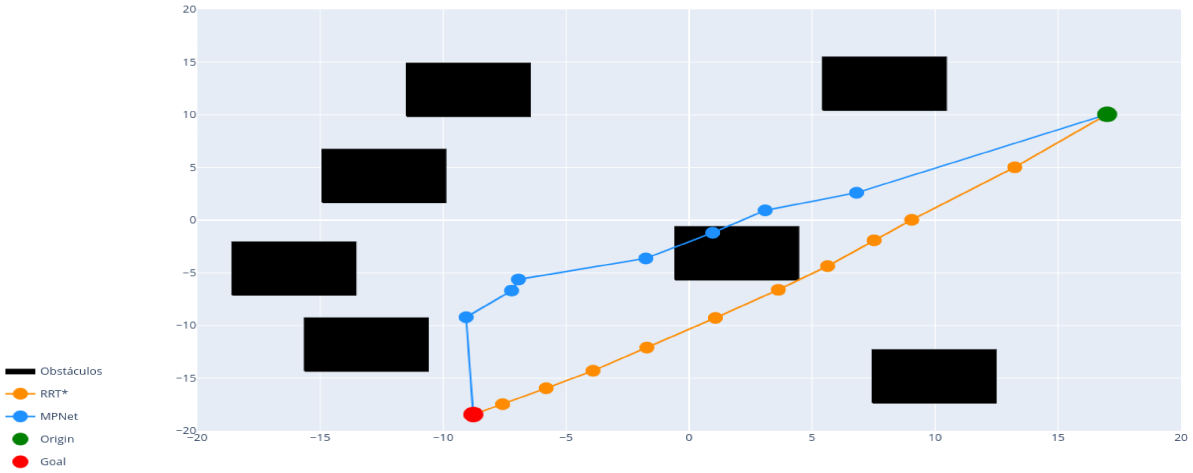
Fonte: Autor

Observando a Figura 5.4, pode-se observar uma diferença na quantidade de estados críticos gerados pelo planejamento bidirecional. Tal quantidade é menor na Figura 5.4a devido ao fato de que os estados gerados após o primeiro estado inválido tinham como destino um estado distante que era válido. Na Figura 5.4b, a iteração que tinha o primeiro estado inválido como destino (descrita pelos estados indicados pelas setas verdes) gerou outro estado inválido (que corresponde ao estado indicado pela seta azulada), o que fez com que os novos estados críticos gerados pela PNet também fossem inválidos. Após isso, lentamente foram gerados estados inválidos mais próximos de sair do obstáculo, até que fosse encontrado um par de estados válidos conectáveis e, portanto, finalizou-se a etapa de planejamento bidirecional.

Figura 5.5 – Etapa do planejamento bidirecional de amostras que resultaram em planejamento mal sucedido



(a) Cenário conhecido



(b) Cenário inédito

Fonte: Autor

Assim como na Figura 5.4a, é possível observar na Figura 5.5 que os estados inválidos tinham como destino um estado válido distante, fazendo com que a PNet tivesse maior facilidade em gerar estados que não colidissem com o obstáculo. Na Figura 5.5a, nota-se que ocorreu o estado indicado pela seta verde representa um recuo na trajetória. Tais recuos podem ser problemáticos na etapa de otimização da trajetória, arriscando reduzir a eficiência do algoritmo, visto que este processo é realizado de acordo com a ordem dos estados críticos na trajetória.

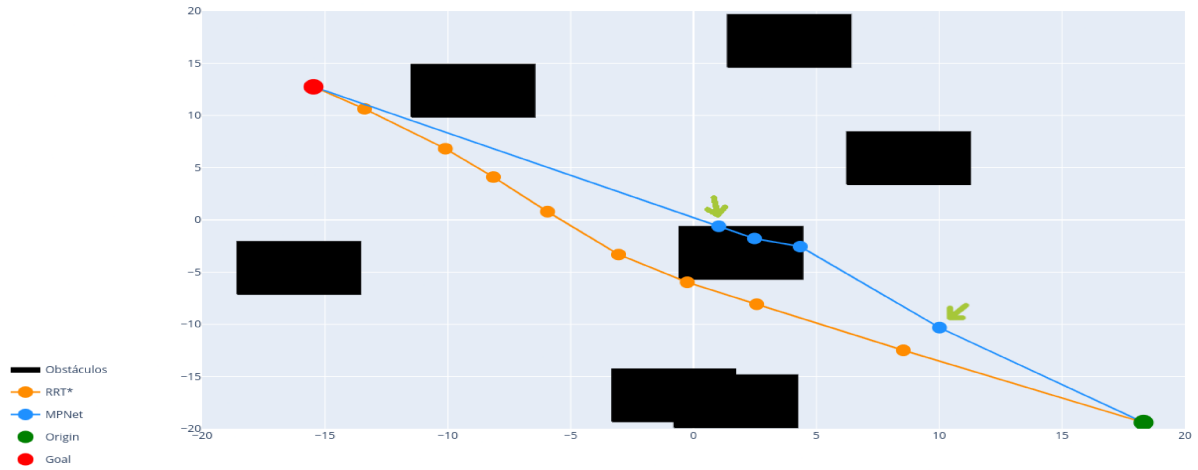
Análise dos resultados da otimização de trajetória

Nesta etapa são otimizadas as trajetórias obtidas na etapa anterior. O algoritmo LSC busca atalhos na trajetória através da conexão entre os seus estados iniciais com os seus estados

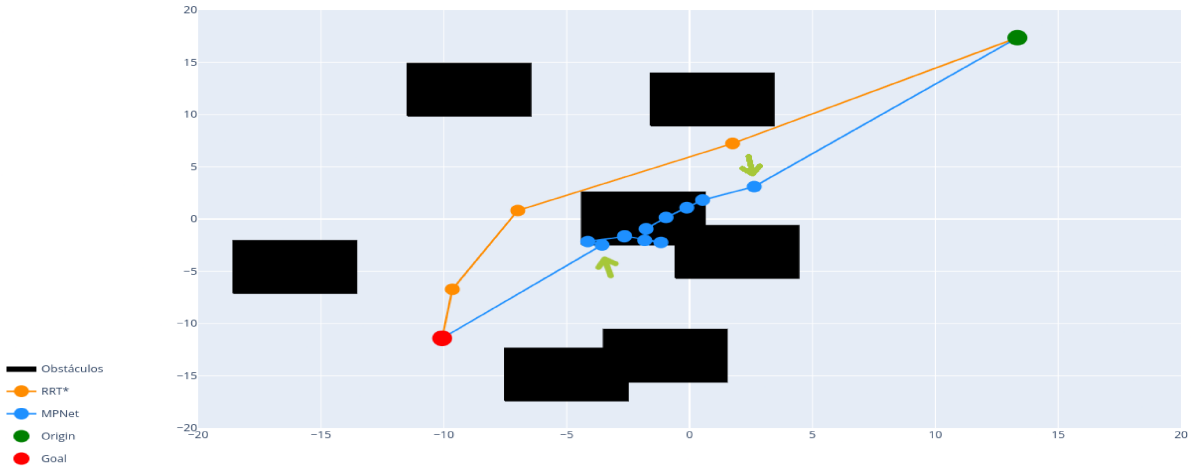
finais, de modo que se prioriza a conexão dos estados de início da trajetória com os estados do final da mesma, conforme descrito na seção 2.3.

As Figuras 5.6 e 5.7 demonstram a otimização das trajetórias analisadas na etapa anterior. A trajetória resultante do planejamento realizado pelo RRT* é incluída para fins de comparação.

Figura 5.6 – Etapa da otimização de amostras que resultaram em planejamento bem sucedido



(a) Cenário conhecido

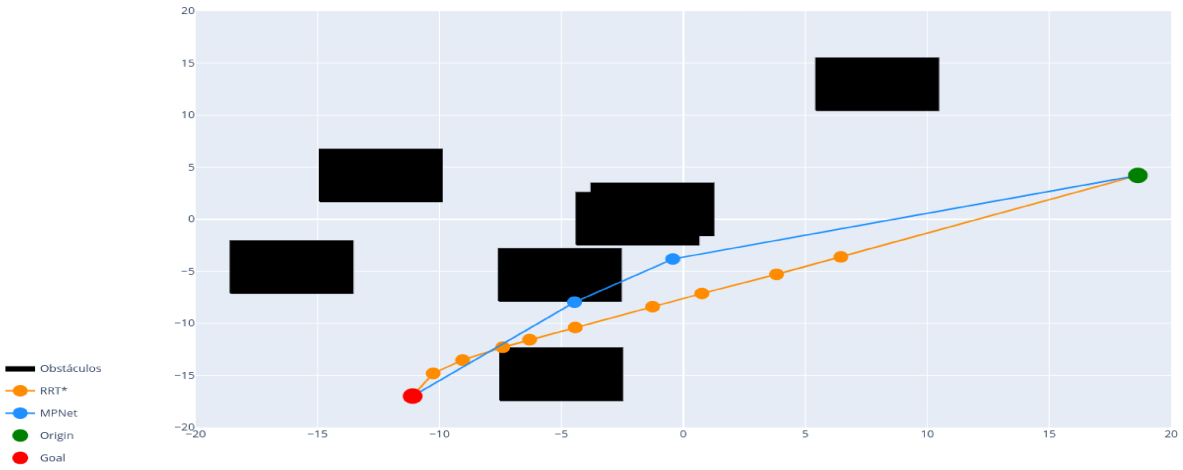


(b) Cenário inédito

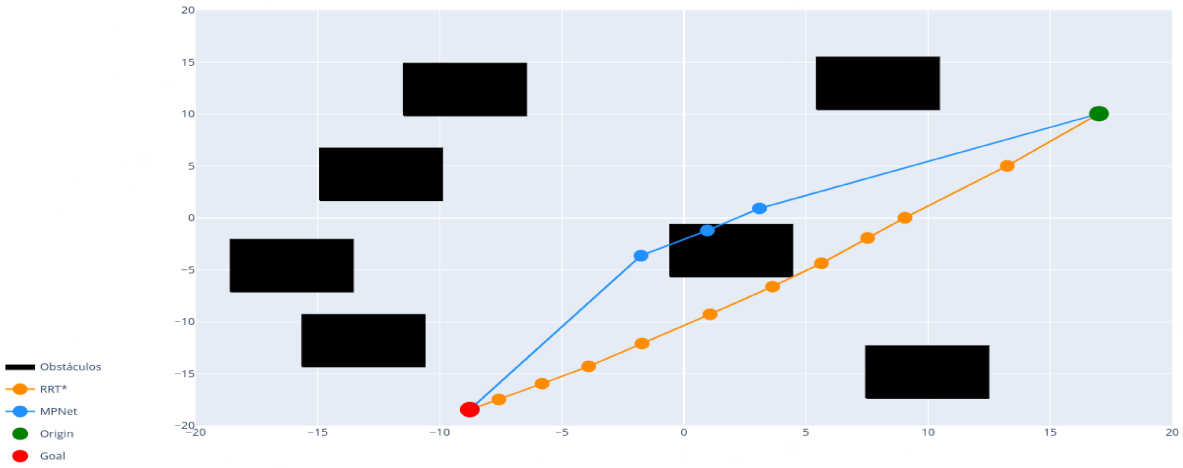
Fonte: Autor

É possível notar que, tanto na Figura 5.6a quanto na Figura 5.6b, os estados críticos que estavam os apontados pelas setas verdes não são otimizados, mesmo sendo inválidos. Isso ocorre devido à necessidade de que os estados sejam conectáveis para que seja possível efetuar uma atividade de otimização a partir deles. Como não foi possível conectar os estados pertencentes à trajetória, os estados inválidos foram mantidos.

Figura 5.7 – Etapa da otimização de amostras que resultaram em planejamento mal sucedido



(a) Cenário conhecido



(b) Cenário inédito

Fonte: Autor

Observando a Figura 5.7, nota-se uma falha na tentativa de desviar do obstáculo, de modo que a colisão ocorre com uma pequena extremidade dele. A seguir será analisada a razão destes exemplos resultarem em falha no replanejamento, apesar de visualmente parecer que não haveria dificuldades em corrigi-los.

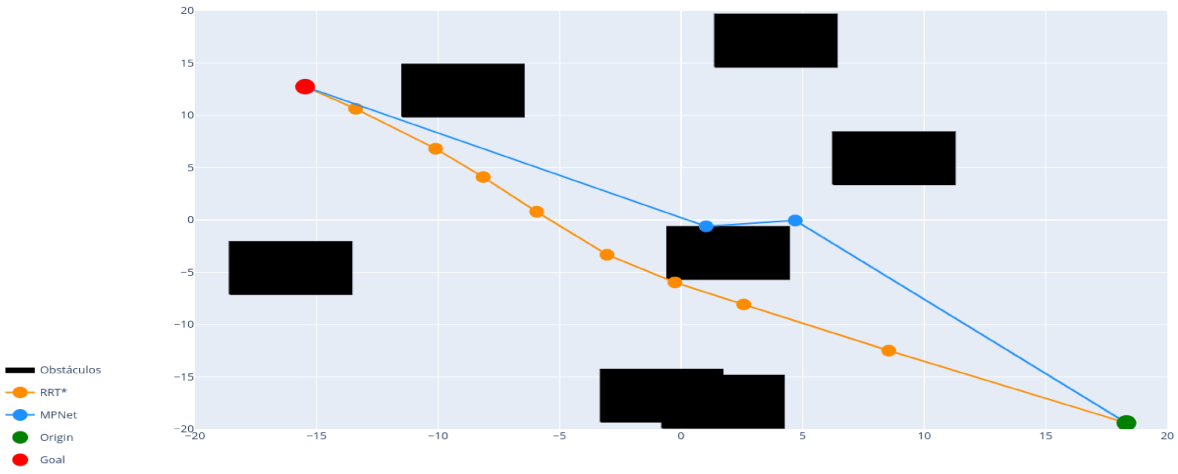
Análise dos resultados do replanejamento

Conforme descrito na subseção 2.3.7, o replanejamento inicia com a remoção de todos os estados críticos inválidos. Em seguida, o planejamento bidirecional é executado para cada par de estados críticos consecutivos que não sejam conectáveis. Os novos estados críticos encontrados são unidos aos estados críticos válidos anteriores, propondo-se, então, uma nova conexão envolvendo o novo conjunto de estados críticos produzidos. Caso a trajetória resultante desta conexão seja válida, o replanejamento resulta em um sucesso. Do contrário, tenta-se realizar o replanejamento

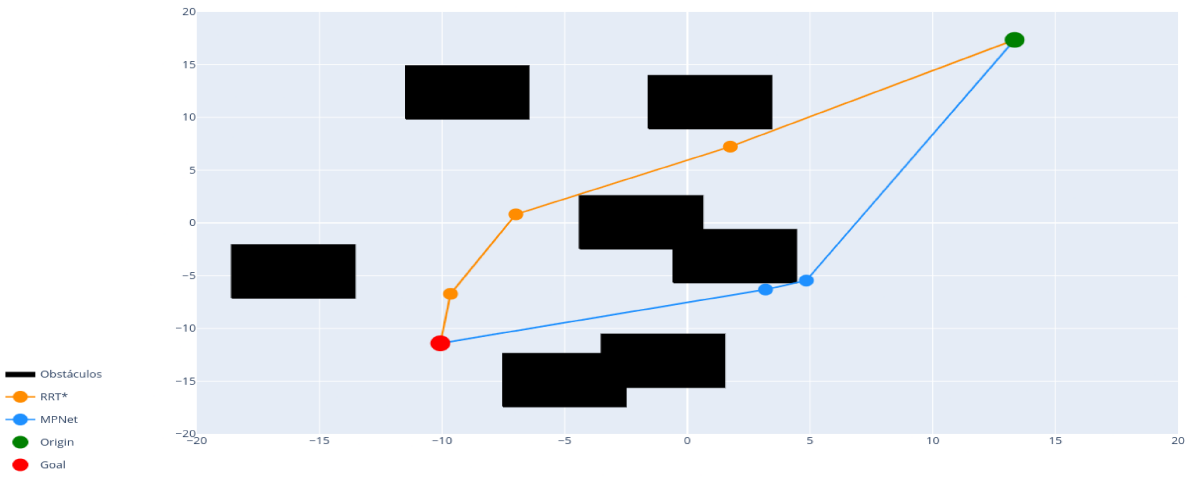
novamente com a nova trajetória. Neste trabalho, esse processo se repete por até 12 vezes. Se não foi obtido um sucesso no replanejamento ao fim da 12^{a} iteração, o replanejamento resulta em falha.

As Figuras 5.8 e 5.9 ilustram o resultado do replanejamento dos exemplos analisados anteriormente. A trajetória resultante do planejamento realizado pelo RRT* é incluída como referência.

Figura 5.8 – Etapa do replanejamento de amostras que resultaram em planejamento bem sucedido



(a) Cenário conhecido



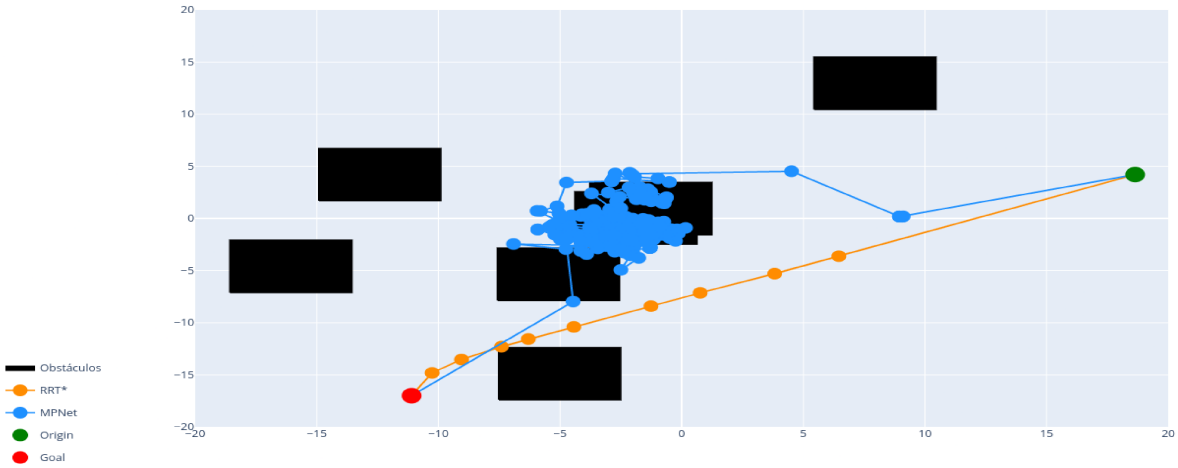
(b) Cenário inédito

Fonte: Autor

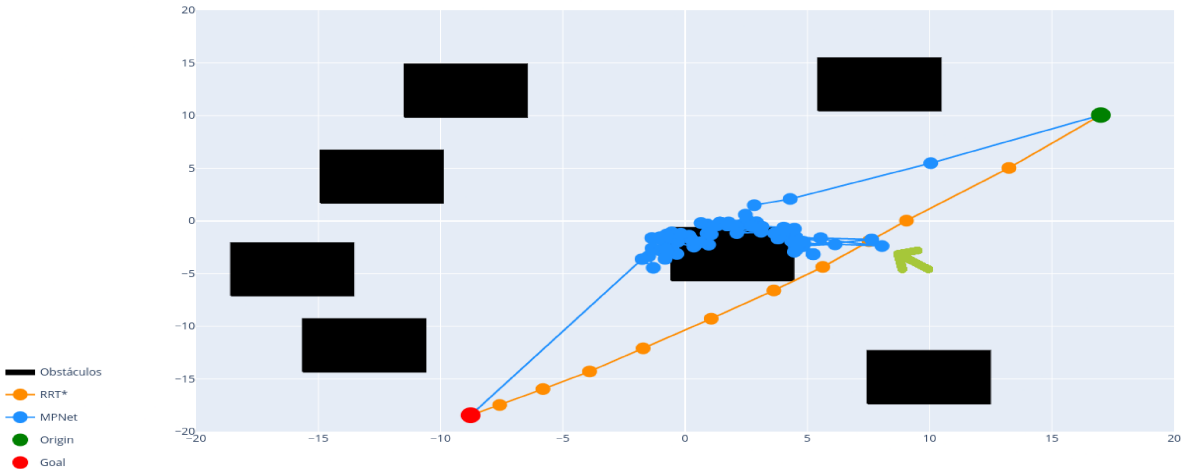
O replanejamento visto na Figura 5.8a ocorreu como o esperado, deslocando um dos ponto para cima, de modo que a trajetória desviasse corretamente do obstáculo. Apesar da trajetória gerada ser válida, um de seus estados críticos se encontra muito próximo de um obstáculo, demonstrando semelhanças com o comportamento presente nos dados rotulados utilizados no treinamento da PNet.

Observando a Figura 5.6b, nota-se que o replanejamento resultou em uma trajetória que atravessa um caminho oposto à trajetória ilustrada na Figura 5.4b e ao caminho utilizado pelo RRT*. Obtém-se tal resultado devido à natureza iterativa do planejador bidirecional, que, no caso, encontrou uma solução que conecta os estados inválidos passando por coordenadas abaixo dos obstáculos, ao invés de acima deles. Esta diferença pode ter relação com a alta taxa de perda de validação descrita pela Tabela 5.2, que evidencia um distanciamento em relação aos dados rotulados provindos do RRT*.

Figura 5.9 – Etapa do replanejamento de amostras que resultaram em planejamento mal sucedido



(a) Cenário conhecido



(b) Cenário inédito

Fonte: Autor

Na Figura 5.9, vê-se que o planejador não foi capaz de encontrar uma solução para a trajetória anterior. Observando a Figura 5.9a, em contraste com a Figura 5.7a, nota-se que o planejador tendeu a buscar soluções utilizando estados acima dos obstáculos, invés de buscar no espaço de estados abaixo deles, como seria intuitivo visualmente. Um fator que dificultou essa

busca foi a proximidade do estado mais próximo ao destino em relação a um obstáculo, o que aumenta a chance da PNet de gerar novos estados críticos que colidam com ele.

Uma característica a ser observada nas Figuras 5.9a e 5.9b é a existência estados da trajetória que, visualmente, pareciam ser opções com maior potencial de gerar uma trajetória válida. Na Figura 5.9b, dois estados gerados no replanejamento (indicados pela seta verde) se aproximam da trajetória gerada pelo RRT*, no entanto o MPNet não identifica o potencial destes estados. Este comportamento é causado pelo algoritmo otimizador de trajetórias. Por priorizar conexões com estados no final da trajetória (é importante lembrar que os estados pertencentes à trajetória são conectados pela ordem em que foram gerados, independentemente do custo ou da validade da trajetória), estes estados potenciais foram ignorados pelo otimizador, o que resultou em um replanejamento mal sucedido. Uma possível solução para esta falha é a utilização de heurística na otimização da trajetória, que resultaria em uma trajetória ótima para o conjunto de estados críticos gerados no planejamento bidirecional. No entanto, tal solução pode resultar no aumento do tempo de processamento.

5.2.4 Análise de tempo de execução

Nesta subseção, são comparados os dados referentes ao tempo de execução do planejador MPNet com RN e do planejador RRT*, brevemente descrito na subseção 4.2.3, para as trajetórias selecionadas. Os dados de tempo de execução obtidos pelo MPNet são analisados de acordo com seu resultado, de modo que é possível analisá-los separadamente.

Espera-se que o MPNet com RN apresente um tempo de execução médio maior para trajetórias que resultaram em falha e em um replanejamento mal sucedido. Isto se deve aos seguintes fatos: no caso de falhas, elas somente são acusadas após a execução de N iterações permitidas para o planejamento bidirecional; já no caso dos replanejamentos mal sucedidos, tais resultados somente são detectados após concluídas as N_{rp} tentativas previstas para tal processo.

Além disso, é esperado que o menor tempo médio de execução seja encontrado nas trajetórias que resultaram em sucesso, devido ao fato de que não houve necessidade de executar todas as N iterações no planejamento bidirecional ou de realizar um replanejamento.

A Tabela 5.5 descreve a média de tempo de execução que o MPNet gasta para gerar um resultado em cenários vistos e novos.

Tabela 5.5 – Tempo de execução médio em segundos e desvio padrão (entre parênteses) para as trajetórias selecionadas

Cenários	Sucesso	Falha	Rep. com sucesso	Rep. com falha	Total
Vistos	0.01 (0.10)	0.37 (0.42)	0.16 (0.24)	0.49 (0.57)	0.07 (0.24)
Novos	0.01 (0.15)	0.26 (0.33)	0.15 (0.36)	0.36 (0.56)	0.06 (0.25)

Conforme esperado, o tempo gasto com trajetórias que resultam em replanejamentos mal

sucedidos é maior do que com outros resultados, assim como o tempo gasto com trajetórias que resultam em sucesso é o menor. Além disso, percebe-se que o tempo gasto em cenários inéditos é inferior ao do gasto em cenários conhecidos. Esta diferença é maior para as trajetórias que resultaram em falha e em replanejamento mal sucedida. Acredita-se que este é o caso devido à baixa quantidade de amostras para os dois resultados em cada tipo de cenário, conforme a Tabela 5.3 descreve. Visto que aproximadamente 70% dos resultados são de sucesso, o tempo médio geral das amostras é influenciado por tais resultados, porém, ainda é possível concluir que o MPNet com RN processa a trajetória completa em menos de um segundo para os cenários utilizados neste trabalho.

A média de tempo gasto por ambos planejadores para todas as trajetórias selecionadas é descrita pela Tabela 5.6.

Tabela 5.6 – Tempo de execução médio em segundos e desvio padrão (entre parênteses) dos planejadores MPNet e RRT* para as trajetórias selecionadas

Planejador	Tempo médio
MPNet	0.06 (0.24)
RRT*	2.96 (1.27)

Observa-se que o MPNet demora quase 50 vezes menos tempo do que o RRT*. Apesar disso, não há garantia de que a trajetória gerada pelo MPNet com RN seja ótima, conforme observado na seção 5.2.

Neste trabalho, não se utilizou o RH, o motivo, no caso, é avaliar o desempenho da MPNet sem auxílio de outros planejadores. O uso de RH serve como garantia caso não seja encontrada uma trajetória válida, de modo que um planejador oráculo possa ser utilizado para encontrar novos estados críticos que a validem. Se requisitado, o RH aumenta o tempo de processamento do MPNet, podendo resultar em um processamento mais lento do que se o planejador oráculo fosse utilizado diretamente ao invés do MPNet. No entanto, como visto na Tabela 5.3, tais resultados são incomuns.

6 Conclusão e Trabalhos Futuros

O presente trabalho efetuou a investigação da abordagem de planejamento de trajetórias MPNet com RN para ambientes Simples 2D, sendo o agente uma partícula capaz de se locomover verticalmente, horizontalmente e diagonalmente pelo ambiente. Para tanto, cenários foram gerados algoritmicamente e utilizou-se o planejador RRT* na geração de trajetórias nestes cenários. Tais dados foram utilizados para treinar as redes neurais ENet e PNet, que são utilizados na estratégia de planejamento bidirecional empregada na MPNet.

Após o treinamento das redes neurais, implementou-se o algoritmo de planejamento bidirecional, que gera estados críticos através da PNet, alternando os estados de origem e de objetivo até que seja possível conectá-los. Os estados críticos gerados são conectados, formando uma trajetória que é otimizada e validada. Caso a trajetória seja inválida, é realizado o replanejamento excluindo os estados que colidem com obstáculos e gerando novos estados críticos através do algoritmo de planejamento bidirecional.

A partir desta implementação, analisou-se a taxa de sucesso na geração de trajetórias válidas obtida e o tempo de execução do MPNet. Os resultados obtidos evidenciam que o MPNet é consistente em gerar trajetórias válidas para cenários Simples 2D, além de ser significantemente mais rápido que o RRT* no seu processamento.

Analisando o comportamento de cada etapa no processamento do MPNet, é possível observar pontos com potencial de aprimoramento. A representação reconstruída dos cenários gerados pelo CAE não engloba totalmente os obstáculos presentes no cenário, podendo resultar na geração de estados críticos inválidos pela PNet. Além disso, o LSC otimiza a trajetória priorizando a ordem dos estados críticos na trajetória, o que pode ocasionar em falhas em determinadas trajetórias ou em trajetórias subotimizadas.

Com o objetivo de aumentar a referida taxa de sucesso, propõe-se a investigação das seguintes alternativas futuramente:

- Aprimoramento do AE utilizado na ENet, buscando uma alternativa que seja capaz de reconstruir os obstáculos com maior precisão. Deve-se estudar qual seria o efeito na geração de estados críticos da PNet;
- Uso de um algoritmo baseado em heurística para otimização de trajetórias, substituindo o LSC. Deve-se investigar se essa alteração provoca aumentos no tempo de execução do MPNet;
- Investigar performance do MPNet em ambientes tridimensionais, dinâmicos e com obstáculos de tamanhos variáveis;

- Investigar performance do MPNet com dados do mundo real, utilizando sensores na percepção do ambiente;

O desenvolvimento do presente trabalho representou uma oportunidade valiosa para o autor de expandir e aprofundar seus conhecimentos em uma área que lhe despertou particular interesse durante a Graduação: Aprendizagem de Máquina aplicado à Robótica. De fato, o cumprimento da proposta exigiu do mesmo um grande esforço para assimilar várias técnicas e teorias fundamentais no corrente estado da arte da Inteligência Artificial concebidas para lidar com problemas relevantes da vida moderna. A experiência na condução deste trabalho abriu um leque de objetivos e desafios novos que representa um divisor de águas na formação acadêmica do autor.

Referências

ALAMI, R.; LAUMOND, J.; SIMÉON, T. Two manipulation planning algorithms, 1995.

AVENDANO-VALENCIA, L.D.; DIMEAS, Fotios; ASPRAGATHOS, Nikos. Human - Robot collision detection and identification based on fuzzy and time series modelling. **Robotica**, v. 33, mai. 2014. DOI: 10.1017/S0263574714001143.

BANK, Dor; KOENIGSTEIN, Noam; GIRYES, Raja. Autoencoders. **arXiv e-prints**, 2020. Disponível em: https://arxiv.org/abs/2003.05991.

BERTSEKAS, D.P. **Dynamic Programming and Optimal Control**. [*S. l.*]: Athena Scientific, 2005. (Athena Scientific optimization and computation series, v. 2). ISBN 9781886529304.

BHARDWAJ, Mohak; CHOUDHURY, Sanjiban; SCHERER, Sebastian A. Learning Heuristic Search via Imitation. **CoRR**, abs/1707.03034, 2017. arXiv: 1707.03034. Disponível em: http://arxiv.org/abs/1707.03034.

BRADY, Michael. Artificial Intelligence and Robotics. **Artif. Intell.**, v. 26, p. 79–121, 1985.

DEB, Sayantini. **Restricted Boltzmann Machine Tutorial | Deep Learning Concepts | Edureka**. [*S. l.: s. n.*], mai. 2020. Disponível em: https://www.edureka.co/blog/restricted-boltzmann-machine-tutorial/. Acesso em: 2021.

DUCHI, John; HAZAN, Elad; SINGER, Yoram. Adaptive subgradient methods for online learning and stochastic optimization. **Journal of Machine Learning Research**, v. 12, p. 2121–2159, 2011.

FALCON, WA *et al.* PyTorch Lightning. **GitHub**, v. 3, 2019. Disponível em: https://github.com/PyTorchLightning/pytorch-lightning. Acesso em: 2021.

FARIA, Matheus Prado Prandini *et al.* **Investigação de métodos de aprendizagem e de representação de ambiente na construção de agentes jogadores: aplicação ao jogo FIFA**. 2020. 168 f. Dissertação (Mestrado em Ciências da Computação) – Universidade Federal de Uberlândia, Uberlândia, 2020.

FERGUSON, Dave; HOWARD, Thomas M.; LIKHACHEV, Maxim. **Motion planning in urban environments: Part II**. [*S. l.: s. n.*], set. 2008. P. 1070–1076. DOI: 10.1109/IROS.2008.4651124.

GAMMELL, Jonathan D.; SRINIVASA, Siddhartha S.; BARFOOT, Timothy D. Batch Informed Trees (BIT*): Sampling-based optimal planning via the heuristically guided search of implicit

random geometric graphs. *In:* 2015 IEEE International Conference on Robotics and Automation (ICRA). [*S. l.: s. n.*], 2015. P. 3067–3074. DOI: 10.1109/ICRA.2015.7139620.

GAMMELL, Jonathan D.; SRINIVASA, Siddhartha S.; BARFOOT, Timothy D. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. **2014 IEEE/RSJ International Conference on Intelligent Robots and Systems**, IEEE, set. 2014. DOI: 10.1109/iros.2014.6942976. Disponível em: http://dx.doi.org/10.1109/IROS.2014.6942976.

GARDNER, M.W; DORLING, S.R. Artificial neural networks (the multilayer perceptron)—a review of applications in the atmospheric sciences. **Atmospheric Environment**, v. 32, n. 14, p. 2627–2636, 1998. ISSN 1352-2310. DOI: https://doi.org/10.1016/S1352-2310(97)00447-0.

GOOGLE. **Compute Engine: Virtual Machines (VMs) | Google Cloud**. [*S. l.: s. n.*], 2021. Disponível em: https://cloud.google.com/compute. Acesso em: 2021.

HADIDI, R. *et al.* Distributed Perception by Collaborative Robots. **IEEE Robotics and Automation Letters**, v. 3, n. 4, p. 3709–3716, 2018. DOI: 10.1109/LRA.2018.2856261.

HAGHTALAB, Nika *et al.* **The Provable Virtue of Laziness in Motion Planning**. [*S. l.: s. n.*], 2017. arXiv: 1710.04101 [cs.RO].

HAUSER, K.; NG-THOW-HING, V. Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts. *In:* 2010 IEEE International Conference on Robotics and Automation. [*S. l.: s. n.*], 2010. P. 2493–2498. DOI: 10.1109/ROBOT.2010.5509683.

HE, Kaiming *et al.* **Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification**. [*S. l.: s. n.*], 2015. arXiv: 1502.01852 [cs.CV].

HINTON, Geoffrey E.; OSINDERO, Simon; TEH, Yee-Whye. A Fast Learning Algorithm for Deep Belief Nets. **Neural Comput.**, MIT Press, Cambridge, MA, USA, v. 18, n. 7, p. 1527–1554, jul. 2006. ISSN 0899-7667. DOI: 10.1162/neco.2006.18.7.1527. Disponível em: https://doi.org/10.1162/neco.2006.18.7.1527.

HOU, Brian; SRINIVASA, S. Deep Conditional Generative Models for Heuristic Search on Graphs with Expensive-to-Evaluate Edges, 2018.

HURMUZLU, Y.; NWOKAH, O.D.I. **The Mechanical Systems Design Handbook: Modeling, Measurement, and Control**. [*S. l.*]: CRC Press, 2017. (Electrical Engineering Handbook). ISBN 9781420036749.

ICHTER, Brian; HARRISON, James; PAVONE, Marco. Learning Sampling Distributions for Robot Motion Planning. **CoRR**, abs/1709.05448, 2017. arXiv: 1709.05448. Disponível em: http://arxiv.org/abs/1709.05448.

ICHTER, Brian; PAVONE, Marco. **Robot Motion Planning in Learned Latent Spaces**. [*S. l.: s. n.*], 2018. arXiv: 1807.10366 [cs.RO].

KARAMAN, Sertac; FRAZZOLI, Emilio. Sampling-based algorithms for optimal motion planning. **The international journal of robotics research**, Sage Publications Sage UK: London, England, v. 30, n. 7, p. 846–894, 2011.

KAUR, K.; KUMAR, Y. Swarm Intelligence and its applications towards Various Computing: A Systematic Review. *In:* 2020 International Conference on Intelligent Engineering and Management (ICIEM). [*S. l.: s. n.*], 2020. P. 57–62. DOI: 10.1109/ICIEM48762.2020.9160177.

KURZER, Karl. **Path Planning in Unstructured Environments: A Real-time Hybrid A* Implementation for Fast and Deterministic Path Generation for the KTH Research Concept Vehicle**. Dez. 2016. Tese (Doutorado). DOI: 10.13140/RG.2.2.10091.49444.

LATOMBE, Jean-Claude; KAVRAKI, Lydia E. Probabilistic roadmaps for robot path planning. **Practical motion planning in robotics: current aproaches and future challenges**, Citeseer, p. 33–53, 1998.

LAVALLE, Steven M. **Planning Algorithms**. [*S. l.*]: Cambridge University Press, 2006. DOI: 10.1017/CBO9780511546877.

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

MARÍN, Pablo *et al.* Global and Local Path Planning Study in a ROS-Based Research Platform for Autonomous Vehicles. **Journal of Advanced Transportation**, v. 2018, p. 1–10, fev. 2018. DOI: 10.1155/2018/6392697.

MATTHIAS, Bjoern *et al.* Safety of collaborative industrial robots: Certification possibilities for a collaborative assembly robot concept. **IEEE International Symposium on Assembly and Manufacturing (ISAM)**, p. 1–6, mai. 2011. DOI: 10.1109/ISAM.2011.5942307.

MCDERMOTT, Drew. Automated Planning. *In:* ENCYCLOPEDIA of Computer Science. GBR: John Wiley e Sons Ltd., 2003. P. 117–119. ISBN 0470864125.

MCNAUGHTON, M. *et al.* Motion planning for autonomous driving with a conformal spatiotemporal lattice. *In:* 2011 IEEE International Conference on Robotics and Automation. [*S. l.: s. n.*], 2011. P. 4889–4895. DOI: 10.1109/ICRA.2011.5980223.

MORANDO, Mark *et al.* Studying the Safety Impact of Autonomous Vehicles Using Simulation-Based Surrogate Safety Measures. **Journal of advanced transportation**, v. 2018, fev. 2018. DOI: 10.1155/2018/6135183.

MOURET, J. -.; DONCIEUX, S. Encouraging Behavioral Diversity in Evolutionary Robotics: An Empirical Study. **Evolutionary Computation**, v. 20, n. 1, p. 91–133, 2012. DOI: 10.1162/EVCO_a_00048.

OKUDA, R.; KAJIWARA, Y.; TERASHIMA, K. A survey of technical trend of ADAS and autonomous driving. *In:* TECHNICAL Papers of 2014 International Symposium on VLSI Design, Automation and Test. [*S. l.: s. n.*], 2014. P. 1–4. DOI: 10.1109/VLSI-DAT.2014.6834940.

PARK, Jong-Hun; HUH, Uk-Youl. Path Planning for Autonomous Mobile Robot Based on Safe Space. **Journal of Electrical Engineering and Technology**, v. 11, p. 1441–1448, set. 2016. DOI: 10.5370/JEET.2016.11.5.1441.

PASZKE, Adam *et al.* PyTorch: An Imperative Style, High-Performance Deep Learning Library. *In:* WALLACH, H. *et al.* (Ed.). **Advances in Neural Information Processing Systems 32**. [*S. l.*]: Curran Associates, Inc., 2019. P. 8024–8035. Disponível em: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

PAWAR, Karishma; ATTAR, Vahida Z. Assessment of Autoencoder Architectures for Data Representation. *In:* **Deep Learning: Concepts and Architectures**. Edição: Witold Pedrycz e Shyi-Ming Chen. Cham: Springer International Publishing, 2020. P. 101–132. ISBN 978-3-030-31756-0. DOI: 10.1007/978-3-030-31756-0_4. Disponível em: https://doi.org/10.1007/978-3-030-31756-0_4.

PENG, Xishuai *et al.* Traffic sign recognition with transfer learning. *In:* 2017 IEEE Symposium Series on Computational Intelligence (SSCI). [*S. l.: s. n.*], 2017. P. 1–7. DOI: 10.1109/SSCI.2017.8285332.

PUTZ, P.; ELFVING, A. A DEVELOPMENT METHODOLOGY FOR SPACE AR CONTROL SYSTEMS. *In:* DEBRA, D.B.; GOTTZEIN, E. (Ed.). **Automatic Control in Aerospace 1992**. Oxford: Pergamon, 1993. (IFAC Symposia Series). P. 363–368. ISBN 978-0-08-041715-8. DOI: https://doi.org/10.1016/B978-0-08-041715-8.50052-3. Disponível em: https://www.sciencedirect.com/science/article/pii/B9780080417158500523.

QURESHI, Ahmed H.; MIAO, Yinglong *et al.* Motion Planning Networks: Bridging the Gap Between Learning-based and Classical Motion Planners. **arXiv e-prints**, arXiv:1907.06013, arxiv:1907.06013, jul. 2019. arXiv: 1907.06013 [cs.RO].

QURESHI, Ahmed Hussain; AYAZ, Yasar. Intelligent bidirectional rapidly-exploring random trees for optimal motion planning in complex cluttered environments. **Robotics and Autonomous Systems**, v. 68, p. 1–11, 2015. ISSN 0921-8890. DOI: https://doi.org/10.1016/j.robot.2015.02.007.

RIFAI, Salah *et al.* Contractive Auto-Encoders: Explicit Invariance During Feature Extraction. *In:*

ROBLA-GÓMEZ, S. *et al.* Working Together: A Review on Safe Human-Robot Collaboration in Industrial Environments. **IEEE Access**, v. 5, p. 26754–26773, 2017. DOI: 10.1109/ACCESS.2017.2773127.

RUMELHART, David E; HINTON, Geoffrey E; WILLIAMS, Ronald J. Learning representations by back-propagating errors. **Nature**, Nature Publishing Group, v. 323, n. 6088, p. 533–536, 1986.

RUSSELL, Stuart; NORVIG, Peter. **Artificial Intelligence: A Modern Approach**. 3rd. USA: Prentice Hall Press, 2009. ISBN 0136042597.

SAGHEER, Alaa; KOTB, Mostafa. Unsupervised Pre-training of a Deep LSTM-based Stacked Autoencoder for Multivariate Time Series Forecasting Problems. **Scientific Reports**, v. 9, n. 1, p. 19038, dez. 2019. ISSN 2045-2322.

SILVA, IN da; SPATTI, Danilo Hernane; FLAUZINO, Rogério Andrade. Redes neurais artificiais para engenharia e ciências aplicadas. **São Paulo: Artliber**, v. 23, n. 5, p. 33–111, 2010.

SMOLENSKY, Paul. Information processing in dynamical systems: Foundations of harmony theory. **Parallel Distributed Process**, v. 1, jan. 1986.

SNIEDOVICH, M. Dijkstra's algorithm revisited: the dynamic programming connexion. **Control and Cybernetics**, Vol. 35, no 3, p. 599–620, 2006.

SRIVASTAVA, Nitish *et al.* Dropout: A Simple Way to Prevent Neural Networks from Overfitting. **Journal of Machine Learning Research**, v. 15, p. 1929–1958, jun. 2014.

TAMAR, Aviv *et al.* **Value Iteration Networks**. [*S. l.: s. n.*], 2017. arXiv: 1602.02867 [cs.AI].

TOMAZ, Lídia Bononi Paiva *et al.* **ADABA: uma nova abordagem de distribuição do Alfa-Beta-aplicação ao domınio do jogo de Damas**. 2018. 196 f. Tese (Doutorado em Ciências da Computação) – Universidade Federal de Uberlândia, Uberlândia, 2018.

WEN, Shuhuan *et al.* Path planning for active SLAM based on deep reinforcement learning under unknown environments. **Intelligent Service Robotics**, v. 13, abr. 2020. DOI: 10.1007/s11370-019-00310-w.

WULFMEIER, Markus *et al.* Large-scale cost function learning for path planning using deep inverse reinforcement learning. **The International Journal of Robotics Research**, v. 36, n. 10, p. 1073–1087, 2017. DOI: 10.1177/0278364917722396.

YAN, Y.; MOSTOFI, Y. Communication and path planning strategies of a robotic coverage operation. *In:* 2013 American Control Conference. [*S. l.: s. n.*], 2013. P. 860–866. DOI: 10.1109/ACC.2013.6579944.

YU, J. J. Q.; HILL, D. J. *et al.* Intelligent Time-Adaptive Transient Stability Assessment System. **IEEE Transactions on Power Systems**, v. 33, n. 1, p. 1049–1058, 2018.

YU, Jinglun; SU, Yuancheng; LIAO, Yifan. The Path Planning of Mobile Robot by Neural Networks and Hierarchical Reinforcement Learning. **Frontiers in Neurorobotics**, v. 14, p. 63, 2020. ISSN 1662-5218. DOI: 10.3389/fnbot.2020.00063.

ZHANG, Guijuan; LIU, Yang; JIN, Xiaoning. A survey of autoencoder-based recommender systems. **Frontiers of Computer Science**, v. 14, n. 2, p. 430–450, abr. 2020. ISSN 2095-2236.