TRABALHO DE GRADUAÇÃO

APRENDIZADO AUTOMÁTICO UTILIZANDO UM MODELO LSTM APLICADO COMO AUXILIAR NO CONTROLE DE ORIENTAÇÃO E VELOCIDADE DE ROBÔ MÓVEL

Thúlio Noslen Silva Santos

Brasília, julho de 2019

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASILIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

APRENDIZADO AUTOMÁTICO UTILIZANDO UM MODELO LSTM APLICADO COMO AUXILIAR NO CONTROLE DE ORIENTAÇÃO E VELOCIDADE DE ROBÔ MÓVEL

Thúlio Noslen Silva Santos

Relatório submetido como requisito parcial de obtenção de grau de Engenheiro de Controle e Automação

Banca Examinadora

Prof. Flávio de Barros Vidal, CIC/UnB _____
Orientador

Profa. Carla Maria Chagas e Cavalcante Koike, CIC/UnB _____
Co-orientadora

Prof. Marcus Vinicius Lamar, CIC/UnB _____
Examinador interno

Prof. José Maurício Santos Torres da Motta, ENM/UnB _____
Examinador interno

Brasília, julho de 2019

FICHA CATALOGRÁFICA

THÚLIO, NOSLEN SILVA SANTOS
Aprendizado automático utilizando um modelo LSTM aplicado como auxiliar no controle de orientação e velocidade de robô móvel,

[Distrito Federal] 2019.

xvii, 67p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2019). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1. Redes Neurais Recorrentes — 2. LSTM
3. Robótica Móvel — 4. Assistência de Direção

I. Mecatrônica/FT/UnB — II. Título (Série)

REFERÊNCIA BIBLIOGRÁFICA

NOSLEN, THÚLIO. S. S., (2019). Aprendizado automático utilizando um modelo LSTM aplicado como auxiliar no controle de orientação e velocidade de robô móvel. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°$004, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 67p.

CESSÃO DE DIREITOS

AUTOR: Thúlio Noslen Silva Santos

TÍTULO DO TRABALHO DE GRADUAÇÃO: Aprendizado automático utilizando um modelo LSTM aplicado como auxiliar no controle de orientação e velocidade de robô móvel.

GRAU: Engenheiro — ANO: 2019

Thúlio Noslen Silva Santos.

SQS 406 Bloco C, Apartamento 203, Bairro Asa Sul.

70255-030 Brasília – DF – Brasil.

Dedicatória

Dedico este trabalho à todos aqueles que já tiveram problemas com sua saúde mental por conta das pressões da vida, principalmente problemas com depressão. Se você já sofreu ou sofre de depressão, saiba que não está sozinho. Umas das principais lições que aprendi durante a graduação é que nossa saúde mental deve vir sempre em primeiro lugar, e não podemos aceitar trocá-la por nada.

Thúlio Noslen Silva Santos

Agradecimentos

Agradeço a todos que se fizeram presentes durante minha graduação e me proporcionaram bons momentos e conversas. São os momentos e conversas que levarei para o futuro e me sempre vou me recordar com carinho. A vida sem bom humor não é nada.
Aos meus amigos do ensino médio, família Orgutal, um muito obrigado. O que antes era um grupo para combinar idas ao Orgutal se tornou uma amizade para a vida inteira. Gabriel Moreira, Gabriel Rogae e Gabriel Soares: agradeço imensamente à vocês.
Agradeço também aos meus colegas de curso. Alguns, de fato, não apenas colegas, mas amigos de verdade. Em ordem alfabética, para ninguém se sentir desmerecido por ter ficado por último: André Mariano, Christian França, Daniel Amaral, Danielle Almeida, Estanislau Dantas, Gabriell Barbedo, João Lucas, Kenneth Lui, Leonardo da Fonseca, Lucas de Moura, Lucas Schiavini, Tiago Gallo... Muito obrigado por terem feito parte da minha jornada na UnB. Com esperança, muitos também farão parte da minha jornada no resto da minha vida.
Agradeço também à professora Carla, que sempre foi bastante solícita e sempre trata os outros com respeito e empatia. Infelizmente, a empatia com os alunos é um requisto que precisa melhorar nas áreas de engenharia da UnB. Por outro lado, nos anos mais recentes, o diálogo sobre saúde mental está aumentando, e este quadro está se transformando. Minha esperança é que pessoas como a Carla se tornem a regra, e não exceções.
Um agradecimento especial a um grande amigo desde que me entendo por gente, Gabriel Alexandre. Esteve presente em todas as etapas importantes da minha vida, assim como estive nas dele. Muitíssimo obrigado.

Thúlio Noslen Silva Santos

RESUMO

Nos últimos anos, aconteceu uma grande popularização da robótica. Muitos robôs, autônomos ou teleoperados, são usados diariamente em operações repetitivas ou que ofereçam risco aos seres humanos. Entretanto, uma desvantagem dos robôs teleoperados com relação aos robôs autônomos é a necessidade de capacitação de um operador. Assim como houve uma popularização da robótica, recentemente também ocorreu uma grande popularização das redes neurais profundas, abrindo-se assim um espaço para pesquisas em inteligência artificial aplicada à robótica. Sabendo que as redes LSTM têm boa capacidade em aprender sequências e sabendo também que perfil de pilotagem de um operador pode ser visto como uma sequência temporal de comandos, este trabalho propõe o treinamento de uma LSTM profunda a fim de criar um módulo de auxílio de direção a partir de dados de pilotagem coletados de um operador experiente. Mais especificamente, este trabalho tem três objetivos. O primeiro objetivo é criar uma base de dados com dados de pilotagem de um operador experiente, considerada como sendo a pilotagem ideal, para o robô Pioneer 3-AT. O segundo objetivo é propor, treinar e validar arquiteturas de LSTM profunda que consigam aprender os padrões da pilotagem ideal. Por fim, o terceiro objetivo é propor e validar um algoritmo que faça correções em tempo real na pilotagem de um usuário que nunca pilotou o Pioneer. Depois de vários experimentos, construiu-se uma base de dados composta de dados de odometria e dos comandos de velocidade de um operador experiente com o robô, e as arquiteturas propostas foram treinadas e validadas. Isso por sua vez mostrou que uma LSTM profunda consegue aprender os padrões da pilotagem ideal. Os melhores modelos obtidos foram então testados no algoritmo de correção, que consistiu em escolher entre o comando do usuário e o comando da rede com base na diferença entre os dois. Com isto, o algoritmo foi validado em testes e entrevistas com usuários sem experiência de pilotagem do robô. Destas entrevistas e do acompanhamento dos testes, pôde-se verificar que as correções feitas pelo algoritmo impuseram movimentos mais suaves aos usuários, ainda que algumas pessoas não se sentissem confortáveis com as correções impostas.

Palavras Chave: Redes Neurais Recorrentes, LSTM, Robótica Móvel, Assistência de Direção

ABSTRACT

Over the last years, there was a massive popularization of robotics. Many robots, autonomous or teleoperated, are used daily for tasks too dangerous or too repetitive for humans. However, one disadvantage of teleoperated robots against autonomous robots is the need to train an operator. Just as there was a popularization of robotics, in recent years there was algo a massive popularization of deep neural networks, opening the way for research in artificial intelligence applied to robotics. Knowing that LSTM networks have the potential to learn sequences and also knowing that a pilot's driving profile can be seen as a temporal sequence of commands, this thesis proposes the use of a deep LSTM network in order to create a steering assistance module from data collected from an experienced pilot. More specifically, this thesis has three obcjetives. The first objective is to create a database composed of driving data from an experienced pilot, considered as the ideal driving, for the Pioneer 3-AT robot. The second obcjetive is to propose, train and validate deep LSTM networks that can learn the patterns of the ideal driving. Lastly, the third objective is to propose and validate an algorithm that corrects the driving of an inexperienced user in real time. After many experiments, a database was constructed, composed of data from the robots' odometry and the experienced operator's commands, and the proposed architectures were trained and validated. This in turn showed that a deep LSTM network can learn the patterns of the ideal driving. The best models obtained were then tested on the real time correction algorithm, which consists of choosing between the pilot's command and the network's suggestion based on the difference of the two. Thus, the algorithm was validated on tests and interviews with people inexperienced in driving the robot. From these interviews and test follow-up, is was verified that the corrections made by the algorithm imposed smoother movements on the users' driving, although some people did not feel comfortable with the imposed corrections.

Keywords: Recurrent Neural Networks, LSTM, Mobile Robotics, Steering Assistance

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos Latinos

v	Velocidade linear	$[m/s]$
a	Aceleração linear	$[m/s^2]$
Δt_{ns}	Tempo decorrido entre publicações de odometria	$[ns]$
s	Vetor de contexto do robô	
jp	Comando do piloto	
jn	Comando da rede	
jm	Comando médio entre o piloto e a rede	
jo	Comando enviado ao robô	
t_{pub}	Tempo de publicação	$[s]$
t_{pred}	Tempo de predição	$[s]$
pa	Percentual de atuação do algoritmo	[%]
$diffm$	Diferença média ao longo do teste	[%]

Símbolos Gregos

ω	Velocidade angular	$[rad/s]$
α	Aceleração angular	$[rad/s^2]$

Grupos Adimensionais

i, k	Contador
N	Limite do contador

Subscritos

x	pertinente ao eixo x
y	pertinente ao eixo y
m	Valor médio

Siglas

LSTM	*Long Short-Term Memory*
ROS	*Robotic Operating System*
CPU	Unidade de Processamento Central – *Central Processing Unit*
GPU	Unidade de Processamento Gráfico – *Graphics Processing Unit*
UnB	Universidade de Brasília
CIC	Ciência da Computação
IMU	Unidade de Medida Inercial – *Inertial Measurement Unit*
SLAM	*Simultaneous Mapping and Localization*
SGD	*Stochastic Gradient Descent*
MSE	Erro Quadrático Médio – *Mean Squared Error*
RMSE	Raiz do Erro Quadrático Médio – *Root Mean Squared Error*
NRMSE	Raiz do Erro Quadrático Médio Normalizado – *Normalized Root Mean Squared Error*
D	Método de correção desligado
R5	Método de correção por rede, limiar de 5%
R20	Método de correção por rede, limiar de 20%
M5	Método de correção por média, limiar de 5%
M20	Método de correção por média, limiar de 20%

Capítulo 1

Introdução

1.1 Contextualização

Um robô é uma máquina programável capaz de realizar vários tipos de tarefas. Eles podem ser guiados por um controle externo ou serem controlados por computadores embarcados. Alguns podem ser construídos em uma forma similar a um ser humano, mas a grande parte é construído levando-se em consideração apenas a atividade a ser realizada, e não sua aparência.

Dentro da robótica, existem os robôs teleoperados e os robôs autônomos. Um robô teleoperador é controlado por um operador externo que toma todas as decisões, e um robô autônomo, por outro lado, toma suas decisões internamente sem influência de um operador externo. Os robôs teleoperados geralmente necessitam de um operador capacitado para sua correta operação.

Esta necessidade de um operador capacitado pode ser custosa. Veículos robóticos teleoperados geralmente não são equipamentos baratos e são equipamentos pesados que podem ferir gravemente um ser humano, então precisam de bastante cuidado no manuseio.

De forma a reduzir o tempo investido na capacitação de operadores, fabricantes de robôs contemplam uma fusão de teleoperação com autonomia, em que as decisões têm tanto influência interna dos algoritmos do robô quanto do operador. Um exemplo disto são drones, que podem ser controlados por um operador mas têm seu controle de estabilidade feitos por um controlador ou algoritmo secundário.

Os robôs puramente autônomos podem ser controlados por algoritmos comuns, em que são criados conjuntos de regras para sua operação, ou podem ser controlados por algoritmos de aprendizado de máquina. A área de aprendizado de máquina estuda algoritmos e modelos estatísticos que permitem a execução de instruções sem a programação explícita. Uma rede neural artificial é um dos tipos de modelos existentes na área de aprendizado de máquina.

Redes neurais recorrentes são um tipo específico de rede neural artificial em que existe uma memória nas conexões entre os neurônios, o que permite que a rede tenha comportamento dinâmico no tempo. Dentro desta categoria existem as redes LSTM, ou *long short-term memory*, que fazem utilização de um estado interno, ou memória, em sua arquitetura.

Estas redes são usadas em tarefas como o reconhecimento de escrita e fala, que necessitam de um contexto sequencial para que os termos sejam corretamente conhecidos. Um exemplo de aplicação são os corretores automáticos comumente usados nos celulares [5].

Tendo isto mente, neste trabalho será feita uma investigação da capacidade de redes LSTM auxiliarem a pilotagem de um robô tal como um piloto experiente. Com isto, existe uma facilidade maior no desenvolvimento de módulos de direção assistida em veículos, uma vez que o algoritmo não é explicitamente programado.

1.2 Descrição do problema e objetivos

Uma das desvantagens de veículos e robôs teleoperados quando comparados aos veículos e robôs autônomos é o fato de que um operador deve ser treinado para teleoperar o equipamento. Assim como para dirigir um carro deve-se passar pela autoescola, um robô teleoperado exige um operador capacitado. Este pode ser um processo custoso, tanto em termos de dinheiro quanto em termos de tempo, e a depender da complexidade da operação do robô, pode ser difícil achar uma pessoa com coordenação motora adequada.

Sabendo que as redes LSTM têm boa capacidade de aprender padrões com relações temporais, a ideia aqui explorada é a capacidade de uma LSTM aprender a maneira como uma pessoa dirige. As ações que uma pessoa toma ao dirigir são ligadas temporalmente numa sequência, uma vez que o contexto determina, em grande parte do tempo, quais são as próximas ações possíveis.

O objetivo geral deste trabalho é o uso de redes neurais profundas para facilitar a condução de veículos robóticos. São, então, três objetivos específicos:

1. Criar uma base de dados consistindo de dados de pilotagem de um operador experiente;
2. Propor e validar algumas arquiteturas de redes neurais recorrentes com estes dados;
3. Validar, com pilotos inexperientes, métodos de correção que façam uso dos modelos treinados.

1.3 Apresentação do manuscrito

Este manuscrito consiste dos seguintes capítulos:

- Capítulo 1 - Introdução: é o presente capítulo, que contém uma introdução ao tema do trabalho, a motivação da pesquisa e seus objetivos;
- Capítulo 2 - Fundamentos teóricos: neste capítulo, será realizada uma revisão bibliográfica de assuntos relacionados ao trabalho e será oferecido o contexto teórico necessário para o entendimento da pesquisa;
- Capítulo 3 - Metodologia: neste capítulo, serão definidas as técnicas de pesquisa utilizadas, listados os procedimentos experimentais, definidas as propostas as arquiteturas das redes neurais e definida a estrutura do algoritmo de correção da pilotagem;

- Capítulo 4 - Resultados e Discussões: neste capítulo, serão mostrados e discutidos os resultados, que foram a criação de uma base de dados e o treinamento e validação quantitativa e qualitativa das arquiteturas propostas;
- Capítulo 5 - Conclusões: contém um resumo do trabalho feito e as considerações finais.

Capítulo 2

Fundamentos teóricos

2.1 Robótica

A robótica é uma das áreas da engenharia que lida com concepção, desenvolvimento, fabricação e operação de robôs. Geralmente, esta é uma área de estudo multidisciplinar que envolve engenharia mecânica, eletrônica e mecatrônica, além de envolver a ciência da computação.

Esta é uma das áreas de pesquisa com maior crescimento nos últimos anos. Além disso, é uma área que está ganhando cada vez mais espaço dentro de ambiente comerciais. Grandes armazéns estão comprando robôs para que possam organizar a movimentação de materiais de forma eficiente [1], conforme a Figura 2.1. Vários lares hoje têm pequenos robôs de limpeza que circulam pelo ambiente aspirando a sujeira do chão, tais como o robô Roomba [2], da Figura 2.2. Em indústrias, desde o surgimento da Indústria 4.0, temos a presença de vários robôs manipuladores em indústrias montadores realizando trabalhos de alta repetibilidade [3], como os da Figura 2.3.

Figura 2.1: Robô móvel usado em armazéns da Amazon. Fonte: [1]

Robôs são compostos de vários sistemas que interligados permitem seu funcionamento. Existe

Figura 2.2: Robô Roomba, da empresa iRobot. Fonte: [2]

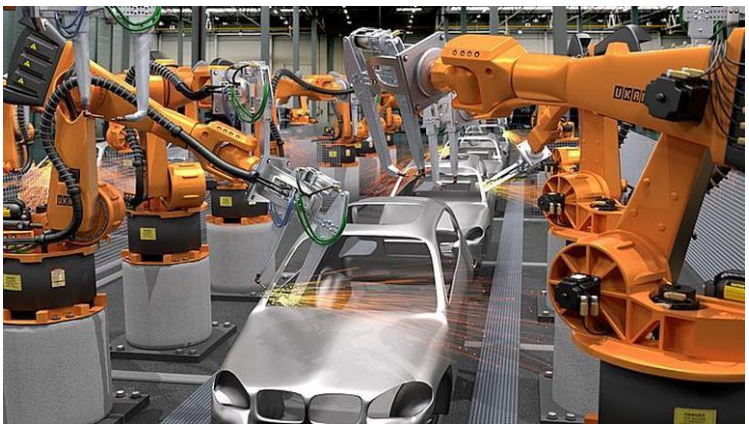

Figura 2.3: Robôs industriais em uma linha de produção de carros. Fonte: [3]

um projeto mecânico que envolve motores e juntas e como o robô se movimenta. Além disso, temos os circuitos eletrônicos do sistema, que dão suporte aos atuadores e sensores. Também temos os computadores usados para programar tais robôs, que podem ou não ser embarcados.

Os robôs são bastante úteis para a realização de tarefas consideradas perigosas ou repetitivas demais para seres humanos. Como exemplos de tarefas repetitivas, temos robôs montadores em indústrias, que podem ser especializados em uma tarefa e operar rapidamente e sem descanso. Para situações perigosas, temos robôs de resgate [6], que podem realizar tarefas junto a socorristas em situações de emergência, como em desabamento de prédios ou incêndios, tal como o robô da Figura 2.4. Temos também robôs de exploração, como as sondas que foram enviadas para Marte [7], na Figura 2.5, e os robôs que são enviados às profundezas dos oceanos da Terra [8]. Em algumas aplicações, têm-se uma economia a longo prazo que justifica o alto investimento inicial no uso de robôs nessas aplicações.

Um dos tipos de robôs existentes é o braço robótico. Outro nome para este tipo de robô é manipulador robótico ou braço manipulador. Geralmente, estes robôs são constituídos de uma estrutura similar a um braço humano e estão fixados a alguma superfície. Estes robôs são muitos

Figura 2.4: Robô de resgate do Corpo de Bombeiros de Tóquio. Fonte: `http://www.eggshell-robotics.com/blog/41-fire-department-robot`

usados em ambientes industriais, tal como mostrado na Figura 2.3, em que há grande controle do ambiente, e este é o tipo de robô mais adequado para atividades que requerem rapidez e precisão na movimentação.

Outro tipo de robô bastante comum na robótica são os robôs móveis. A robótica móvel é focada em robôs que podem se deslocar dentro de um ambiente sem grandes restrições. Como estes robôs podem estar fora de ambiente controlados, existem diferentes necessidades para diferentes tarefas. Assim, nesta área de pesquisa existe uma grande variedade de robôs. Estes robôs vão desde humanoides até robôs diferenciais, de robôs simples até robôs complexos. É possível que um robô móvel tenha como parte de si braços robóticos, tal como a sonda Curiosity na Figura 2.5. Um exemplo adicional de robôs móveis são os carros autônomos da Tesla [9], tal como o da Figura 2.6.

2.1.1 Componentes básicos de robôs móveis

O texto desta e das seguintes subseções está baseado em [10] e [11].

Os componentes básicos de um robô móvel são: um controlador, um software de controle, atuadores e sensores. Estes elementos trabalham de forma conjunta para que o controle do robô seja implementado.

Atuadores são elementos capazes de gerar movimentação em um sistema. Geralmente, este elemento exige uma fonte de energia e um sinal de controle. Pode-se dizer que os atuadores são a parte do robô responsável pela interação com o ambiente. Motores de diversos tipos, como motores lineares, motores de passo ou servomotores entram nesta categoria.

Sensores são elementos capazes de medir características de um sistema. Este elemento pode ou não exigir uma fonte de energia e fornecem informações por algum sinal que geralmente é condicionado antes da leitura. Pode-se dizer que os sensores são a parte do robô responsável pela

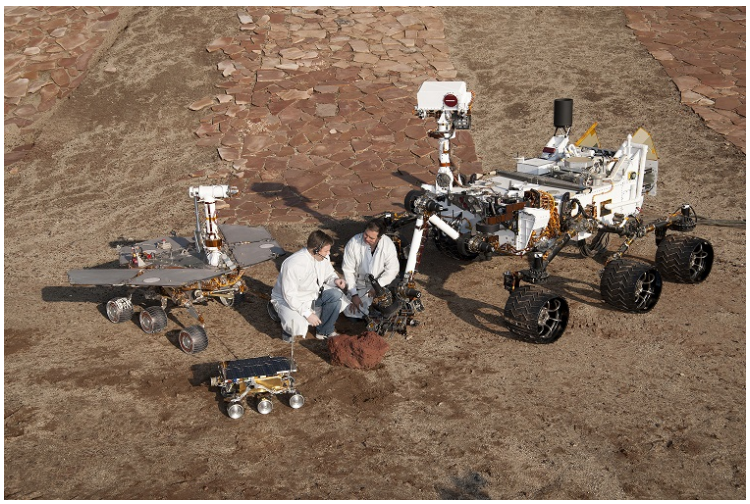

Figura 2.5: Sondas Sojourner, Opportunity e Curiosity, em ordem crescente de tamanho. Fonte: `https://www.nasa.gov/multimedia/imagegallery/image_feature_2154.html`

Figura 2.6: Tesla Model X. Fonte: `http://www.thedrive.com/tech/10090/tesla-wants-drivers-to-share-autopilot-driving-videos-to-help-self-driving-cars`

percepção do ambiente. Nesta categoria, temos sensores como sonares, sensores de distância à laser, odômetros, sensores infravermelhos, sensores de umidade e sensores de nível de tensão da bateria. Os sensores geralmente são conectados a uma unidade de condicionamento de sinal de forma que seu sinal de saída possa ser lido de maneira conveniente por algum um outro módulo ou sistema.

Controladores são elementos que, quando ligados a atuadores e sensores, conseguem atuar no sistema, observando e corrigindo o impacto de suas ações em tempo real. Este é um elemento que, nos casos mais simples, descreve a relação entre o sinal de referência, a saída dos sensores e a entrada dos atuadores. Fora dos ambientes industriais, o controlador é um microcontrolador integrado ou um computador externo conectado ao robô por alguma rede de comunicação. Em robôs móveis, pode-se usar desde pequenos computadores embarcados no robô, como um Raspberry Pi, até computadores externos potentes conectados ao controlador interno do robô por uma rede de comunicação, cabeada ou sem fio.

Por fim, o software de controle é composto pelos algoritmos carregados no controlador. A depender da complexidade do controlador, estes algoritmos podem ser de alto ou baixo nível. Os controladores de baixo nível têm software mais simples, como os controladores que são ligados diretamente nos motores, e podem vir embarcados no robô. Para um controle de trajetória ou de comportamento do robô, temos algoritmos de alto nível que se comunicam com os controladores de baixo nível, e tendem a requerer controladores mais complexos. Sempre que um sistema de controle é criado, são necessários modelos dos atuadores e sensores de forma que informações possam ser inferidas e ações planejadas.

A quantidade e o tipo de sensores e atuadores, assim como a complexidade e a robustez dos controladores dependem da finalidade do robô e do ambiente em que serão realizadas suas tarefas. Um robô destinado à limpeza do chão tem, no mínimo, atuadores para suas rodas e sensores de contato e proximidade para que consiga se locomover no ambiente e para que não se danifique no processo [12]. Seu sistema de controle muitas vezes é simples, requerendo que o robô se locomova aleatoriamente pelo chão com algumas restrições de segurança e outras pequenas tomadas de decisão. Um robô bípede humanoide destinado à pesquisa, por outro lado, tem um grau de complexidade maior, requerendo dezenas de atuadores para todas as suas juntas e dezenas de sensores como câmeras, unidades de medida inercial (IMUs) e sensores de distância a laser. Além disso, o sistema de controle cuida de muitas variáveis ao mesmo tempo, então acaba sendo mais complexo e requer maior poder computacional para execução do software de controle [13].

Traçando um paralelo com sistemas de controle, os atuadores são os elementos que atuam no sistema sem observar os resultados da ação. Os sensores são os elementos que permitem que seja realizada esta observação. Quando se faz a realimentação da saída do sistema para o controlador, este poderá especificar a ação do atuador tendo um resultado específico como objetivo. Um diagrama desta descrição pode ser visto na Figura 2.7.

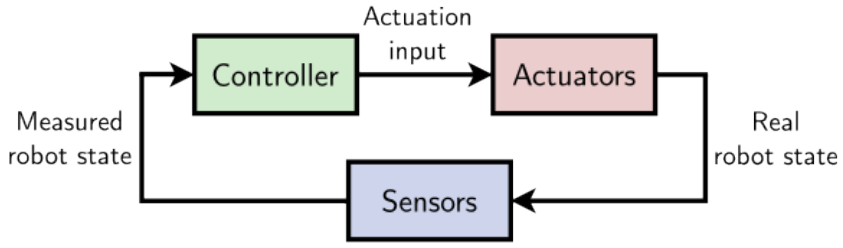

Figura 2.7: Diagrama de sistema com controladores, sensores e atuadores. Fonte: `https://scaron.info/teaching/what-is-a-controller.html`

2.1.2 Tipos de sensores em robôs móveis

Dentre os tipos de sensores para robôs móveis citados anteriormente, analisaremos abaixo os que são relacionados com este trabalho: odômetro, sonares, sensores a laser e câmeras.

2.1.2.1 Odometria

A odometria fornece informações como a posição atual do robô ou o incremento de posição desde o instante anterior. Geralmente, esta informação é o resultado do processamento de um ou vários elementos usados para inferir a movimentação do robô, como IMUs ou *encoders*.

Para o caso de robôs com rodas, a odometria resulta do processamento dos *encoders* presentes no eixo das rodas. Com a medida de quanto a roda girou, o intervalo de tempo entre pulsos do *encoder* e o raio da roda, pode-se determinar o quanto o robô se locomoveu e, utilizando a posição do instante anterior, determinar a posição atual do robô. Na Figura 2.8 temos dois exemplos de *encoders* rotativos.

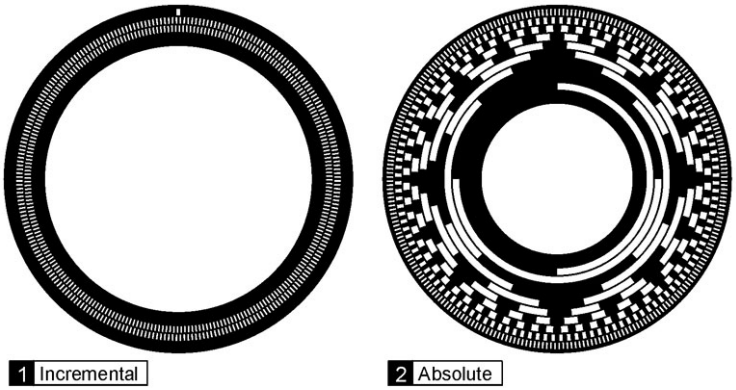

Figura 2.8: Diagramas de dois *encoders*, o incremental e o absoluto. Fonte: `https://www.machinedesign.com/motion-control/what-s-difference-between-absolute-and-incremental-encoders`

Os *encoders* absolutos podem informar a posição angular do motor com bastante precisão. Eles são mais comuns em motores de passo ou servomotores, que não operam com várias voltas e sim dentro de um faixa angular predeterminada. Os *encoders* incrementais são mais simples e os pulsos não informam a posição angular do motor. Eles são mais comuns em motores que podem girar livremente, como é o caso de motores de rodas de robôs móveis.

A Equação (2.1) permite determinar o deslocamento incremental a partir de pulsos consecutivos de um *encoder* incremental:

$$\Delta S = 2\pi R \frac{\alpha_t \Delta n_t}{2\pi} = R\alpha_t \Delta n_t, \tag{2.1}$$

em que ΔS é o deslocamento linear, α_t é ângulo percorrido entre dois pulsos consecutivos, Δn_t é a quantidade de pulsos emitidos no intervalo de tempo considerado e R é o raio da roda. Veja que a informação de tempo é incluída no cálculo de Δn_t.

Assim, com o deslocamento linear de um período de tempo para outro, podemos determinar a posição em um dado instante de tempo com a Equação (2.2):

$$S_t = S_{t-1} + \Delta S, \tag{2.2}$$

em que S_t é a posição no tempo t e S_{t-1} é a posição no tempo $t-1$.

Entretanto, em alguns casos, esta abordagem não é factível. Para robôs humanoides, por exemplo, este processamento seria muito complexo, uma vez que deve-se fazer uma composição da movimentação de todas as juntas para calcular o deslocamento. Uma alternativa mais simples é processar os dados de uma IMU a fim de determinar a posição atual. Uma IMU dá informações de aceleração e velocidade do robô, não de posição no espaço, então é necessário integrar estas informações no tempo para determinar a movimentação do robô. A Equação (2.3) descreve a posição do robô em tempo contínuo:

$$S_t = S_0 + \int_0^t v_{IMU}(\tau)d\tau, \tag{2.3}$$

em que S_0 é a posição inicial e v_{IMU} é a velocidade medida com a IMU. Em tempo discreto, tem-se a Equação (2.4):

$$S_t = S_0 + \sum_{k=0}^{k_t} v_{IMU}(k)\Delta t, \tag{2.4}$$

em que Δt é o intervalo de tempo considerado, k é um instante de tempo qualquer e k_t é o instante de tempo final. A Equação (2.5):

$$S_t = S_{t-1} + v_{IMU}(k_t)\Delta t \tag{2.5}$$

descreve a forma recursiva da Equação (2.4).

2.1.2.2 Sensores de distância

Outra categoria de sensores são os de medição de distância. Os mais comuns são os baseados em ultrassom, que são os sonares, e os baseados em feixes de laser, que são os sensores a laser. A ideia básica deste tipo de sensor é a emissão e captação de algum tipo de onda e determinação da distância a partir da comparação entre a onda emitida e a onda captada.

No caso dos sonares, temos a emissão de ultrassom, que é o som em altas frequências. Um alto falante emite o som e um microfone próximo capta o retorno da onda. Este microfone tem uma certa tolerância para que a própria onda emitida seja ignorada, e sejam capturadas somente as ondas refletidas. Calculando-se a diferença de tempo entre emissão e retorno da onda e sabendo qual é a velocidade do som no meio, pode-se determinar qual a distância do objeto detectado pelo sonar. A Equação (2.6) descreve o cálculo da distância:

$$d = \frac{v_{som}\Delta t}{2}, \tag{2.6}$$

em que d é a distância até o objeto, v_{som} é a velocidade do som no meio e Δt é o intervalo de tempo entre emissão e retorno.

Um diagrama de funcionamento de um único sonar pode ser visto na Figura 2.9. Geralmente, existe um anel de sonares em torno dos robôs de forma que seja possível detectar objetos em várias direções. Um exemplo deste anel pode ser visto na Figura 2.10, em que 6 sonares frontais estão visíveis logo acima da câmera.

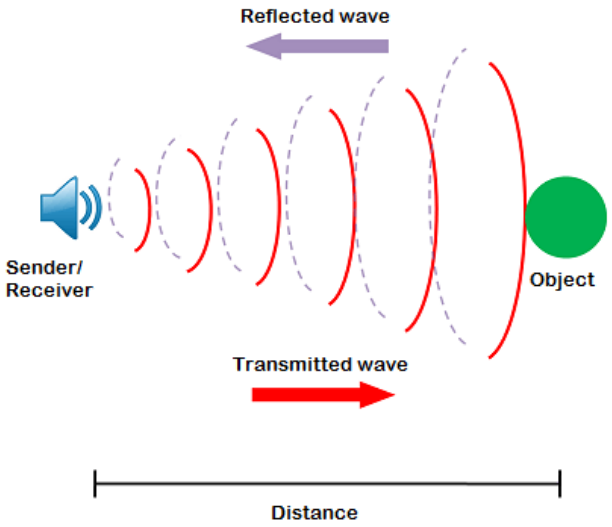

Figura 2.9: Funcionamento de um sonar. Fonte: `http://www.physics-and-radio-electronics.com/blog/sonar-or-sound-navigation-and-ranging/`

Figura 2.10: Robô Pioneer equipado com anel de sonares, sensor a laser, uma câmera e um braço manipulador. Fonte: `https://robots.ieee.org/robots/pioneer/`

Já nos sensores a laser, temos a emissão de feixes de laser contendo uma ou várias frequências distintas. Com base no retorno do feixe, existem 3 opções comuns para se determinar a distância até o obstáculo:

- A primeira se dá como nos sonares: sabendo-se a velocidade da luz e o tempo decorrido entre emissão e retorno, pode-se determinar a distância, mas isso necessita de circuitos de contagem de tempo com precisão de nanosegundos. A Equação (2.7) descreve o cálculo da distância, de maneira similar à do sonar:

$$d = \frac{c\Delta t}{2}, \tag{2.7}$$

em que c é a velocidade da luz.

- A segunda opção existe quando o feixe de laser contém múltiplas frequências. A partir da medição do deslocamento de fase entre a onda emitida e a onda recebida em diferentes frequências, pode-se determinar o tempo decorrido entre a emissão e o retorno com maior confiabilidade. O cálculo da distância se dá como na Equação (2.7), mas o cálculo do tempo se dá como na Equação (2.8):

$$\Delta t = \frac{\phi}{\omega}, \tag{2.8}$$

  em que ϕ é deslocamento de fase e ω é a frequência angular da onda. Ao medir este tempo em diferentes frequências, pode-se eliminar a ambiguidade que surge ao se trabalhar com medidas de natureza angular.

- Por fim, a terceira opção é fazer uso de técnicas de interferometria, quer requer um sistema mais complexo com um interferômetro dentro do laser [14].

Geralmente, um sensor a laser tem um espelho interno giratório que permite que feixes sejam disparados em várias direções em um curto intervalo de tempo. Assim, uma grande área é coberta e com bastante precisão entre feixes consecutivos. Um exemplo de sensor a laser montado em um robô pode ser visto na Figura 2.10. Nesta foto, o sensor é o componente de cor azul montado acima do robô e logo abaixo do braço manipulador.

O sensor a laser geralmente exige mais potência e mais complexidade no processamento de dados, mas consegue fornecer mais informações e de forma mais precisa. Além de sofrer menos com ruído do que os sonares, o sensor a laser também tem distância de detecção muito maior. O uso de cada um deles depende do que a aplicação requer.

2.1.2.3 Câmeras

Alguns robôs possuem câmeras embarcadas, principalmente quando há um hardware potente capaz de realizar o processamento de imagens. Por serem sensores complexos, ainda que requeiram alto poder computacional, são excelentes fontes de dados. Os dados das imagens podem ser tratados com técnicas já estabelecidas de processamento de imagens para realizar o reconhecimento de padrões e objetos ou podem servir de entrada para alguma rede neural, geralmente com camadas convolucionais, para que informações de alto nível sejam extraídas e posteriormente processadas por algum outro algoritmo [15]. Na Figura 2.10, o robô possui uma câmera embarcada, apontando na direção do efetuador do braço robótico.

2.1.3 Cinemática para robôs móveis

Nos algoritmos de controle de baixo nível dos robôs móveis, é preciso atuar diretamente nos motores das rodas ou das juntas do robô para que ele realize suas tarefas. Com isto, é necessário ter conhecimento da relação entre a movimentação dos motores ou juntas e a movimentação do robô no espaço. A estas relações dá-se o nome de cinemática direta e inversa.

2.1.3.1 Cinemática direta

A cinemática direta refere-se ao uso das equações cinemáticas dos componentes do robô a fim de calcular a posição do robô ou do efetuador de um manipulador robótico no espaço. A partir dos estados das juntas e atuadores, o objetivo é determinar qual a posição do robô no espaço.

Quando o modelo cinemático de um robô é criado, geralmente chega-se nas equações da cinemática direta, uma vez que esses modelos são mais simples de descrever. Sabendo-se o que está acontecendo com as juntas e motores, não são necessários algoritmos especiais para se determinar a posição do robô no espaço. Uma cadeia cinemática pode ser descrita como um conjunto de transformações, e basta substituir valores nas equações do modelo para encontrar uma solução. Um exemplo de cadeia cinemática são os braços robóticos.

2.1.3.2 Cinemática inversa

A cinemática inversa refere-se ao cálculo dos estados das juntas e atuadores a fim de se atingir uma determinada posição do robô ou do efetuador de um manipulador no espaço. A partir da posição no espaço desejada para o robô ou o manipulador, é necessário determinar os estados das juntas e atuadores.

Quando se trata de cinemática inversa, o problema pode requerer mais que o modelo cinemático para se obter uma solução. Uma vez que várias combinações de estados de juntas e motores podem levar a uma mesma posição final no espaço, várias soluções são possíveis, mas uma única deve ser escolhida. Este não é um problema tão severo para robôs com rodas simples, mas para robôs humanoides, por exemplo, temos equações de alta complexidade, e determinar soluções para uma equação cinemática inversa pode ser um problema muito custoso para resolver em tempo real [16], e muitas vezes aproximações heurísticas são usadas. Além disso, costuma-se incluir no processo de solução as restrições físicas das juntas e motores [17].

2.1.4 Tipos de modelos cinemáticos

Existem vários tipos de modelos cinemáticos para robôs móveis. Estes se diferenciam pela quantidade de atuadores e juntas e como eles estão conectados entre si. Em robô com rodas, diferentes tipos de rodas podem ser utilizadas. Existem os robôs diferenciais, omnidirecionais, síncronos e robôs com geometria Ackerman, para citar os mais comuns.

Os robôs omnidirecionais, como o nome implica, são robôs que conseguem se mover em qualquer direção. A forma mais usual destes robôs é com o uso de rodas omnidirecionais. Estas são rodas que permitem movimento em direções ortogonais ao sentido de movimento usual da roda. Com 3 ou mais rodas independentes, é possível selecionar individualmente o movimento de cada roda de forma que o robô se desloque em qualquer direção. Na Figura 2.11 temos um exemplo deste tipo de modelo cinemático.

Outro tipo de geometria, comum em robôs de 3 rodas, é a chamada geometria síncrona. Nesta geometria, as rodas são conectadas entre si e a outros 2 motores: um controla a velocidade de

Figura 2.11: Exemplo de robô omnidirecional. Fonte: `https://www.robotshop.com/en/3wd-100mm-omni-wheel-mini-mobile-robot.html`

movimento e outro controla a angulação das rodas em conjunto. A Figura 2.12 mostra um exemplo desta geometria.

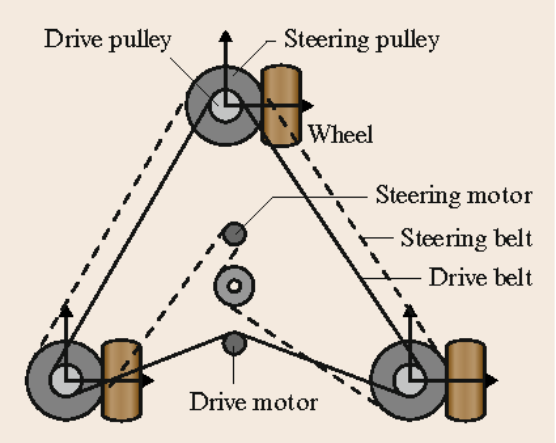

Figura 2.12: Exemplo de geometria síncrona. Fonte: [4]

A geometria Ackerman, ou geometria de Ackerman, é a geometria usada em automóveis [18]. Este é um sistema projetado para resolver o problema da diferença de alinhamento das rodas interiores e exteriores durante a curva de um veículo, e é uma geometria usualmente empregada em sistemas de 4 rodas. Enquanto as rodas traseiras têm direção de movimento fixa e apontando para a frente do veículo, as rodas dianteiras podem ter seu alinhamento alterado, e o alinhamento de cada uma é independente. Com isto, as rodas dianteiras podem se alinhar individualmente com o centro de rotação da curva. Isto é necessário porque o raio de rotação de cada roda com relação ao centro da curva é diferente. Um diagrama deste tipo de geometria pode ser visto na Figura 2.13.

No caso de robôs móveis com essa geometria, é comum ter dois motores, de maneira similar à geometria síncrona: um controla a velocidade de movimentação das rodas traseiras; o segundo controla o ângulo das rodas do eixo dianteiro.

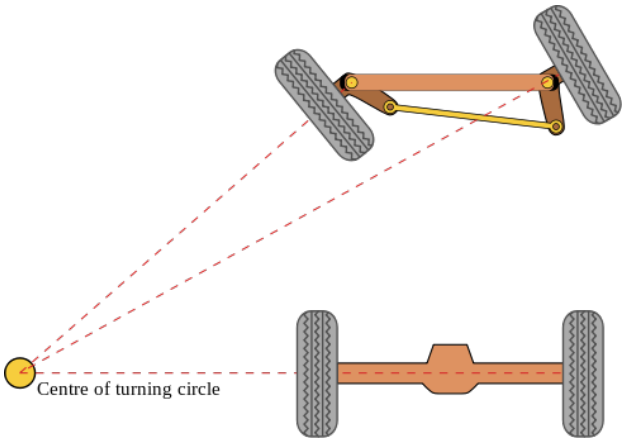

Figura 2.13: Exemplo de geometria Ackerman. Fonte: `https://en.wikipedia.org/wiki/Ackermann_steering_geometry`

2.1.5 Robô diferencial

Robôs diferenciais são robôs móveis, com pelo menos 2 rodas, em que a movimentação de cada roda é independente, mas a angulação das rodas é fixa. Para os casos mais simples, de robôs de 2 rodas, podemos controlar a velocidade angular das rodas de forma independente e com isto a direção do movimento do robô. Para tornar a movimentação mais fácil e não ter que lidar com o problema de ter o equivalente de um pêndulo invertido sobre as 2 rodas, geralmente adiciona-se uma terceira roda, que pode se movimentar em qualquer direção. Um diagrama de geometria diferencial para 2 rodas pode ser visto na Figura 2.14.

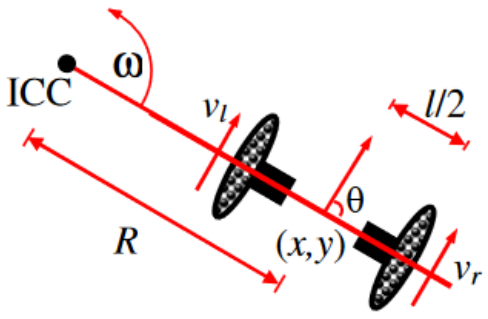

Figura 2.14: Exemplo de robô diferencial de 2 rodas. Fonte: `https://k9-build.blogspot.com/2017/12/where-next-predicting-future-of.html`

No caso de um robô diferencial de 4 rodas, é comum simplificar o controle e conectar pares de rodas do mesmo lado entre si, de forma que o robô seja controlado tal qual um robô de 2 rodas mas sem o problema do equilíbrio sobre 2 rodas. Neste trabalho será utilizado um robô de

4 rodas com geometria diferencial, então este detalhamento receberá mais atenção. Além disso, o robô utilizado é não-holonômico, que significa que ele tem restrições de graus de liberdade em sua movimentação.

Para o caso de um robô diferencial de 4 rodas em que as rodas laterais estão sincronizadas, temos uma simplificação no controle, uma vez que podemos analisar o modelo cinemático considerando que existem apenas 2 rodas virtuais. Além disso, os eixos são tomados com relação ao robô, como na Figura 2.15. As equações cinemáticas são dadas pelas Equações (2.9):

$$v_x = \frac{v_r + v_l}{2} \tag{2.9}$$

e (2.10):

$$\omega_z = \frac{v_r - v_l}{L}, \tag{2.10}$$

em que v_r é a velocidade da roda direita, v_l é a velocidade da roda esquerda, v_x é a velocidade linear no eixo x, ω_z é a velocidade angular no eixo z e L é a distância entre as duas rodas.

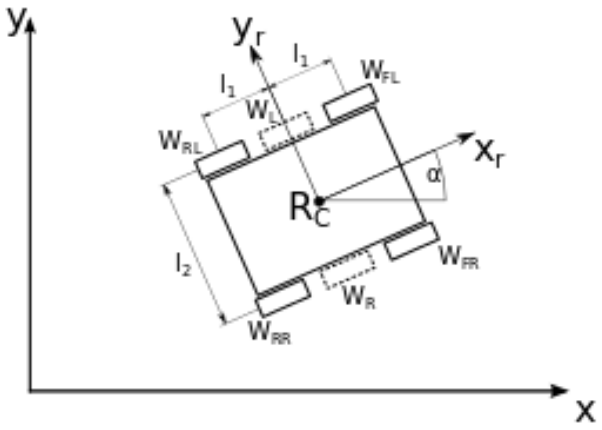

Figura 2.15: Exemplo de eixos para um robô diferencial. Fonte: `https://husarion.com/tutorials/ros-tutorials/3-simple-kinematics-for-mobile-robot/`

Para a cinemática inversa do robô, como temos 2 variáveis de entrada e 2 variáveis de saída, podemos determinar equações exatas. As Equações (2.11):

$$v_r = v_x + \frac{\omega_z L}{2} \tag{2.11}$$

e (2.12):

$$v_l = v_x - \frac{\omega_z L}{2} \tag{2.12}$$

descrevem a relação inversa.

Com isto, o controle de baixo nível deste tipo de robô se dá de maneira simples, e permite que recursos computacionais sejam dedicados a outros algoritmos de alto nível. O modelo pode ser posteriormente refinado relacionando a velocidade angular do motor com a velocidade linear de cada roda.

2.1.6 Teleoperação e autonomia

O texto desta e das seguintes subseções está baseado em [10] e [19].

Robôs móveis podem ser teleoperados, autônomos ou uma fusão dos dois. Um robô teleoperado é controlado por um operador e a tomada de decisões está fora do alcance do robô. Podemos ter, por exemplo, um pequeno carro controlado por uma criança com um controle remoto. Em contraste, um robô completamente autônomo não é controlado por um operador externo, e toma decisões de forma independente ou de acordo com algum conjunto de regras. Um exemplo é um robô destinado a mapear um ambiente sem auxílio de um operador. Dado um certo conjunto de regras para que o robô se mantenha operacional, não há influência de operadores externos em suas tarefas e na tomada de decisão.

Tanto robôs teleoperados quanto robôs autônomos funcionam da mesma forma no que diz respeito aos atuadores. As diferenças entre ambos existem no funcionamento do controle e em que tipo de sensores são necessários para a operação do robô. Para robôs teleoperados, como o controle é feito por um operador externo, geralmente há menos sensores do que em robôs autônomos. Para os autônomos, existe maior necessidade de dados vindos de sensores uma vez que não há um operador externo analisando a situação e tomando as decisões do que fazer. Por consequência, também existe uma necessidade de um controle mais complexo que faça o processamento dos dados sensoriais e defina quais ações tomar em tempo real.

A grande maioria dos robôs móveis possui um controle misto entre essas duas categorias. Estes são robôs em que a tomada de decisões tem influência tanto do controle autônomo do robô quanto de um operador externo. Os exemplos vão desde um drone controlado por controle remoto mas com controle de estabilidade independente do operador até um robô que pode ser teleoperado desde que não seja posto em risco pelo operador; se o robô detectar perigo, como uma colisão iminente com uma parede próxima, ele toma o controle e entra em modo de desvio de obstáculos.

2.1.7 Problemas em robótica autônoma

A natureza dos problemas para robôs teleoperados é diferente da natureza dos problemas para robôs autônomos. Para os robôs teleoperados, as soluções dos problemas são geralmente delegadas ao ser humano controlando o robô, então é necessária a capacitação do operador. Para os robôs autônomos, os problemas muitas vezes tem interseção com o campo de inteligência artificial. Os principais problemas para a operação autônoma são o mapeamento, localização, navegação e localização e mapeamento simultâneos, ou SLAM [10].

2.1.7.1 Mapeamento

O mapeamento se refere-se à criação de uma representação interna do ambiente no robô, equivalente a um mapa. Para isto, assume-se que o robô consegue se localizar dentro do ambiente. Existem dois tipos de mapas comumente usados em robótica: os mapas topológicos e os mapas métricos.

Mapas topológicos são representações de alto nível de um ambiente [20]. A maneira usual de implementação é a criação de regiões dentro do mapa e a associação destas regiões a diferentes nós de um grafo. Com isto, são determinados os arcos ligando os nós e seus pesos. Este peso pode ser determinado de diversas maneiras, e é dado por alguma função de custo de navegação entre os nós, que leva em consideração a distância, inclinação do terreno, dificuldade de navegação na região, entre outros. A criação destes mapas geralmente é delegada a alguma outra entidade que já tem conhecimento do mapa, uma vez que determinar uma divisão ótima de regiões no mapa durante o mapeamento é um problema complexo. Um exemplo de mapa topológico pode ser visto na Figura 2.16.

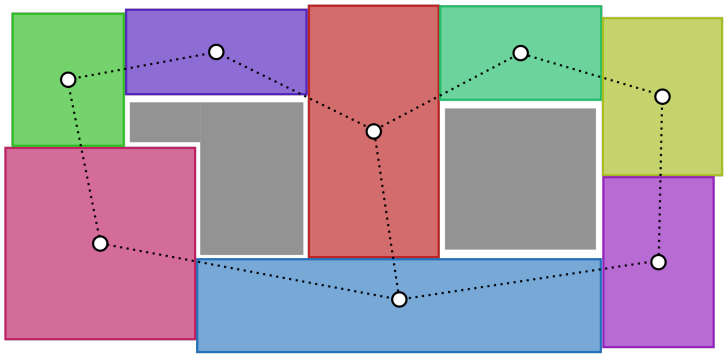

Figura 2.16: Exemplo de mapa topológico. Fonte: `https://www.groundai.com/project/topomap-topological-mapping-and-navigation-based-on-visual-slam-maps/`

Grades de ocupação são representações de baixo nível de um ambiente, e são um tipo de mapa métrico. O tamanho mínimo de uma célula é definido de antemão e uma matriz é criada para representar o ambiente. Geralmente, são designados números entre 0 a 1 ou 0 a 255 para representar a ocupação daquela célula. Este tipo de mapa é construído com base em dados de sensores em tempo real, e pode estar sujeito a erros caso a localização acumule erros com o tempo, como é o caso de localização baseada puramente em odometria. Sabendo-se a posição do robô no mapa e conseguindo decompor as leituras do sensor a laser em posição relativas ao robô, é possível criar um mapa. Existem também outras técnicas, como o uso de câmeras estéreo para criação de um mapa em 3 dimensões [21]. Um exemplo de grade de ocupação em 2 dimensões pode ser visto na Figura 2.17.

2.1.7.2 Localização

A localização, por outro lado, refere-se à determinação da posição do robô no ambiente. Para isto, assume-se que o robô já tem uma representação do ambiente disponível. A maneira mais crua de localização é baseada em *dead reckoning*, que consiste em determinar a posição atual do robô com base na posição anterior e com estimativas de velocidade e tempo decorrido. Isto, no entanto, está sujeito ao acúmulo de erros ao longo do tempo, uma vez que o sistema pode ter limitações numéricas nos cálculos de integração da velocidade e, no caso de um robô com rodas, a roda pode deslizar no chão.

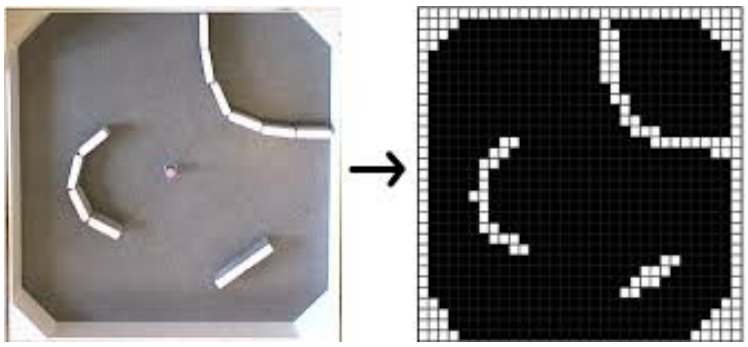

Figura 2.17: Exemplo de transformação de ambiente em grade de ocupação. Fonte: `http://www.ep.liu.se/ecp/048/007/ecp1048007.pdf`

Uma forma mais elaborada de localização envolve a detecção de características no ambiente que possam indicar onde o robô está. Isto requer que o mapa previamente criado tenha estas representações armazenadas. Isto, no entanto, também está sujeito a erros de ambiguidade caso o ambiente tenha locais diferentes com características parecidas. Uma representação desta ambiguidade pode ser vista na Figura 2.18. O robô vê uma porta mas em todo o prédio existem portas, então ele pode estar em vários locais diferentes. Assim, é comum o uso de técnicas probabilistas que levem em conta a movimentação do robô baseada em *dead reckoning* e a probabilidade de que uma certa característica seja detectada naquela posição estimada. Com isto, o robô consegue manter a localização com menor erro por mais tempo. Um outro exemplo é o uso de câmeras estéreo para localização em 3 dimensões [22].

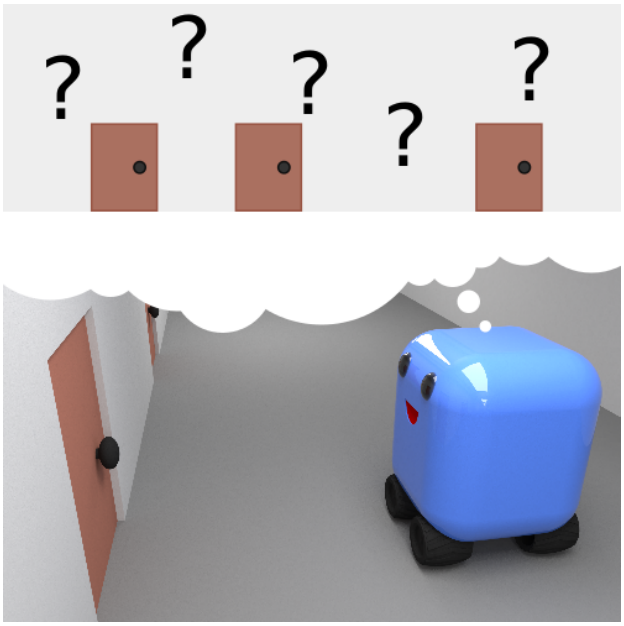

Figura 2.18: Representação do problema da ambiguidade em localização. Fonte: `https://en.wikipedia.org/wiki/Monte_Carlo_localization`

2.1.7.3 Navegação

Depois que um mapa é criado e que o robô consegue localizar-se dentro do ambiente, temos a possibilidade de realizar a navegação dentro do ambiente. Com isto, surgem possibilidades para planejamento de rotas dentro ambiente, por exemplo. Algoritmos como a determinação do caminho mais curto em um grafo têm bastante utilidade na navegação.

2.1.7.4 SLAM

Os problemas de localização e mapeamento são interdependentes. A obtenção de uma solução para um depende da obtenção prévia uma solução para o outro. Isto é uma limitação prática importante, e a este problema dá-se o nome de *simultaneous mapping and localization*, ou SLAM. Apesar de ser um problema NP-completo, em que há solução mas é caro demais obtê-la, existem algoritmos que podem ser executados em tempo real para se obter soluções aproximadas [23].

2.2 Aprendizado profundo

2.2.1 Aprendizado de máquina

O texto desta e das seguintes subseções está baseado em [24] e [25].

O aprendizado de máquina é uma área dentro da inteligência artificial que abrange técnicas de reconhecimento de padrões e análise estatística de dados. Com isto, podem ser criados modelos que aprendam com dados anteriores e façam previsões ou tomem decisões para dados futuros sem que os algoritmos sejam explicitamente programados.

Muitas vezes, usa-se aprendizado de máquina para problemas cujo modelo analítico é bastante complexo mas existem dados disponíveis. Os modelos em aprendizado de máquina aproximam o modelo analítico diretamente com os dados. Como exemplos, pode-se usar aprendizado de máquina em tarefas simples como regressão ou para realizar análises médicas e previsões em dados [26].

Existem várias técnicas na área, como regressão linear, regressão polinomial, árvores de decisão e máquinas de vetores de suporte (do inglês *support vector machines*), e todas têm arquiteturas e parâmetros diferentes. Assim, podem ser aproximadamente modelados uma variedade enorme de sistemas com aprendizado de máquina, lineares ou não-lineares em sua natureza.

Além dos modelos mencionados anteriormente, existem outras técnicas no campo de inteligência artificial baseadas em outros princípios que não necessariamente o de regressão, como algoritmos genéticos. Esta é uma abordagem que busca imitar o processo de seleção natural, em que mutações e cruzamentos aleatórios acontecem dentro de uma população e ao longo do tempo, as soluções mais eficientes para o dado problema vão se tornando dominantes.

2.2.2 Redes neurais artificiais

Dentro da área de aprendizado de máquina, existem as redes neurais artificiais. Estas redes, que podem ter diferentes profundidades, são compostas por camadas de neurônios conectados entre si, de forma similar mas não exatamente igual às redes neurais biológicas.

Entre a entrada e a saída de uma rede neural, existem o que são chamadas de camadas ocultas. Inicialmente, as camadas mais usadas eram as camadas totalmente conectadas, como as duas camadas ocultas da Figura 2.19. Neste tipo de configuração, todos os nós de uma camada tem uma ligação com cada um dos nós da próxima camada. Cada ligação, além de ter um peso associado, é aplicada no que é chamada função de ativação. Assim, cada neurônio tem um limite mínimo para que seja ativado. Para cada um dos neurônios, temos a relação da Equação (2.13):

$$y = \sigma\left(\boldsymbol{w} \cdot \boldsymbol{x} + b\right) = \sigma\left(\sum_{j=0}^{m} w_j x_j + b\right), \tag{2.13}$$

em que $\boldsymbol{x}$ é a entrada do neurônio, y é a saída do neurônio, $\sigma(.)$ é a função de ativação do axônio de saída, $\boldsymbol{w}$ e b são os pesos e o viés e m é o número de entradas do neurônio. Os parâmetros $\boldsymbol{w}$ e b são ajustados durante o treinamento da rede neural.

Várias funções de ativação podem ser escolhidas, e podem ser de natureza não-linear. Uma função bastante popular nos anos iniciais de pesquisa é a sigmoide, dada pela Equação (2.14):

$$\sigma\left(\boldsymbol{w} \cdot \boldsymbol{x} + b\right) = \frac{1}{1 + e^{-(\boldsymbol{w} \cdot \boldsymbol{x} + b)}}. \tag{2.14}$$

Nos anos mais recentes, é comum o uso de variações da função ReLU [27], ou *rectified linear unit*, descrita pela Equação (2.15):

$$\sigma\left(\boldsymbol{w} \cdot \boldsymbol{x} + b\right) = \max(0, \boldsymbol{w} \cdot \boldsymbol{x} + b). \tag{2.15}$$

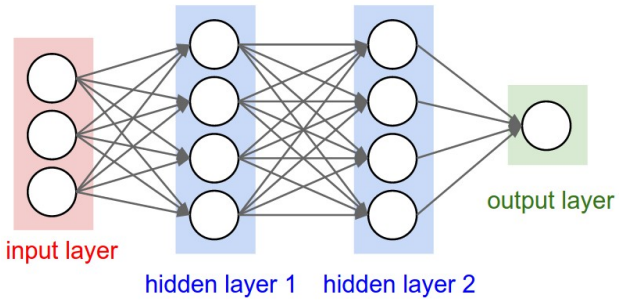

Figura 2.19: Exemplo de rede neural simples, com 2 camadas totalmente conectadas. Fonte: `http://cs231n.github.io/convolutional-networks/`

2.2.3 Retropropagação em redes neurais artificiais

Inicialmente, o treinamento de redes neurais com mais de uma camada oculta se mostrou um desafio [28], por conta da dificuldade de se calcular quanto deveria ser o ajuste de cada parâmetro

em função do erro na saída. Este problema foi resolvido com o desenvolvimento do algoritmo de *backpropagation*, ou retropropagação [29]. Este algoritmo possibilita que, dado o valor do erro na saída, seja determinado o ajuste necessário nos pesos individuais dos neurônios em todas as camadas anteriores para que este erro seja reduzido. Isto possibilitou o treinamento de redes neurais com várias camadas ocultas em tempo computacional factível.

O algoritmo de retropropagação faz parte de um conjunto de técnicas chamadas de diferenciação automática. Neste tipo de algoritmo, as derivadas parciais de todas as operações são calculadas durante as operações com o uso da regra da cadeia em sequência. Com isto, pode-se quantificar o quanto cada neurônio influencia no erro final e ajustar o peso adequadamente. Assim, o algoritmo impõe uma restrição na função de ativação: esta deve ser diferenciável. A ideia geral é que a alteração de cada peso seja calculada conforme a Equação (2.16):

$$\Delta w_{ij} = -\eta \frac{E}{\delta w_{ij}}, \tag{2.16}$$

em que w_{ij} é o peso entre dois neurônios i e j na rede, η é a taxa de aprendizado, que é sempre positiva, e E é o erro da saída. Este algoritmo é implementado no componente chamado de otimizador durante o treinamento das redes neurais.

2.2.4 Componentes de redes neurais artificiais

Uma rede neural pode ser decomposta em três componentes principais: a arquitetura da rede, a função de perda e o otimizador. Criar uma arquitetura de rede neural que dê bons resultados é uma tarefa bastante complexa, portanto é comum que em pesquisas escolha-se uma arquitetura que é comprovadamente boa como ponto de partida. O processo de treinamento de uma rede neural é, em termos rasos, um processo de tentativa e erro, em que a rede calcula a saída dada uma entrada, a função de perda determina quão longe do esperado está essa saída e o otimizador determina o melhor ajuste de parâmetros para que a saída se aproxime do resultado esperado.

2.2.4.1 Função de perda

A função de perda é responsável por determinar quão longe do esperado está a saída de rede. Uma das funções de custo mais utilizadas para tarefas de classificação é a entropia cruzada, que pode ser definida pela Equação (2.17):

$$L = -\sum_{i=0}^{n} y_i \log \hat{y}_i, \tag{2.17}$$

em que n é o número de amostras, y é a verdade absoluta, $\hat{y}$ é a estimativa da rede e L é a perda, ou *loss*. Em tarefas de regressão, geralmente usa-se o erro médio quadrático, dado pela Equação (2.18):

$$L = \frac{1}{n} \sum_{i=0}^{n} (y_i - \hat{y}_i)^2. \tag{2.18}$$

2.2.4.2 Otimizador

O otimizador é responsável por determinar qual o melhor ajuste dos parâmetros da rede para que a função de perda seja minimizada. Quanto aos otimizadores, não existe um dominante, mas o SGD, ou *stochastic gradient descent*, é um dos mais consolidados. O SGD funciona conforme a Equação (2.19):

$$w := w - \eta \nabla Q(w) = w - \eta \sum_{i=1}^{n} \nabla Q(w_i)/n, \tag{2.19}$$

em que o parâmetro w representa o conjunto de pesos das camadas da rede, e Q é a função de perda a ser minimizada, como a entropia cruzada ou o erro médio quadrático descritos anteriormente.

O parâmetro η é a taxa de aprendizado, e define a magnitude das correções feitas pelo SGD. Além disso, η é considerado um hiperparâmetro, que são parâmetros ajustados antes de se começar o processo de treinamento, ao invés de serem aprendidos pela rede. O somatório é utilizado quando o treinamento é feito em lotes, ou *batches*, em que várias entradas são fornecidas de uma vez só, sendo n o tamanho do lote. Na Figura 2.20, temos um exemplo em 1 dimensão do SGD.

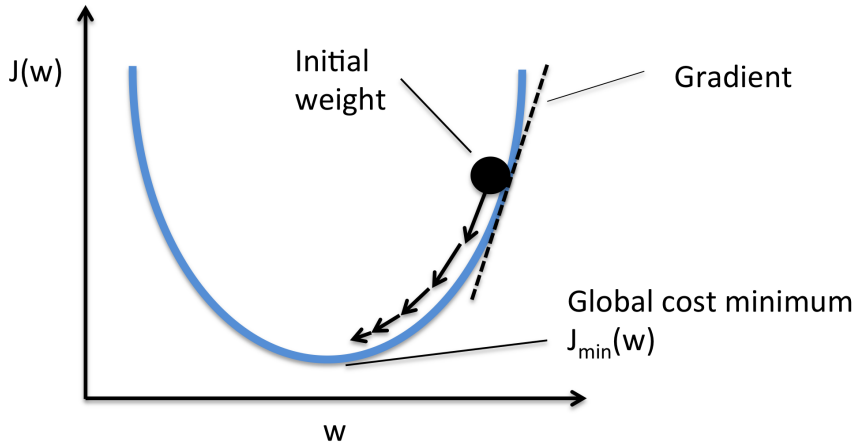

Figura 2.20: SGD em 1 dimensão. Fonte: `http://rasbt.github.io/mlxtend/user_guide/general_concepts/gradient-optimization/`

Outros otimizadores modernos, considerados extensões do SGD, são o Adam e o RMSProp. Estes otimizadores, nas condições corretas, ajudam o algoritmo a convergir mais rapidamente e de maneira menos volátil usando conceitos como decaimento da taxa de aprendizado e momento. Isto é bastante útil, por exemplo, em treinamento por transferência de aprendizado.

2.2.5 Arquiteturas de redes neurais artificiais

Dentro da categoria de redes neurais existem vários tipos específicos de redes que se se destacam em alguns domínios. Dentre as mais comuns, temos:

- Redes neurais profundas, do inglês *deep neural networks*;
- Redes neurais convolucionais, do inglês *convolutional neural networks*;

- Redes neurais recorrentes, do inglês *recurrent neural networks*.

As três variantes não são excludentes entre si. De fato, redes neurais convolucionais profundas são o tipo mais comum em visão computacional.

2.2.5.1 Redes neurais profundas

As redes neurais eram constituídas de poucas camadas no seu surgimento, muito por consequência das limitações de poder computacional da época. Em 2012, houve um grande ressurgimento no interesse por redes neurais, principalmente por redes neurais profundas, por conta do desenvolvimento de hardware com grande capacidade de paralelismo, como as placas de vídeo e os processadores multinúcleo. Ao campo de estudo das redes neurais profundas dá-se o nome de aprendizado profundo.

Deep learning, ou aprendizado profundo, é uma subárea do aprendizado de máquina que busca treinar redes que aprendam por si só a extrair características, ou *features*, dos dados, em contraste com técnicas de aprendizado de máquina que geralmente usam extratores de características manualmente construídos. No aprendizado profundo, empregam-se redes neurais profundas, que possuem múltiplas camadas entre a entrada e a saída da rede. Em visão computacional, é comum que estas camadas sejam compostas em grande parte de convoluções e subamostragens. Um exemplo de rede profunda composta apenas por camadas totalmente conectadas pode ser vista na Figura 2.21. O ponto a partir do qual uma rede é considerada profunda é nebuloso, sendo ainda um ponto de debate na área.

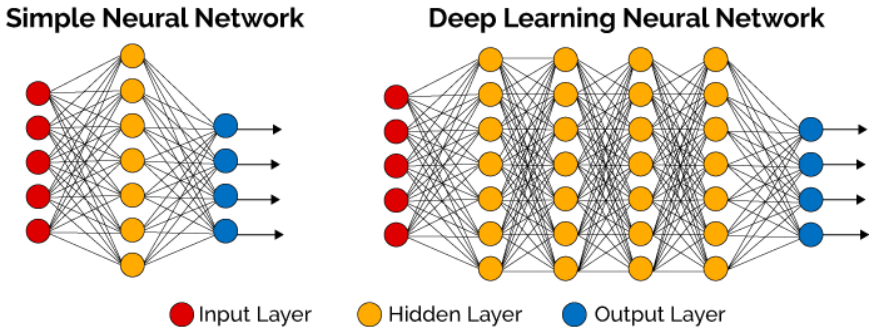

Figura 2.21: Exemplo de rede neural simples e profunda. Fonte: `https://becominghuman.ai/deep-learning-made-easy-with-deep-cognition-403fbe445351`

Redes neurais profundas tem um problema com relação a redes neurais tradicionais: é necessária uma grande quantidade de dados no treinamento. Como são muitas camadas, também são muitos parâmetros a serem otimizados durante o treinamento, e se for fornecida uma quantidade de dados muito pequena, as redes não conseguem generalizar bem para o conjunto de testes.

Este tipo de rede, principalmente empregadas com camadas convolucionais, são muito usadas em análise de imagens. Os exemplos vão desde redes neurais que conseguem indicar se numa imagem obtida por raio-X existe alguma doença [30] até redes que conseguem detectar e indicar a posição de objetos em uma imagem [31]. Outros problemas cuja solução aproximada pode ser

obtida por redes neurais profundas incluem o reconhecimento de voz [32].

2.2.5.2 Redes neurais convolucionais

Redes neurais convolucionais são redes que contém camadas convolucionais, que são camadas que aplicam filtros convolucionais com núcleo de tamanho pré-definido em uma imagem. O tamanho da imagem de entrada e de saída também são previamente definidos. Um exemplo visual de convolução pode ser visto na Figura 2.22. A ideia é que o núcleo deslize sobre a imagem, realizando uma multiplicação do núcleo com a imagem, elemento a elemento, e somando os resultados. A soma então é colocada na célula correspondente de uma nova imagem de tamanho menor.

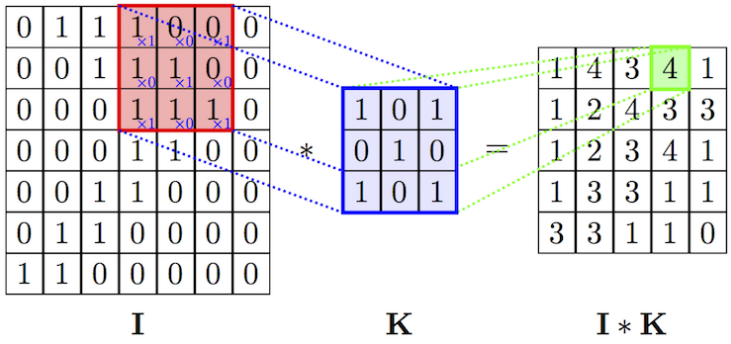

Figura 2.22: Operação de convolução. Fonte: `http://www.deeplearningessentials.science/convolutionalNetwork/`

As redes convolucionais foram inspiradas em processos biológicos. Neurônios biológicos responsáveis pelo processamento de imagens respondem a estímulos em regiões restritas do campo visual, e as regiões entre diferentes neurônios são sobressalentes. Com base nisto, as operações de convolução aplicam filtros que conseguem determinar se na pequena região de análise do núcleo há alguma informação de interesse.

Os parâmetros dos filtros são aprendidos durante o treinamento. O que é fixado é o tamanho do núcleo e as dimensões de entrada e saída da camada. Com isto, não há necessidade de filtros previamente elaborados por um ser humano, que torna a visão computacional um campo com bastante espaço para aplicação de redes neurais convolucionais.

Um exemplo de rede neural profunda convolucional é a AlexNet, uma das primeiras redes a chamar a atenção de pesquisadores por ter ganhado com bastante distância do segundo lugar o desafio ILSVRC, ou *ImageNet Large Scale Visual Recognition Challenge*, em 2012 [33]. Sua arquitetura pode ser vista na Figura 2.23.

2.2.5.3 Redes neurais recorrentes

Redes neurais recorrentes possuem um estado interno, ou memória, que são usados no processamento junto às entradas. Isto faz com que este tipo de rede seja útil em tarefas em que existam relações sequenciais entre as entradas, como por exemplo realizar o reconhecimento de

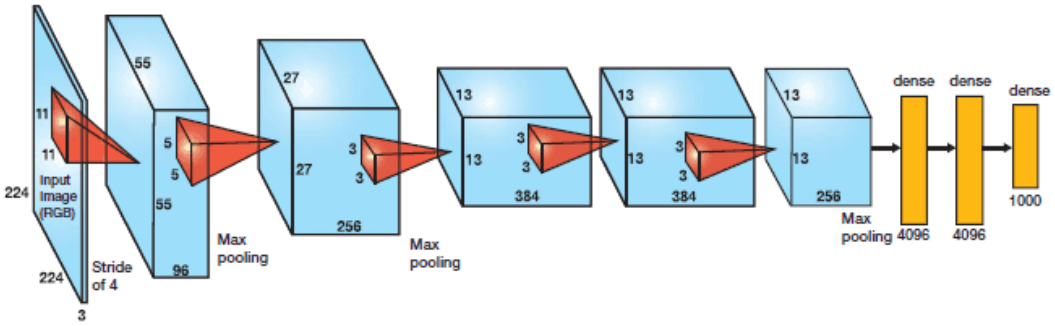

Figura 2.23: Arquitetura da rede AlexNet. Fonte: `https://cv-tricks.com/cnn/understand-resnet-alexnet-vgg-inception/`

escrita e prever a próxima palavra, chamado de processamento de linguagem natural [34], ou o reconhecimento de voz [35].

Uma representação simplificada de uma rede neural recorrente pode ser vista na Figura 2.24, à esquerda. O elemento é tal qual uma rede comum, em que no tempo t um elemento de entrada x_t é fornecido à rede A e uma saída h_t é fornecida pela rede. No entanto, temos um laço que permite que a informação de um instante de tempo seja fornecida à rede no próximo instante.

No entanto, uma rede neural recorrente pode ser vista como uma conexão de várias redes neurais usuais, como na Figura 2.24, à direita. Isto implica que grande parte do estudo de outros tipos de redes neurais pode ser reaproveitado no estudo deste tipo de rede.

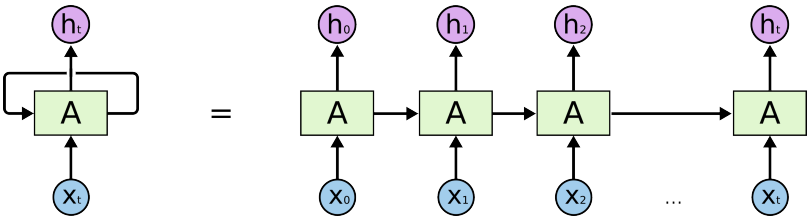

Figura 2.24: Rede neural recorrente e rede neural simples equivalente. Fonte: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Este tipo de rede precisa de um contexto prévio para que possa analisar as entradas. Em muitas situações, o contexto necessário está sequencialmente próximo da entrada sendo analisada. Em outros casos, pode ser que o contexto necessário, ou a sequência de dados necessária, esteja distante na sequência. Neste caso, teremos dependências de longo prazo, e as redes neurais recorrentes mais simples não estão equipadas para lidar com isso.

Houve, então, o surgimento de redes LSTM [36], que contornam o problema de dependências de longo prazo. A principal ideia que soluciona o problema é a inclusão do estado da célula na transmissão de informação, ao passo que redes recorrentes originais só transmitem a saída. Além disso, o estado da célula é uma informação cuidadosamente manipulada pelo resto da célula, de forma que a informação possa trafegar pela rede de forma quase intacta. Na Figuras 2.25 e 2.26, temos uma comparação entre as duas arquiteturas em sua forma expandida.

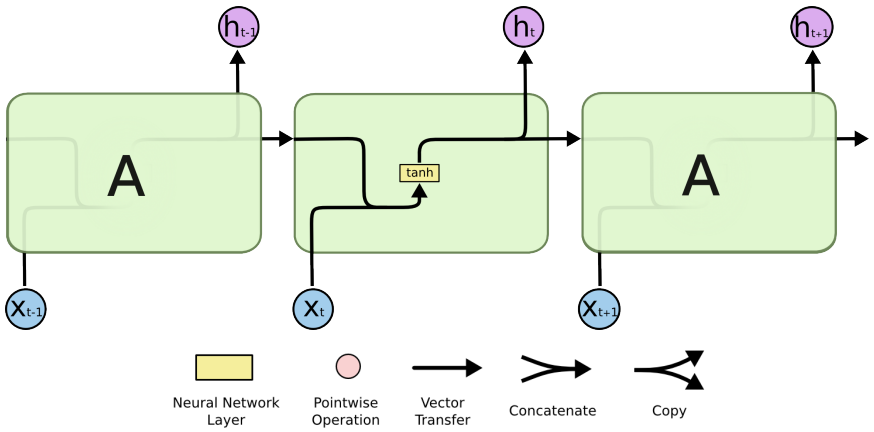

Figura 2.25: RNN simples. Fonte: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

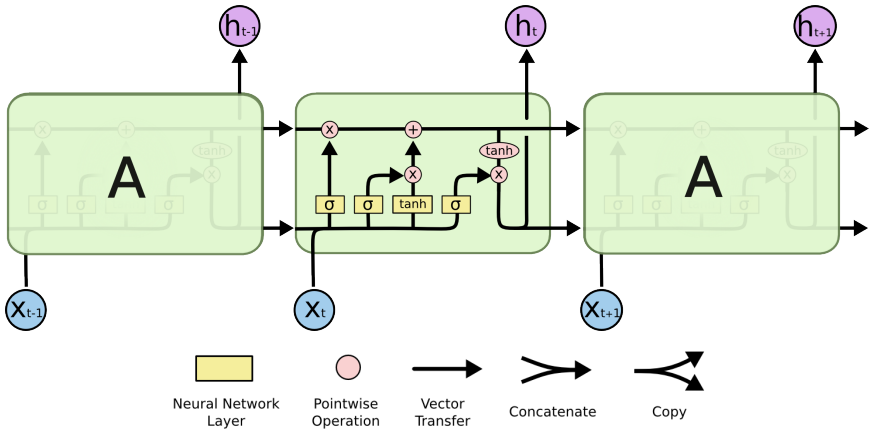

Figura 2.26: Rede LSTM. Fonte: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

Uma célula LSTM guarda uma entrada por um período arbitrário de tempo. Com isto, quando uma rede LSTM é treinada com retropropagação com respeito ao tempo, o gradiente não se anula. Em redes recorrentes comuns, o gradiente pode se anular caso existam espaçamentos muito grandes na sequência entre eventos importantes. Assim, o treinamento de redes LSTM se dá como para os outros tipos de rede, podendo usar os mesmos otimizadores e funções de perda.

Veja que enquanto uma rede recorrente simples tem apenas uma camada na célula, a rede LSTM tem 4 camadas na célula. Temos a adição de 3 camadas, que atuam como portas que controlam a passagem de informação. As portas são compostas por uma função de ativação e uma operação de multiplicação em seguida. Durante o treinamento, os pesos em cada porta são ajustados junto à operação de ativação usual. Em resumo, temos:

- Porta de entrada: controla a medida na qual uma nova entrada entra no valor da célula;
- Porta de esquecimento: controla a medida na qual um valor permanece no valor da célula;
- Porta de saída: controla a medida na qual o valor da célula é usado na saída.

Existem variantes das redes LSTM. Podemos ter uma conexão do estado da célula nas portas, de forma que as decisões sejam tomadas com influência do estado, e podemos também ter um acoplamento da porta de entrada com a porta de esquecimento, de forma que as duas operações sempre ocorram juntas.

2.2.6 Validação cruzada

Para se fazer comparações entre arquiteturas de redes neurais, pode ser usada uma técnica chamada de validação cruzada, ou *cross-validation*, com *k-folds*. Este método consiste em separar o *dataset* em k *folds* mutuamente exclusivos de amostras e treinar k redes usando, para cada uma, $k-1$ *folds* como o conjunto de testes e 1 *fold* como o conjunto de testes. O *fold* escolhido como o de testes varia para cada rede. Na Figura 2.27 temos uma representação gráfica do processo.

Figura 2.27: Representação da validação cruzada com 10 iterações. Fonte: `http://karlrosaen.com/ml/learning-log/2016-06-20/`

Ao fim dos treinamentos, pode-se calcular a média e desvio padrão da função de perda e assim estimar quantitativamente qual arquitetura terá melhor performance naquele conjunto de dados.

2.2.7 Transferência de aprendizado

Transferência de aprendizado [37], ou *transfer learning*, é uma técnica que aproveita os pesos de uma rede neural treinada em um certo conjunto de dados para aplicação em outro conjunto de dados que não foi usado para treinar a rede. É uma técnica muito comum quando os dados são os imagens, já que economiza bastante tempo de treinamento. Um exemplo é a utilização de uma rede capaz de classificar carros para a classificação de caminhões. Os pesos da rede são usados como chute inicial e as camadas finais são ajustadas durante o treinamento para que a rede agora classifique caminhões.

Essa técnica funciona porque a maior parte dos pesos nas camadas iniciais de uma rede neural

são utilizados para extrair características de baixo a médio nível dos dados, e só os pesos das camadas finais são utilizados para extrair características de alto nível. Ou seja, quando se usa uma rede que já aprendeu a extrair características específicas de uma imagem, por exemplo, ganha-se tempo com o treinamento se estes pesos forem reutilizados. Quanto mais similares os conjuntos de dados forem, maiores as chances de ter sucesso com a técnica.

Normalmente, as primeiras camadas das redes neurais aprendem a extrair características genéricas, como linhas e cantos em uma imagem. Já as camadas mais profundas aprendem a extrair as características mais específicas dos dados, como formas geométricas ou padrões de cores. A técnica de transferência de aprendizado em si consiste em substituir as últimas camadas da rede pré-treinada com novas camadas e manter fixos os pesos não substituídos. Com isto, a rede é treinada usando o novo conjunto de dados, e um ajuste fino é realizado nas camadas novas. Um exemplo visual do que acontece com a rede pode ser visto na Figura 2.28.

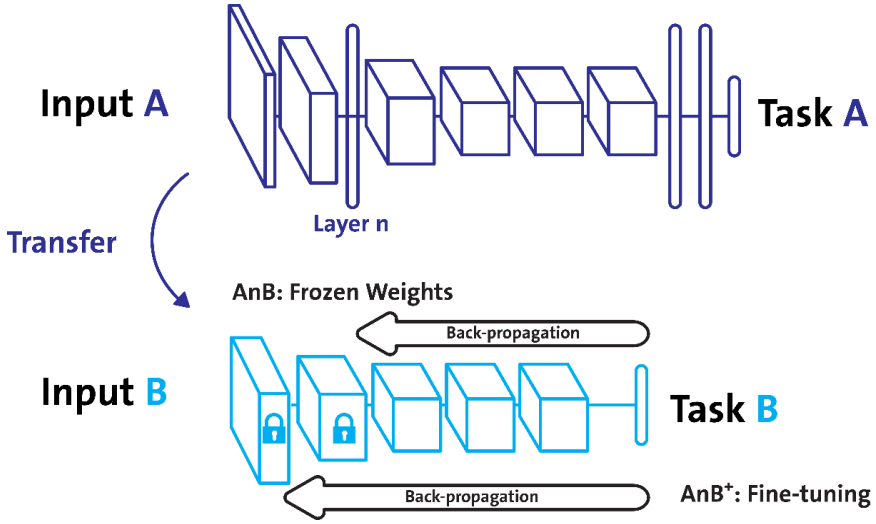

Figura 2.28: Exemplo de transferência de aprendizado. Fonte: `https://www.oreilly.com/library/view/java-deep-learning/9781788997454/de1d99a5-576d-45de-b77f-ee5563550894.xhtml`

Para um bom funcionamento desse método, é comum utilizar uma rede com pesos já treinados em outro conjunto de dados grande, uma vez que quanto maior o conjunto, mais genérica serão as características extraídas nas primeiras camadas. Em classificação de imagens, é comum que se utilize como base redes treinadas no ImageNet [38], que é um dos maiores bancos de dados de imagens existente.

Capítulo 3

Metodologia

Neste trabalho foi realizada uma pesquisa descritiva, em que foram observadas e avaliadas características de diferentes métodos de correção de pilotagem. Foram feitas abordagens quantitativas e qualitativas para a análise dos resultados.

Primeiro, foi realizada uma pesquisa bibliográfica sobre o tema. Inicialmente, os principais livros-texto na área de Robótica e Aprendizado Profundo foram consultados para familiarização com o tema. Foi então feita uma revisão bibliográfica para encontrar os trabalhos mais relevantes e similares ao que foi proposto aqui.

Em seguida, foi planejada a coleta de dados do perfil ideal de pilotagem. A definição do perfil ideal será definido mais adiante. Para treinar uma rede neural profunda, é necessário um grande volume de dados. Foram levantadas então quais variáveis seriam relevantes para o aprendizado da rede. Depois disso, foram escolhidos os percursos de teste desejados para que este fizessem parte da base de dados. Assim, os dados foram coletados e armazenados para uso posterior.

A próxima etapa foi uma pesquisa de campo para treinar e validar quantitativamente as arquiteturas de redes neurais propostas. Foram propostas arquiteturas baseadas em outras já utilizadas em trabalhos similares, em que os dados tem ligação temporal. Em seguida, com a base criada, foram treinados modelos e o RMSE, ou raiz do erro quadrático médio, foi avaliado para cada modelo de forma a dizer se o aprendizado ocorreu com sucesso.

Por fim, foi feita uma pesquisa de campo com entrevistas para a validação qualitativa dos modelos treinados. Foram chamadas pessoas sem experiência com o robô para pilotá-lo, com 2 ou 3 métodos de correção por pessoa. Ao final de cada conjunto de testes, foi feita uma entrevista presencial, de natureza exploratória, para saber qual foi o método preferido e porque. A análise destes dados se deu de forma qualitativa, interpretando o que as respostas dadas significam em termos de validação dos modelos.

3.1 Equipamento utilizado

Para este trabalho, o robô Pioneer 3-AT foi utilizado, visto na Figura 3.1, que é um robô diferencial de 4 rodas não-holonômico. O robô tem um computador embarcado, que contém um sistema operacional instalado. O robô não tem IMU, e os dados da odometria são obtidos a partir dos *encoders* das rodas.

Figura 3.1: Robô Pioneer 3-AT, usado neste trabalho. Fonte: o autor.

O robô funciona com o ROS, ou *robotic operating system*, um *framework* que contém bibliotecas e ferramentas que permitem o desenvolvimento de aplicações para robôs em linguagens de programação usuais. As APIs e drivers que permitem a comunicação entre os algoritmos e os sensores e atuadores do robô estão implementados no ROS.

Foi utilizado Python 3, uma das linguagens suportadas pelo ROS. Além disso, foi utilizado Keras como *framework* de aprendizado de máquina, com TensorFlow de *backend*.

Para o treinamento das redes neurais, foram usados um computador de mesa com GPU dedicada e um *cluster* de servidores da Intel, chamado DevCloud. Para os testes, foi usado um notebook com GPU dedicada conectado ao Pioneer. Para a guiagem do robô, foi usado um controle de videogame conectado por cabo ao notebook.

Na Tabela 3.1 estão listadas as configurações dos computadores utilizados no trabalho. Por brevidade, só foram incluídos os modelos dos equipamentos. Os processadores listados são da Intel e as placas de vídeo listadas são da NVIDIA.

Por fim, os experimentos aconteceram no 2º andar do prédio da Ciência da Computação (CIC) da UnB, cuja planta aproximada pode ser vista na Figura 3.2.

Tabela 3.1: Configurações dos computadores utilizados neste trabalho. O computador de bordo do Pioneer é bastante antigo, e requer uso de um computador auxiliar para tarefas que exijam maior processamento.

Computador	CPU	GPU	OS	Python	Keras	TensorFlow
Desktop	Core i7-8700K	1070 Ti	Ubuntu 18.04	3.6.5	2.1.6	1.12.0
Notebook	Core i5-7200U	940MX	Ubuntu 18.04	3.6.7	2.1.6	1.12.0
DevCloud	Xeon Gold 6128	-	Ubuntu 18.04	3.6.3	2.1.6	1.13.1
Pioneer	Pentium M	-	Ubuntu 12.04	-	-	-

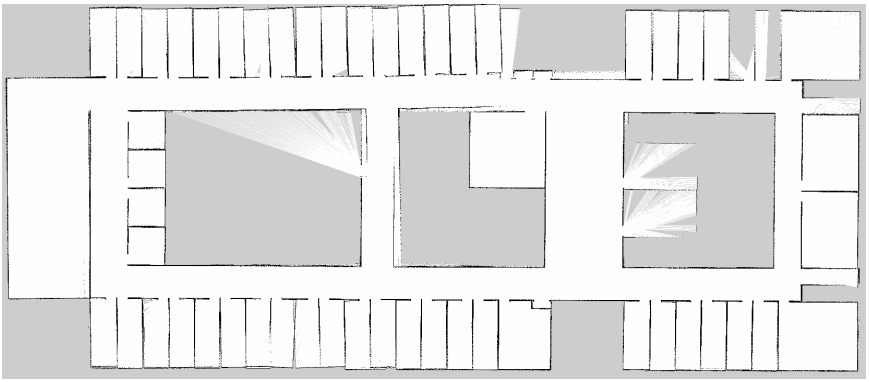

Figura 3.2: Planta alta aproximada do 2º andar do CIC, onde todos os experimentos foram realizados. Fonte: Gabriel Araújo (`https://github.com/Gastd/fcr2018`)

3.2 Criação da base de dados

3.2.1 Definição da base de dados

Para treinar a rede neural de forma que ela aprenda a operar como um bom piloto, foi preciso construir uma base de dados. Foram decididos então qual o comportamento desejado do piloto experiente e os cenários de teste.

A definição de boa pilotagem é bastante subjetiva, mas para este trabalho, o comportamento ideal do piloto seria uma pilotagem suave, sem manobras bruscas, em que não há aceleração em retas e as curvas são feitas com baixa aceleração, independente da velocidade.

Foram escolhidos dois percursos de teste. A Figura 3.3 mostra o percurso realizando nos corredores do prédio do CIC. O segundo percurso pode ser visto na Figura 3.4, que consiste em fazer voltas em forma de 8, em ambos os sentidos de movimento. Foram feitas marcações no chão para o percurso em oito, que podem ser conferidas na Figura 3.5.

A entrada da rede neural, que será proposta mais adiante, é composta do comando do usuário de um vetor de contexto do robô. A saída é a predição de qual deveria ser o próximo comando se um piloto experiente estivesse guiando o robô.

O comando do usuário é dado pela posição da alavanca analógica do controle remoto nos eixos x e y (Figura 3.7), representada pelo vetor $jp = (jp_x, jp_y)$, e a sugestão do próximo comando é

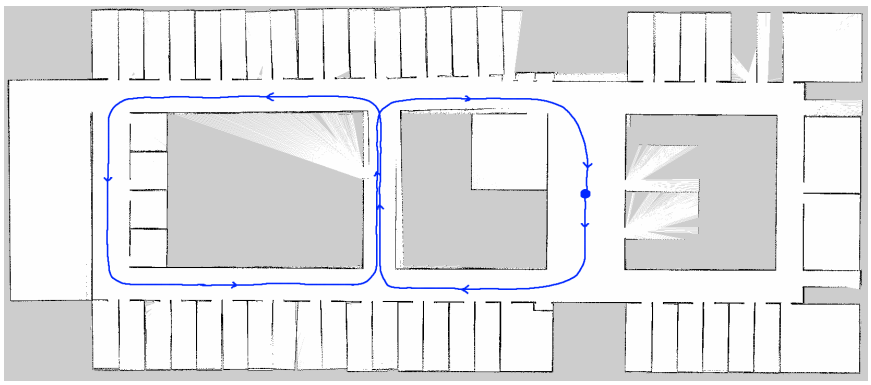

Figura 3.3: Percurso no prédio, desenhado por cima da planta alta. O ponto de início coincide com o de chegada, e está marcado na planta. As setas indicam o sentido de movimento durante os testes. Fonte: adaptado de Gabriel Araújo (`https://github.com/Gastd/fcr2018`)

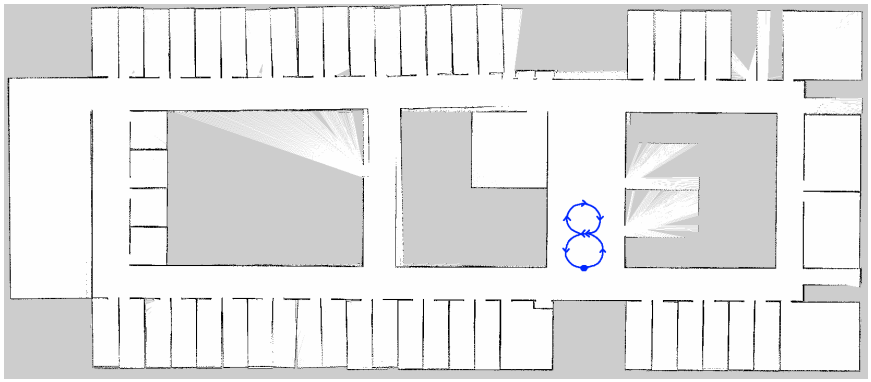

Figura 3.4: Percurso em oito, desenhado por cima da planta alta. Este é um percurso reduzido em um dos corredores do prédio, e novamente temos um ponto de partida e chegada marcado e as setas indicando um dos sentidos de movimento dos testes. Fonte: adaptado de Gabriel Araújo (`https://github.com/Gastd/fcr2018`)

representada por um vetor similar, dado por $jn = (jn_x, jn_y)$. Assim, na construção do conjunto de dados para o treinamento, jn em um instante de tempo k é dado por $jn[k] = jp[k+1]$.

Para auxiliar o aprendizado e capturar melhor o contexto do robô, também foram usadas as velocidades v_x e ω_z do robô, que são as velocidades linear em x e angular em z, respectivamente. Os eixos do robô se dão conforme a Seção 2.1.5, em que o eixo x aponta para a frente do Pioneer e o eixo z aponta para cima.

Além disso, a diferença no tempo das variáveis também foi utilizada, de forma a capturar a dinâmica de primeira ordem do sistema, e neste caso elas estarão prefixadas de Δ, como por exemplo Δv_x, $\Delta \omega_z$ ou Δjp_x. A definição exata dessa diferença pode ser vista na Equação (3.1):

$$\Delta u[k] = u[k] - u[k-1], \tag{3.1}$$

em que u é uma variável qualquer.

Por fim, também foi usado como variável de entrada o tempo decorrido, em nanosegundos, entre as publicações das informações de odometria, chamado de Δt_{ns}. O nó do ROS que publica

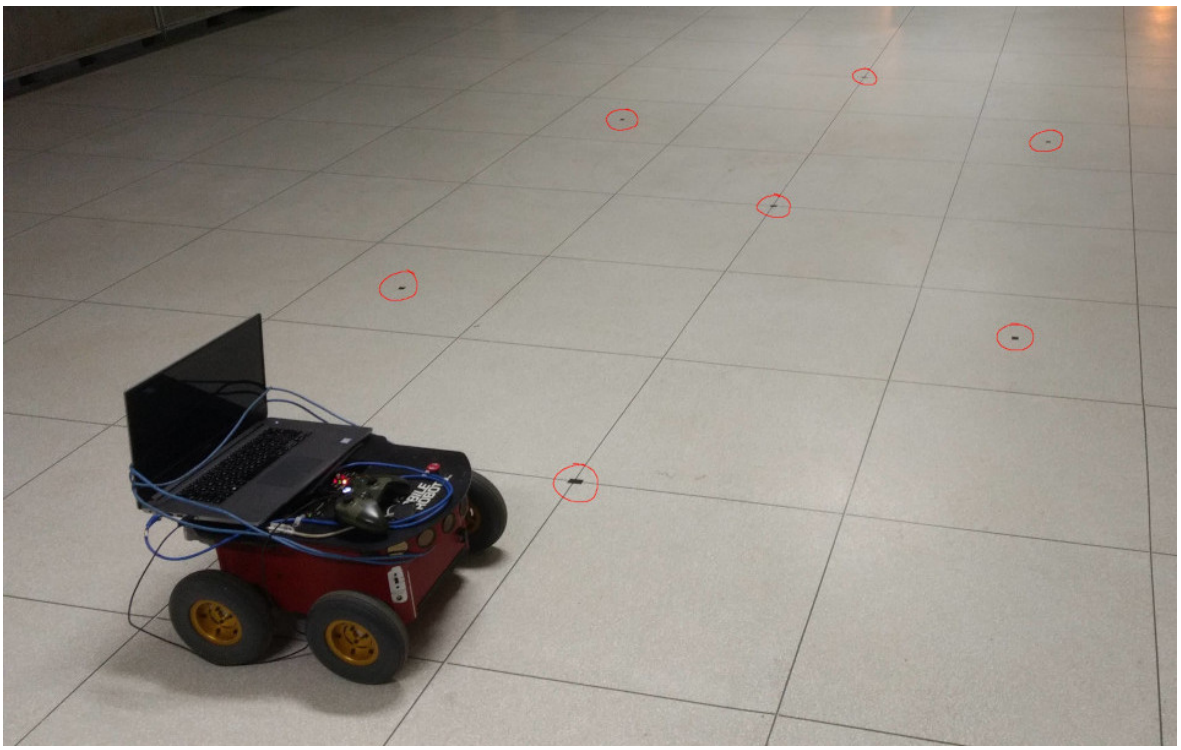

Figura 3.5: Marcações para o percurso em oito. Elas são pequenas, então foram destacadas em vermelho na imagem. Também está visível o Pioneer com notebook e controle conectados, que serão explicados mais adiante. Fonte: o autor.

as informações de odometria tem frequência esperada de 10 Hz, mas o valor real do período pode variar, então essa informação é repassada à rede para auxiliar o aprendizado.

A Tabela 3.2 resume as variáveis de entrada e saída empregadas no aprendizado. De forma a facilitar a notação, foi criado um vetor s para representar as informações de contexto do robô, dado por $s = (\Delta t_{ns}, v_x, \Delta v_x, \omega_z, \Delta \omega_z)$.

Tabela 3.2: Campos de entrada e saída selecionados para o aprendizado.

Tipo	Entrada									Saída	
Vetor	s					jp		Δjp		jn	
Campo	Δt_{ns}	v_x	Δv_x	ω_z	$\Delta \omega_z$	jp_x	jp_y	Δjp_x	Δjp_y	jn_x	jn_y
Unidade	ns	m/s	m/s	rad/s	rad/s	–	–	–	–	–	–

3.2.2 Coleta dos dados

Para realizar a coleta de dados, um notebook foi conectado ao Pioneer por um cabo de rede, tal como na Figura 3.6. Foi conectado também um controle de videogame ao notebook, utilizado para guiar o robô. O ROS permite que numa configuração deste tipo, um computador seja configurado como mestre, que atua como o centralizador de todas as informações e algoritmos. O notebook, nesse caso, foi configurado como mestre.

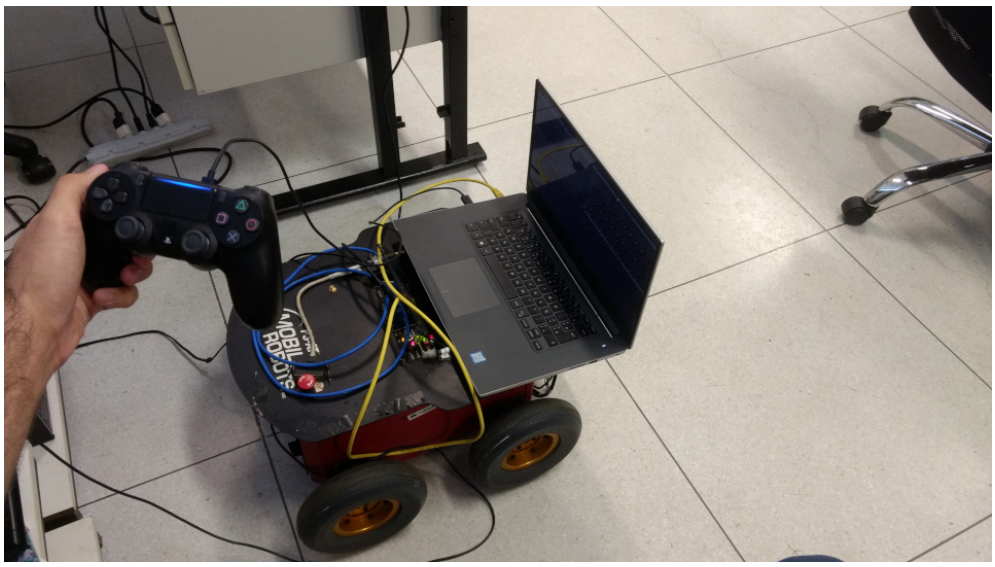

Figura 3.6: Pioneer com notebook e controle conectados. O controle está conectado ao notebook por um cabo USB, e o notebook conectado ao Pioneer por um cabo de rede. Fonte: o autor.

O modo pelo qual o robô é guiado pelo controle é simples: os deslocamentos da alavanca analógica do controle são mapeamento em comandos de velocidade no robô. Os eixos da alavanca podem ser visualizados na Figura 3.7. O mapeamento dos deslocamentos em comandos de velocidade nos eixos do robô se deu como nas Equações (3.2):

$$v_x = 0.5 j_y \tag{3.2}$$

e (3.3):

$$\omega_z = 0.5 j_x, \tag{3.3}$$

em que v_x é a velocidade linear do Pioneer, ω_z é a velocidade angular do Pioneer e j_x e j_y são deslocamentos quaisquer nos eixos x e y da alavanca analógica.

Figura 3.7: Eixos x e y desenhados por cima da alavanca analógica esquerda do controle. Neste caso, o robô é pilotado com os dedos da mão esquerda. Fonte: `https://www.playstation.com/pt-pt/explore/accessories/dualshock-4-wireless-controller/`

O algoritmo de pilotagem sem interferência da rede neural está diagramado na Figura 3.8. Em um nível mais baixo, no que diz respeito aos nós do ROS que implementam este diagrama, temos o diagrama da Figura 3.9. O nó *joy_node* é o que faz a leitura dos sensores do controle e estas informações são publicadas no tópico *joy*. Este tópico é lido por *teleop_node*, que converte os deslocamentos da alavanca analógica em comandos de velocidade e escreve no tópico *cmd_vel*. Por fim, este tópico é lido pelo driver do robô, *p2os_driver*, que efetivamente atua nos motores.

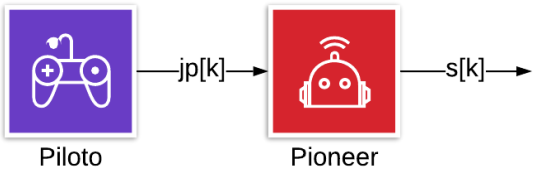

Figura 3.8: Diagrama do sistema sem correção. Os comandos do piloto são diretamente enviados para o Pioneer. É o sistema utilizado na coleta de dados. Fonte: o autor.

Figura 3.9: Nós do ROS no sistema sem correção. Fonte: o autor.

Assim, tendo o robô, o notebook e o controle configurados, o Pioneer foi guiado nos percursos descritos anteriormente, tal como nas fotos da Figura 3.10, e os dados da pilotagem foram armazenados em disco no formato *bag*, que é padrão do ROS. Em seguida, usando a API do *rosbag* em Python, os campos relevantes foram extraídos e armazenados em formato *csv*.

Figura 3.10: Coletando os dados de pilotagem no CIC. Fonte: o autor.

3.3 Arquitetura da rede neural

3.3.1 Proposta de arquitetura

Para que correções sejam feitas na pilotagem de um usuário inexperiente, é necessária uma rede neural que consiga estimar qual deve ser o próximo comando de velocidade com base nos comandos anteriores do usuário e na pilotagem ideal. Este tipo de predição requer contexto, uma vez que é preciso saber o que o usuário está tentando fazer, seja fazer uma curva ou seja seguir uma reta, para que se possa determinar a sugestão ideal. Assim, as redes LSTM apresentam um bom potencial [39].

A arquitetura proposta é composta de duas até quatro camadas LSTM com número variável de neurônios. A quantidade de camadas e o número de neurônios em cada uma foram parametrizados e vários treinamentos foram instanciados. Na Tabela 3.3 podem ser conferidas as propostas de arquitetura. As colunas *Épocas* e *Paciência* serão explicadas na próxima seção, que trata do treinamento das redes.

A fim de simplificar a notação, as arquiteturas da tabela serão referenciadas como

$$N = (neuronios\_por\_camada) - tamanho\_do\_batch - n\_iteracao,$$

com ou sem $n\_iteracao$ a depender do contexto. Um exemplo seria

$$N = (32, 16) - 64$$

significando que é uma arquitetura composta por 2 camadas, uma com 32 neurônios seguida de outra com 16 neurônios, com tamanho de *batch* de 64. O $n\_iteracao$ será útil apenas na seção de resultados, a fim de referenciar modelos específicos obtidos no treinamento.

Tabela 3.3: Arquiteturas propostas, hiperparâmetros de treinamento e sua notação simplificada. Essa notação será referenciada várias vezes no restante deste trabalho.

Épocas	Paciência	Camadas	Neurônios por camada	Tam. *batch*	Notação simplificada
200	50	2	(32, 16)	32	(32, 16) – 32
200	70	2	(32, 16)	64	(32, 16) – 64
200	50	2	(64, 32)	32	(64, 32) – 32
200	70	2	(64, 32)	64	(64, 32) – 64
200	50	3	(96, 64, 32)	32	(96, 64, 32) – 32
200	70	3	(96, 64, 32)	64	(96, 64, 32) – 64
200	50	4	(128, 96, 64, 32)	32	(128, 96, 64, 32) – 32
200	70	4	(128, 96, 64, 32)	64	(128, 96, 64, 32) – 64

Conforme já descrito anteriormente, a ideia desta rede neural é que ela tome como entrada o comando do usuário e consiga predizer qual deveria ser o próximo comando se um piloto experiente estivesse guiando o robô. Temos então o diagrama de entradas e saídas da rede na Figura 3.11. A rede toma como entrada o comando do usuário, $jp[k]$, e o vetor de contexto, $s[k]$, e dá como saída a sugestão de comando, $jn[k]$.

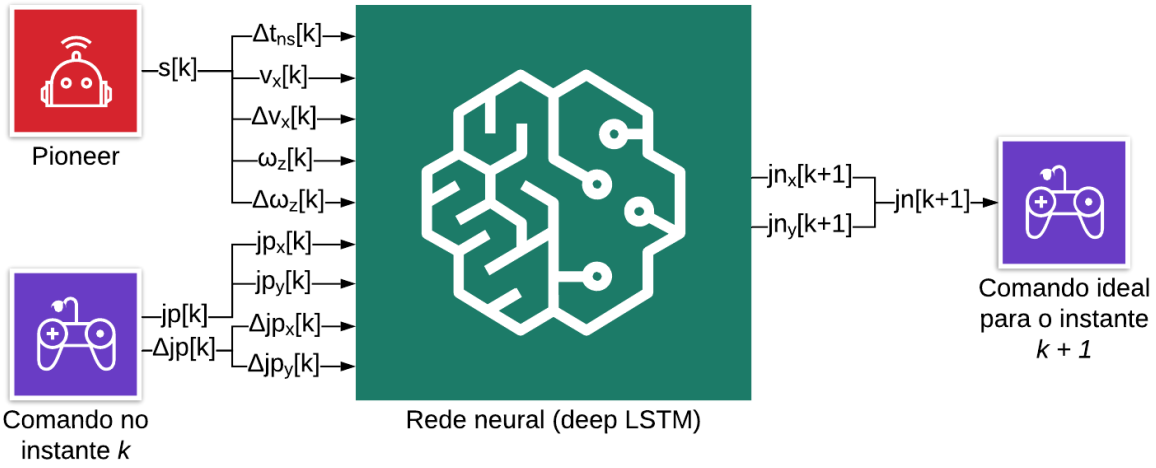

Figura 3.11: Diagrama de entradas e saídas da rede. As entradas e saídas são as mesmas da Tabela 3.2. Fonte: o autor.

3.3.2 Treinamento da rede

Foi usado MSE como função de perda. As entradas e saídas foram normalizadas entre 0 e 1, e cada variável teve seu normalizador específico. Dessa forma, o aprendizado é facilitado e o MSE calculado durante o treinamento já está normalizado. Assim, podemos também determinar o erro RMSE normalizado, ou NRMSE, que pode ser interpretado como uma porcentagem.

Foi usado Adam [40] como otimizador com taxa de aprendizado de $\eta = 0.001$, e não foi usado decaimento da taxa de aprendizado. Os parâmetros β_1 e β_2, específicos do Adam, foram $\beta_1 = 0.9$ e $\beta_2 = 0.999$. Estes são os valores padrão para o Adam no Keras. Ao longo dos experimentos, não houve necessidade de modificá-los.

Para obter maior generalização no conjunto de dados, em cada camada LSTM, com exceção da última, foi usado *dropout*. Neste caso, existem dois tipos de *dropout*: o *dropout* comum, que atua entre as entradas e saídas das camadas LSTM, e o *dropout* recorrente, que atua nas portas internas da LSTM. Ambos foram configurados para 20%.

Além disso, foi utilizado também a parada prematura, ou *early stopping*, para reduzir os tempos de treinamento. Todos os modelos são treinados com no máximo 200 épocas. O parâmetro de paciência foi ajustado empiricamente como $p = 20 \log_2(\textit{tamanho do batch}) - 50$ épocas. Isto resulta em $p = 50$ épocas pra tamanho de *batch* de 32 e $p = 70$ épocas para tamanho de *batch* de 64. O modelo salvo é sempre o modelo com a menor perda no conjunto de validação.

De forma a validar quantitativamente o aprendizado das arquiteturas propostas na Tabela 3.3, foi feito um treinamento usando validação cruzada com *k-folds*. Foram usados 10 *folds* para cada arquitetura em cada um dos dois percursos e foram calculados a média e o desvio padrão do NRMSE para cada arquitetura. Nesta etapa, os modelos usados para realizar as correções ainda não foram treinados; o treinamento com validação cruzada apenas serviu para demonstrar que as arquiteturas conseguem aprender com os dados.

Depois de validar as arquiteturas, os modelos de fato utilizados foram treinados. Para isso, o *dataset* foi separado em conjuntos de treino, validação e teste, com divisão de 70%, 15% e 15%, respectivamente. As gravações dos dados foram analisadas e conjuntos foram separados manualmente. Com isto, 5 iterações de cada uma das arquiteturas propostas foram treinadas e o melhor modelo geral em um percurso foi escolhido para ser usado no algoritmo de correção.

3.4 Validação qualitativa dos modelos

Por fim, depois de ter os modelos treinados em mãos, fez-se uma validação qualitativa desses modelos. Esta validação consistiu em chamar algumas pessoas que nunca haviam pilotado o robô para guiá-lo no prédio e também fazer um oito no chão enquanto um dos modelos corrigia seus comandos em tempo real.

A correção é realizada somente quando a diferença entre o comando do usuário e a sugestão do algoritmo ultrapassa um limiar de seleção T. A métrica de diferença utilizada é a da Equação (3.4):

$$diff(ja, jb) = \frac{||ja - jb||}{2\sqrt{2}} = \frac{\sqrt{(ja_x - jb_x)^2 + (ja_y - jb_y)^2}}{2\sqrt{2}}, \tag{3.4}$$

em que ja e jb são dois comandos quaisquer. Como cada elemento do vetor pode estar no intervalo $[-1, 1]$, temos a divisão por $2\sqrt{2}$ para normalizar a diferença.

A correção do comando do usuário é realizada por dois métodos diferentes. O primeiro é o método de correção por rede, que consiste no chaveamento entre $jp[k]$ e $jn[k]$, conforme o diagrama da Figura 3.12 e a Equação (3.5):

$$jo[k] = \begin{cases} jp[k], & \text{se } diff\,(jp[k], jn[k]) < T; \\ jn[k], & \text{se } diff\,(jp[k], jn[k]) \geq T. \end{cases} \tag{3.5}$$

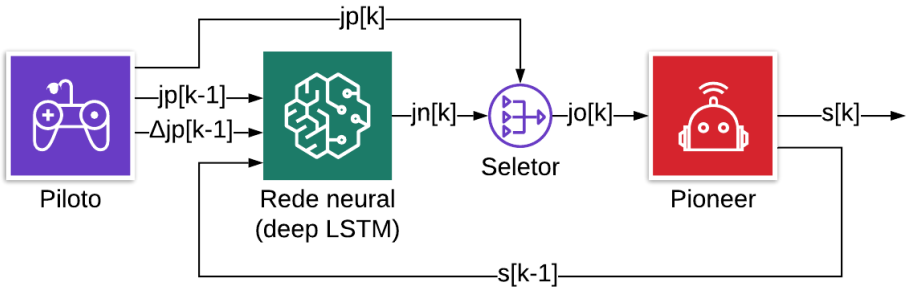

Figura 3.12: Diagrama do sistema com correção por rede. O seletor passa o comando do usuário ao Pioneer quando o comando é próximo da sugestão da rede ou passa a sugestão da rede ao Pioneer caso contrário. Fonte: o autor.

O segundo é o método de correção por média, que consiste no chaveamento entre $jp[k]$ e

$\frac{jp[k]+jn[k]}{2}$, conforme o diagrama da Figura 3.13 e a Equação (3.6):

$$jo[k] = \begin{cases} jp[k], & \text{se } diff\left(jp[k], \dfrac{jp[k]+jn[k]}{2}\right) < T; \\ \dfrac{jp[k]+jn[k]}{2}, & \text{se } diff\left(jp[k], \dfrac{jp[k]+jn[k]}{2}\right) \geq T. \end{cases} \tag{3.6}$$

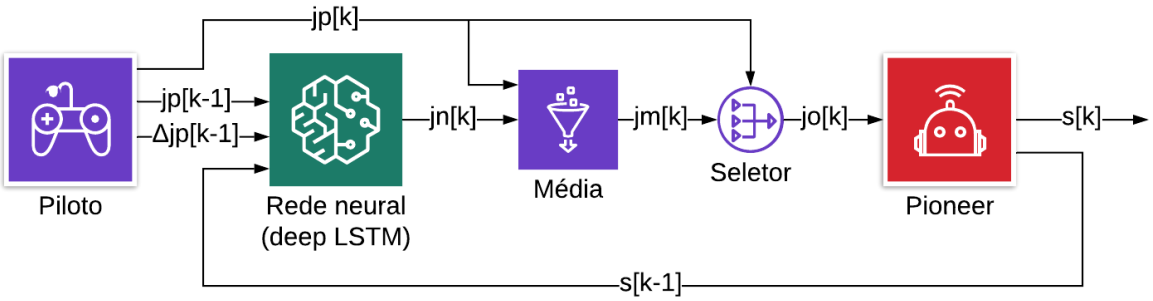

Figura 3.13: Diagrama do sistema com correção por média aritmética. O seletor passa o comando do usuário ao Pioneer quando o comando é próximo da média dos comandos ou passa a média dos comandos ao Pioneer caso contrário. Fonte: o autor.

No que diz respeito ao ROS, temos a implementação dos nós na Figura 3.14. Aqui, a diferença é que o nó *joy_node* agora publica em *joyous* os comandos do piloto e um novo nó *correct* implementa os algoritmos de comparação e correção dos comandos. O nó *correct* lê as informações de contexto do robô do tópico *pose*, que por sua vez recebe mensagens vindas do driver do robô, *p2os_driver*. A taxa de publicação do nó *correct* foi configurada como $20Hz$.

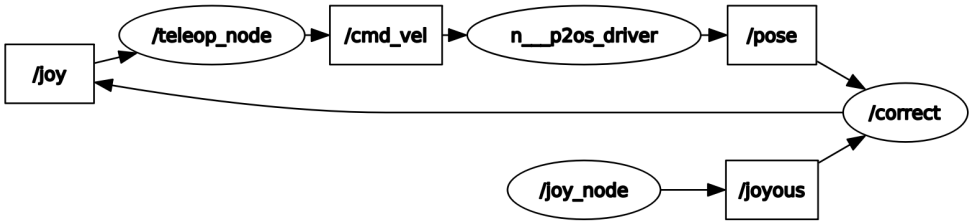

Figura 3.14: Nós do ROS no sistema com correção. Agora, tem-se uma realimentação no sistema, em que as informações de contexto do Pioneer são usadas no controle. Fonte: o autor.

Existem dois parâmetros a serem ajustados ao executar o algoritmo num percurso: o limite do seletor e o método de correção. Para os testes, foram selecionados dois limites, 5% ou 20%. Assim, junto aos dois métodos descritos anteriormente, temos 5 combinações de parâmetros como descritos na Tabela 3.4. A fim de simplificar a notação, a coluna *Notação simplificada* mostra um identificador de cada combinação de parâmetros.

Os testes prosseguiram da seguinte forma: cada pessoa testou os dois percursos, e foram comparadas 2 ou 3 combinações da Tabela 3.4 para cada percurso. Para tentar eliminar o viés na validação, as combinações escolhidas e a ordem dos testes eram aleatórias para cada pessoa. A ordem dos percursos também foi escolhida aleatoriamente. Além disso, não foi informado quais

Tabela 3.4: Combinações de parâmetros para validação e notação simplificada correspondente. Esta notação será referenciada várias vezes no próximo capítulo.

Método de correção	Limiar de seleção (%)	Notação simplificada
Desligado	-	D
Rede	5	R5
Rede	20	R20
Média	5	M5
Média	20	M20

eram as combinações sendo testadas. Isto resultou na Tabela 3.5, que contém os testes realizados. Não foram comparados R5 vs. M20 nem M5 vs. R20 a fim de simplificar a análise dos resultados. A Tabela 3.5 é a mais detalhada; as tabelas que seguirão ao final deste capítulo e no próximo serão agrupamentos derivados desta tabela.

Tabela 3.5: Testes realizados para validação. Esta tabela é a mais detalhada, e diferencia os testes por ordem de realização. Dela, serão obtidas outras duas tabelas importantes: o número de pessoas que testaram cada dupla de métodos e o número de pessoas que testaram cada método individual.

Ordem de teste	Combinação 1	Combinação 2	Combinação 3	Pessoas
Prédio → Oito	R5	M5	-	2
Prédio → Oito	M5	R5	-	1
Prédio → Oito	R20	M20	-	2
Prédio → Oito	M20	R20	-	2
Prédio → Oito	R5	R20	-	1
Prédio → Oito	M5	M20	-	1
Prédio → Oito	M20	M5	D	1
Prédio → Oito	D	M5	R5	1
Oito → Prédio	R5	M5	-	1
Oito → Prédio	M5	R5	-	1
Oito → Prédio	R20	R5	-	1

Ao total, os 11 testes foram realizados com 14 pessoas diferentes que nunca haviam pilotado um robô móvel. Alguns testes foram realizados mais de uma vez, indicado pela coluna *Pessoas* na tabela. Via de regra, os testes com 3 combinações foram exceções.

Para a análise qualitativa dos métodos de correção, um agrupamento importante é o de quantas pessoas testaram cada dupla de métodos, independente da ordem. As preferências dos usuários conduzirão a análise. Este agrupamento está na Tabela 3.6.

Para a análise quantitativa dos métodos de correção, um agrupamento importante é o de quantas pessoas testaram cada método individual. Neste caso, podem ser extraídas métricas específicas de cada método. Este agrupamento está na Tabela 3.7.

Tabela 3.6: Número de pessoas que testaram cada dupla de métodos de correção.

Método	D	R5	R20	M5	M20
D	-	1	0	2	1
R5	1	-	2	5	-
R20	0	2	-	-	4
M5	2	5	-	-	2
M20	1	-	4	2	-

Tabela 3.7: Número de pessoas que testaram cada método de correção.

Método	Número de pessoas
D	2
R5	8
R20	8
M5	6
M20	6

Durante os testes, a cada ciclo de publicação do nó *correct*, uma amostra das informações do sistema foi coletada. Assim, foram extraídas algumas métricas, listadas abaixo, para a avaliação quantitativa do desempenho da rede:

- Tempo médio de publicação (tm_{pub}): permite avaliar o atraso médio de publicação. A equação é dada por:

$$tm_{pub} = \frac{1}{N} \sum_{k=0}^{N} t_{pub}[k], \tag{3.7}$$

  em que t_{pub} é o tempo decorrido entre a k-ésima publicação de *correct* e a publicação anterior.

- Tempo médio de predição do algoritmo (tm_{pred}): permite avaliar o atraso devido à rede neural. A equação é dada por:

$$tm_{pred} = \frac{1}{N} \sum_{k=0}^{N} t_{pred}[k], \tag{3.8}$$

  em que t_{pred} é o tempo necessário para a rede neural realizar a predição no k-ésimo ciclo de *correct*.

- Percentual de atuação do algoritmo (pa): permite determinar o quanto o algoritmo corrigiu o usuário. A equação é dada por:

$$pa = \frac{\sum_{k=0}^{N} a[k]}{\sum_{k=0}^{N} a[k] \vee m[k]}, \tag{3.9}$$

em que as funções $a[k]$ e $m[k]$ são definidas por:

$$a[k] = \begin{cases} 1, & \text{se o algoritmo estava atuando no ciclo } k; \\ 0, & \text{caso contrário.} \end{cases}$$

$$m[k] = \begin{cases} 1, & \text{se o robô estava no movendo no ciclo } k; \\ 0, & \text{caso contrário.} \end{cases}$$

- Diferença média ao longo do teste (*diffm*): permite determinar se a pilotagem foi similar à do algoritmo. A equação é dada por:

$$\textit{diffm} = \frac{1}{N} \sum_{k=0}^{N} \textit{diff}\left(jp[k], js[k]\right), \tag{3.10}$$

  em que $js[k]$ pode ser tanto $jn[k]$ quanto $jm[k]$.

- Aceleração linear média (am_x): permite determinar a suavidade das mudanças de velocidade linear durante o teste. A equação é dada por:

$$am_x = \frac{1}{N} \sum_{k=0}^{N} \left| \frac{\Delta v_x[k]}{\Delta t[k]} \right|. \tag{3.11}$$

- Aceleração angular média (αm_z): permite determinar a suavidade das mudanças de velocidade angular durante o teste. A equação é dada por:

$$\alpha m_z = \frac{1}{N} \sum_{k=0}^{N} \left| \frac{\Delta \omega_z[k]}{\Delta t[k]} \right|. \tag{3.12}$$

Nas equações acima, N é o número total de amostras coletadas durante o teste, que é o número de ciclos executados pelo algoritmo.

De forma a permitir que am_x e αm_z sejam indicativos de suavidade, essas acelerações foram calculadas usando o valor absoluto da diferença de velocidade, $|\Delta v_x|$ e $|\Delta \omega_z|$.

Ao final de cada teste, perguntou-se para cada pessoa qual das combinações ela mais gostou e porque. Nas Figuras 3.15 e 3.16 temos pessoas testando os algoritmos.

Figura 3.15: Pessoa testando os algoritmos de correção no prédio. O robô está sendo conduzido com o controle, conforme descrito anteriormente, Fonte: o autor.

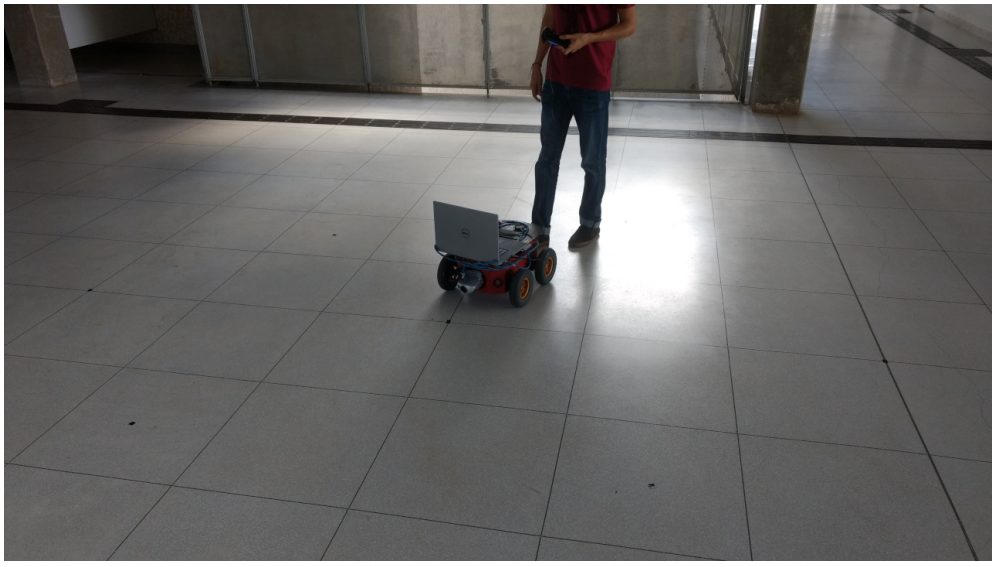

Figura 3.16: Pessoa testando os algoritmos de correção no oito. Fonte: o autor.

Capítulo 4

Resultados

4.1 Criação da base de dados

A base de dados com o perfil de um piloto experiente nos dois percursos propostos foi criada. Os dados capturados estão de acordo com a proposta de ter baixa variação de velocidade. Os dados foram salvos em formato *csv* com os campos descritos conforme a Tabela 3.2.

Nas Figuras 4.1 e 4.2, temos nos gráficos à esquerda os comandos de velocidade salvos para os percursos no prédio e em oito. Além disso, os conjuntos de treino, teste e validação podem ser identificados por cores diferentes, conforme legenda das figuras. Em azul, tem-se os dados de treino, em verde os dados de validação e em laranja os dados de teste. Nos gráficos à esquerda, nos eixos à esquerda, tem-se o deslocamento da alavanca analógica, que não tem unidade, e nos eixos à direita tem-se a velocidade correspondente.

O robô tem uma pequena falha de calibração que faz com que ele tenda a se mover para a esquerda; por conta disto, temos a trajetória na Figura 4.1. No percurso em oito, que é menor e mais uniforme, este problema é menos aparente, como pode ser visto na Figura 4.2.

Para o percurso em oito, a velocidade linear apresentou maior variação que o esperado. Além disso, o conjunto de dados utilizados para o oito origina-se de apenas 10 minutos de pilotagem, que reduz a capacidade de generalização do modelo. Para maior qualidade da base de dados, ambos os aspectos tem espaço para melhoria em trabalhos futuros.

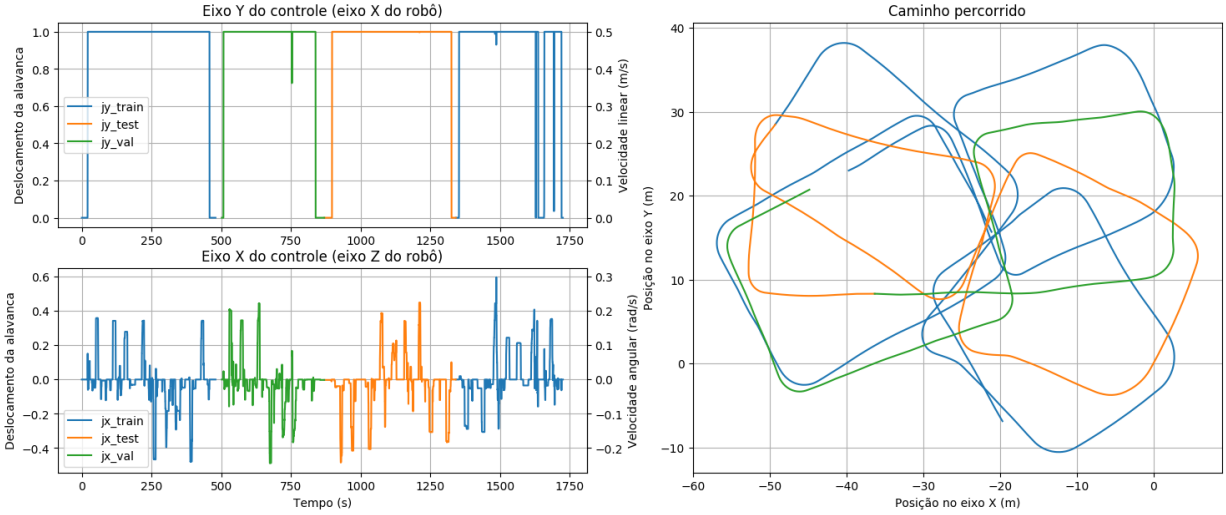

Figura 4.1: Dados coletados durante percurso no prédio. Em azul: treino, em verde: validação, em laranja: teste. No gráfico da trajetória, pode se observar que a posição do robô dada pela odometria não permanece exata por muito tempo. Existem alguns picos de $0.1m/s$, que são pequenas correções de direção devido à tendência do robô de se curvar para a esquerda. Fonte: o autor.

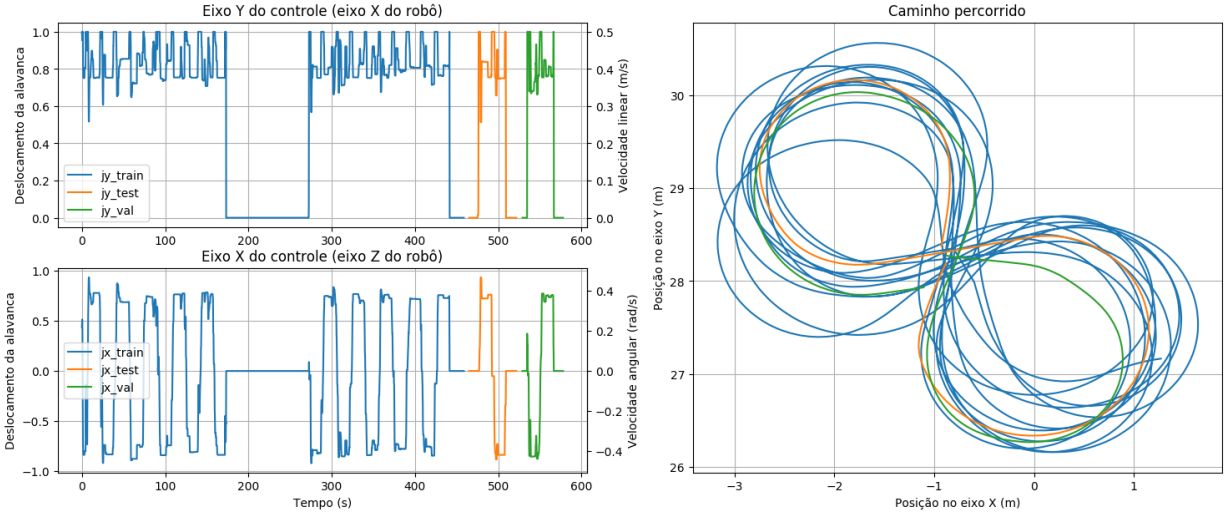

Figura 4.2: Dados coletados durante percurso em oito. Em azul: treino, em verde: validação, em laranja: teste. A trajetória é uniforme e as curvas foram feitas com bastante suavidade, apesar da variação de velocidade linear. Os trechos com velocidade zero foram pausas na captura dos dados. As oscilações na velocidade angular no percurso do oito correspondem à mudanças de direção que acontecem quando se cruza o meio do oito. Fonte: o autor.

4.2 Validação quantitativa dos modelos

Os resultados da validação cruzada do aprendizado é apresentado nas Tabelas 4.1 e 4.2, na coluna *NRMSE (%) - VC*. Nas Figuras 4.3 e 4.4, temos aglomerados das curvas de aprendizado durante a validação cruzada da arquitetura $N = (128, 96, 64, 32) - 64$ para os dois percursos.

Para o treinamento de fato, tem-se agora a coluna *NRMSE (%) - TR* das Tabelas 4.1 e 4.2, que contém os resultados do treinamento com as divisões de treino, teste e validação feitas manualmente. O melhores modelos obtidos estão na Tabela 4.3. Suas curvas de aprendizado estão nas Figuras 4.5 e 4.7, e as predições no *dataset* podem ser vistas nas Figuras 4.6 e 4.8.

Tabela 4.1: Resultados da validação cruzada (VC) e do treinamento (TR) para o prédio. As curvas de aprendizado para o modelo (128, 96, 64, 32) – 64 no prédio estão na Figura 4.3.

Épocas	Paciência	Arquitetura	NRMSE (%) - VC	NRMSE (%) - TR
200	50	(32, 16) – 32	4.18 ± 0.45	3.92 ± 0.15
200	70	(32, 16) – 64	4.16 ± 0.52	3.92 ± 0.22
200	50	(64, 32) – 32	4.33 ± 0.33	3.93 ± 0.14
200	70	(64, 32) – 64	4.26 ± 0.35	4.05 ± 0.11
200	50	(96, 64, 32) – 32	3.73 ± 0.44	3.49 ± 0.11
200	70	(96, 64, 32) – 64	3.68 ± 0.49	3.25 ± 0.08
200	50	(128, 96, 64, 32) – 32	3.56 ± 0.47	3.24 ± 0.15
200	70	(128, 96, 64, 32) – 64	3.45 ± 0.42	3.34 ± 0.14

Tabela 4.2: Resultados da validação cruzada (VC) e do treinamento (TR) para o oito. As curvas de aprendizado para o modelo (128, 96, 64, 32) – 64 no oito estão na Figura 4.4.

Épocas	Paciência	Arquitetura	NRMSE (%) - VC	NRMSE (%) - TR
200	50	(32, 16) – 32	4.71 ± 0.56	5.10 ± 0.23
200	70	(32, 16) – 64	4.59 ± 0.68	5.36 ± 0.22
200	50	(64, 32) – 32	4.76 ± 0.63	5.13 ± 0.12
200	70	(64, 32) – 64	4.77 ± 0.79	5.18 ± 0.04
200	50	(96, 64, 32) – 32	4.36 ± 0.83	4.90 ± 0.26
200	70	(96, 64, 32) – 64	4.46 ± 0.75	4.84 ± 0.22
200	50	(128, 96, 64, 32) – 32	4.41 ± 0.68	5.08 ± 0.18
200	70	(128, 96, 64, 32) – 64	4.36 ± 0.73	5.12 ± 0.14

Tabela 4.3: Melhores modelos obtidos. Os gráficos para o modelo (128, 96, 64, 32) – 32 estão nas Figuras 4.5 e 4.6, e os gráficos para o modelo (96, 64, 32) – 32 estão nas Figuras 4.7 e 4.8.

Percurso	Neurônios por camada	Tamanho do *batch*	Iteração	NRMSE (%)
Prédio	(128, 96, 64, 32)	32	3	3.00
Oito	(96, 64, 32)	64	0	4.52

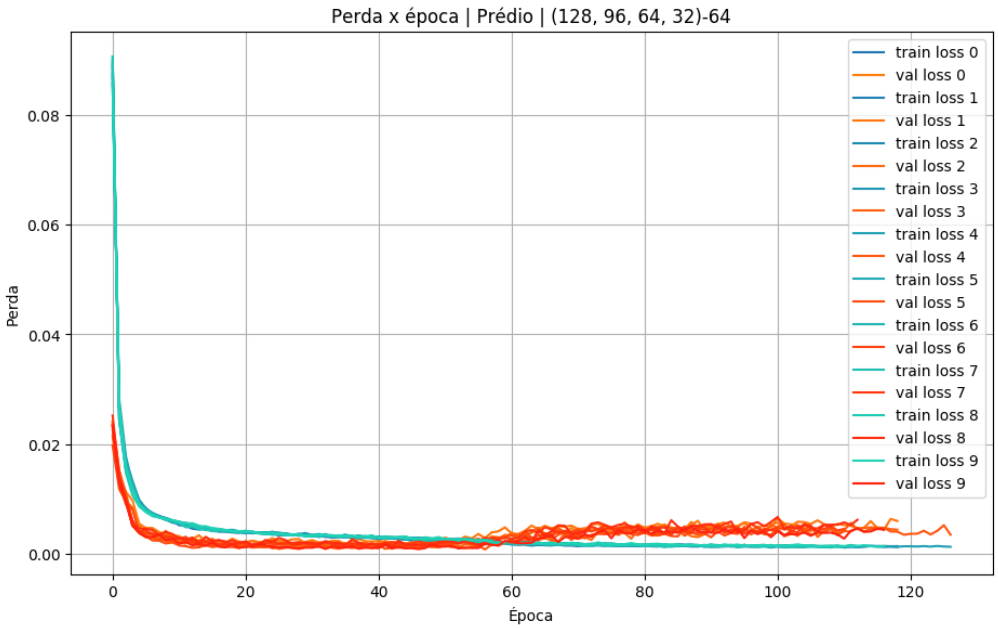

Figura 4.3: Aglomerado de curvas de aprendizado no prédio para a arquitetura $N = (128, 96, 64, 32) - 64$. Fonte: o autor.

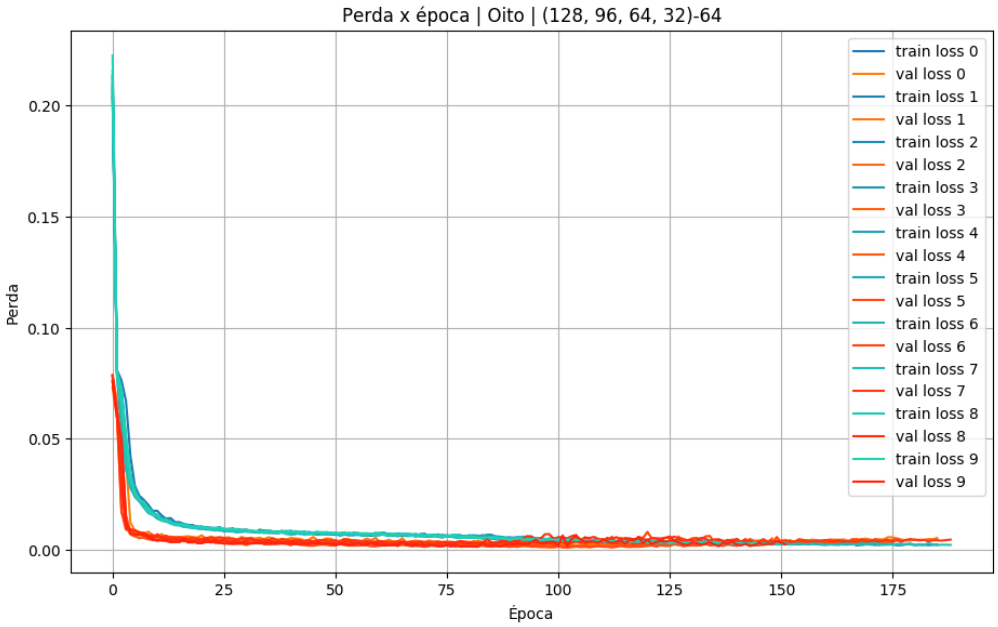

Figura 4.4: Aglomerado de curvas de aprendizado no oito para a arquitetura $N = (128, 96, 64, 32) - 64$. Fonte: o autor.

Quanto à validação cruzada, verifica-se nas Tabelas 4.1 e 4.2 que todas as arquiteturas propostas tem potencial de aprendizado com os dados de treinamento, uma vez que a média do NRMSE ficou abaixo de 5% para todas as arquiteturas. Nas curvas das Figuras 4.3 e 4.4 pode-se observar que não há grande variabilidade entre os *folds* durante a validação cruzada, o que indica que o *dataset* é uniforme e as arquiteturas tem treinamento estável.

Quanto ao treinamento de fato, o potencial de aprendizado se confirma, e os casos $N = (96, 64, 32)$ e $N = (128, 96, 64, 32)$ apresentam os melhores resultados. As curvas de aprendizado nas Figuras 4.5 e 4.7 e as predições em 4.6 e 4.8 indicam convergência no aprendizado.

A discrepância entre os NRMSEs da validação cruzada e os do treinamento possivelmente se deve ao método pelo qual o *dataset* foi separado em treino, validação e teste. Na validação cruzada, o *dataset* foi dividido aleatoriamente em 10 *folds*, enquanto que no treinamento a divisão foi feita manualmente, conforme explicado na seção 3.3.2.

Os NRMSEs são maiores no oito do que no prédio, apesar de os dados do oito serem mais padronizados. Isso possivelmente se deve ao fato de que os dados do prédio, em boa parte do tempo, consistem de seguir retas, que torna o aprendizado mais fácil.

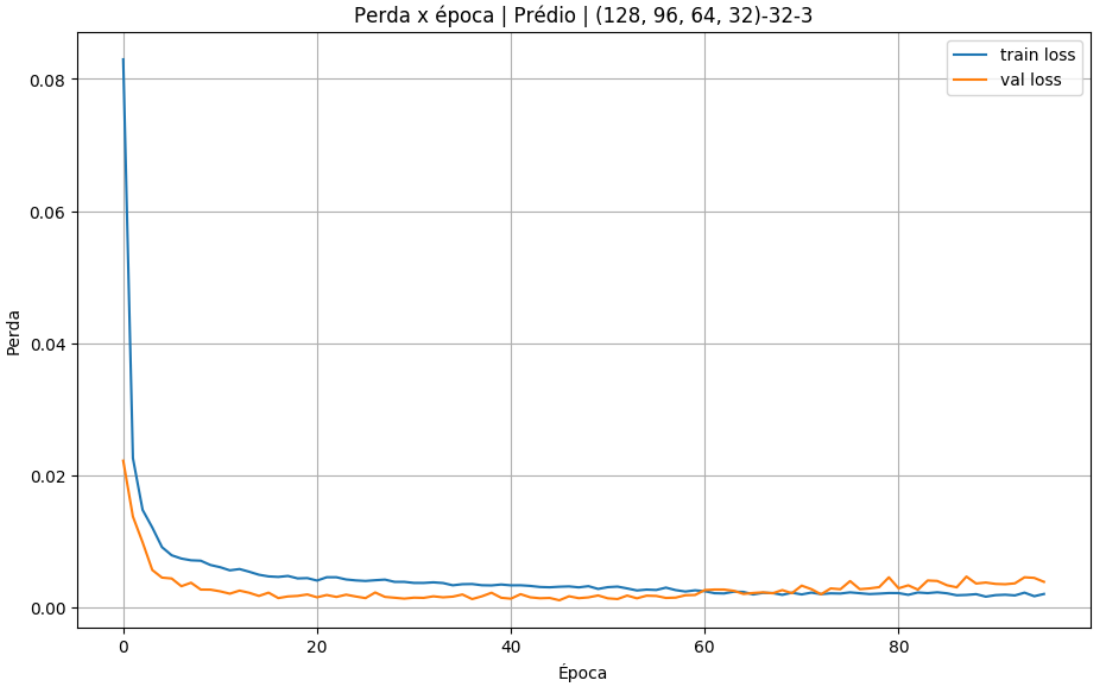

Figura 4.5: Curva de aprendizado no prédio para $N = (128, 96, 64, 32) - 32 - 3$. Fonte: o autor.

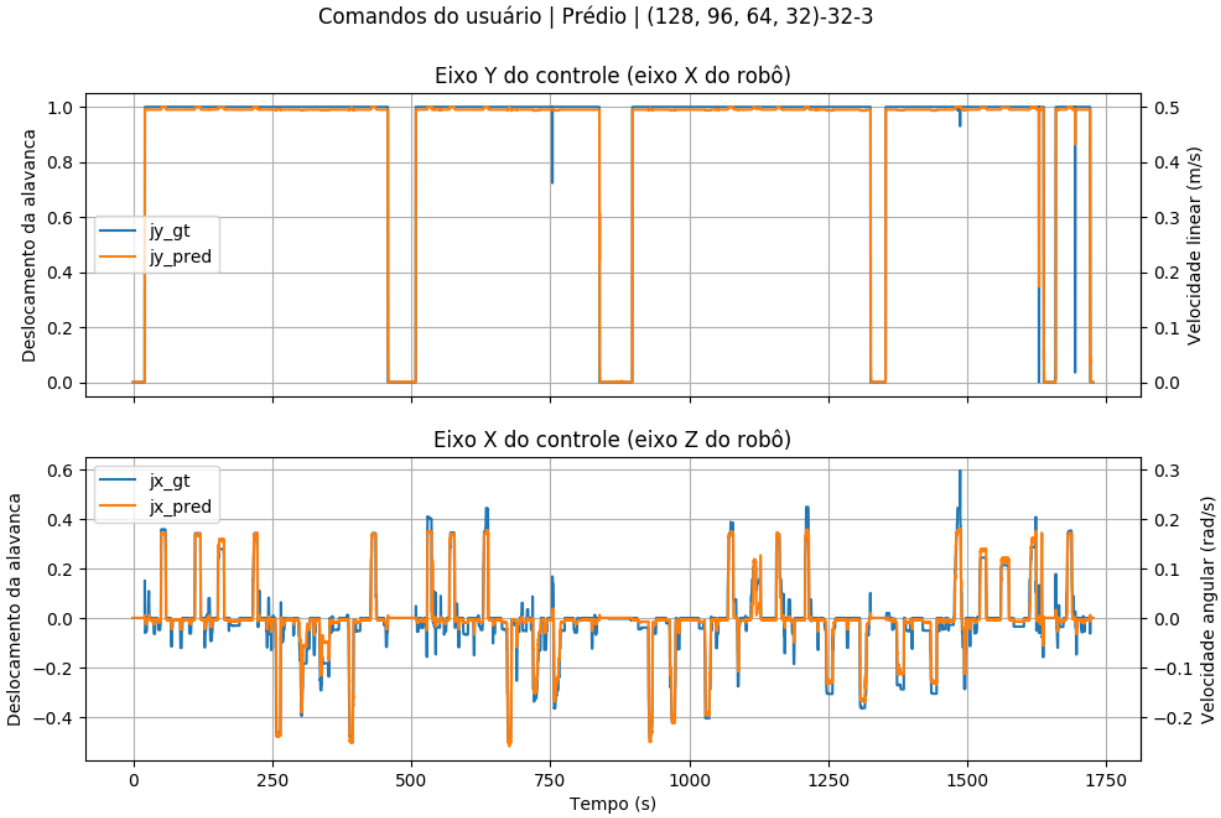

Figura 4.6: Gráfico da predição dos comandos no prédio para $N = (128, 96, 64, 32) - 32 - 3$. A rede não aprendeu a reproduzir os picos de $0.1 m/s$. Fonte: o autor.

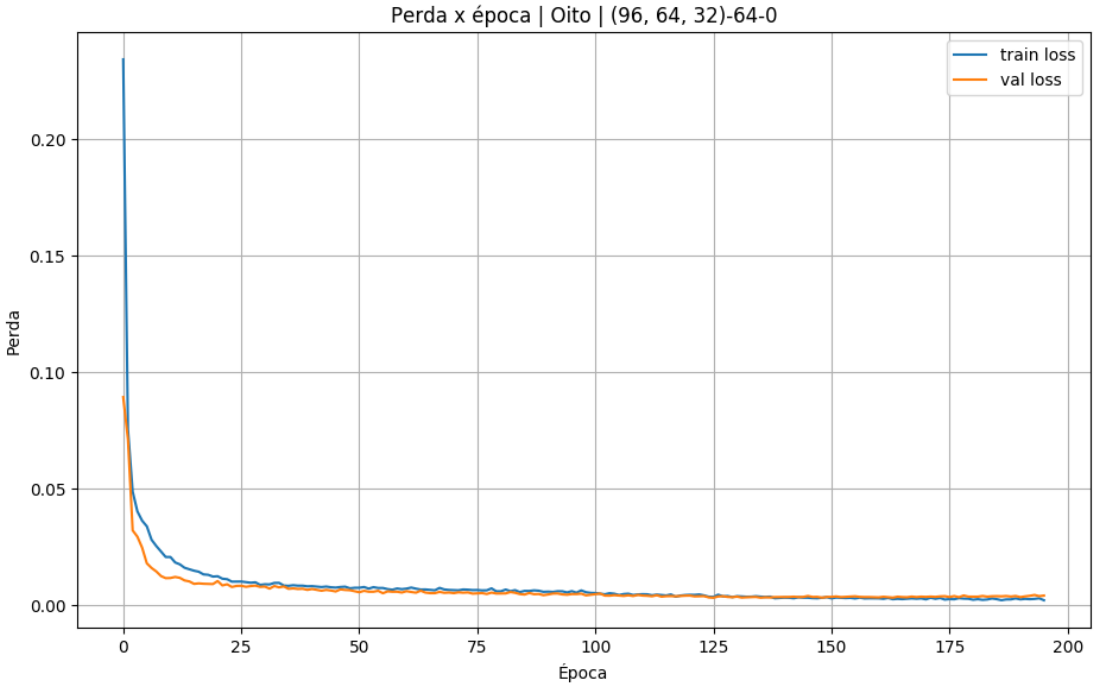

Figura 4.7: Curva de aprendizado no oito para $N = (96, 64, 32) - 64 - 0$. Fonte: o autor.

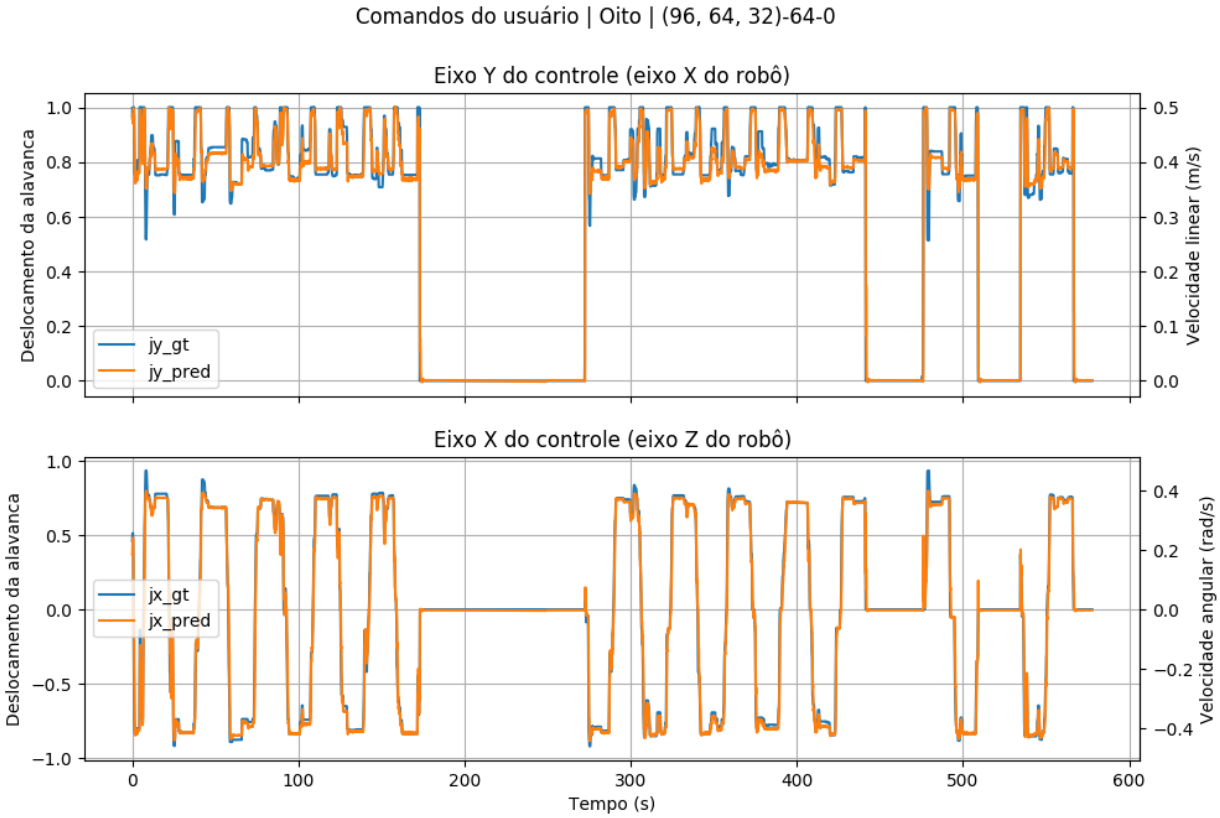

Figura 4.8: Gráfico da predição dos comandos no oito para $N = (96, 64, 32) - 64 - 0$. A rede aprendeu a reproduzir a variação de velocidade linear. Fonte: o autor.

4.3 Validação qualitativa dos modelos

A Tabela 4.4 contém as métricas de desempenho para o nó *correct* calculadas em ambos os percursos. A Tabela 4.5 contém as métricas do algoritmo completo para o percurso no prédio, e a Tabela 4.6 para o percurso em oito.

Tabela 4.4: Resultados de desempenho do nó *correct* durante a validação qualitativa.

		Prédio		Oito	
Método	Pessoas	tm_{pub} (ms)	tm_{pred} (ms)	tm_{pub} (ms)	tm_{pred} (ms)
D	2	50.0 ± 0.0	-	51.4 ± 1.4	-
R5	8	50.0 ± 0.0	10.4 ± 0.3	53.0 ± 1.4	9.2 ± 0.2
M5	8	56.8 ± 18.1	10.6 ± 0.3	55.8 ± 4.9	9.2 ± 0.2
R20	6	50.0 ± 0.0	10.5 ± 0.3	55.3 ± 2.8	9.1 ± 0.2
M20	6	50.1 ± 0.3	10.6 ± 0.2	55.2 ± 3.4	9.2 ± 0.2
R	14	50.0 ± 0.0	10.5 ± 0.3	53.8 ± 2.3	9.2 ± 0.2
M	14	53.9 ± 14.0	10.6 ± 0.2	55.6 ± 4.4	9.2 ± 0.2

Tabela 4.5: Resultados do algoritmo durante a validação qualitativa no prédio.

Método	Pessoas	pa (%)	$diffm$ (%)	am_x (m/s^2)	αm_z (m/s^2)
D	2	-	-	0.0333 ± 0.0041	0.1020 ± 0.0165
R5	8	23.44 ± 7.52	4.58 ± 1.69	0.0248 ± 0.0050	0.1016 ± 0.0283
M5	8	7.44 ± 4.71	1.47 ± 0.53	0.0279 ± 0.0076	0.1210 ± 0.0261
R20	6	6.27 ± 7.37	4.17 ± 3.45	0.0377 ± 0.0272	0.1436 ± 0.0917
M20	6	0.11 ± 0.22	1.83 ± 0.79	0.0407 ± 0.0272	0.1384 ± 0.0574
R	14	16.08 ± 11.30	4.40 ± 2.60	0.0303 ± 0.0193	0.1196 ± 0.0670
M	14	4.30 ± 5.09	1.62 ± 0.68	0.0334 ± 0.0198	0.1284 ± 0.0433

Tabela 4.6: Resultados do algoritmo durante a validação qualitativa no oito.

Método	Pessoas	pa (%)	$diffm$ (%)	am_x (m/s^2)	αm_z (m/s^2)
D	2	-	-	0.1381 ± 0.0524	0.2428 ± 0.0474
R5	8	67.52 ± 10.10	9.61 ± 2.74	0.0928 ± 0.0253	0.2133 ± 0.0480
M5	8	14.34 ± 11.86	3.46 ± 1.19	0.1222 ± 0.0418	0.2534 ± 0.0769
R20	6	12.83 ± 5.30	8.52 ± 1.48	0.1234 ± 0.0243	0.2610 ± 0.0679
M20	6	0.09 ± 0.17	3.20 ± 0.99	0.1131 ± 0.0271	0.2142 ± 0.0618
R	14	47.99 ± 27.61	9.22 ± 2.43	0.1037 ± 0.0290	0.2303 ± 0.0604
M	14	9.25 ± 11.71	3.37 ± 1.13	0.1189 ± 0.0375	0.2394 ± 0.0743

Na Tabela 4.4, podemos observar que a taxa de publicação ficou próxima de $20Hz$, ou com período de $50ms$, conforme o esperado. Além disso, o tempo de predição da rede ficou próximo de $10ms$. Isto nos indica que poderíamos ter aumentado a taxa de publicação sem criar uma fila de

mensagens. De fato, 2 usuários de testes reportaram que sentiram um pequeno atraso na direção do robô independente do método de correção. Esta é uma outra melhoria para um trabalho futuro.

Na Tabela 4.5 podemos observar em *pa* e *diffm* que o método de correção por rede atua mais que a correção por média, que é de se esperar. Ainda, a correção por rede tende a suavizar mais a direção do piloto com limiar de 5%, evidenciado quando am_x e αm_z são comparadas entre as linhas da tabela. Isto nos indica que para limiar de 5%, a correção por rede causou uma suavização da pilotagem maior que a da correção por média. Para 20%, temos que am_x está conforme o esperado, mas αm_z está maior para a rede do que para a média.

Na Tabela 4.6 temos resultados similares, mas para o limiar de 20%, temos suavização maior para a correção por média, que não era o resultado esperado. No entanto, deve-se levar em consideração que o chaveamento entre o comando do piloto e do algoritmo não é uma transição suave, principalmente no limiar de 20% de diferença. Isso pode afetar na pilotagem, principalmente no oito, em que há maior atuação da rede do que no prédio. Além disso, observou-se que no percurso em oito, quando o perfil é muito diferente da rede, o piloto constantemente sai do trajeto. Isso acaba acarretando numa maior aceleração, uma vez que o piloto necessita ajustar sua pilotagem. Isso será ser observado mais a frente, em que serão analisados alguns casos específicos de pilotagem para explorar alguns perfis de piloto diferentes.

A análise da aceleração média sem nenhuma correção se torna complicada devido ao baixo número de pessoas. Com somente 2 pessoas, as variações no perfil de pilotagem se tornam muito mais significativas.

Nas Figuras 4.9, 4.10 e 4.11, temos as preferências dos pilotos em gráficos de setores para algumas duplas de combinações. As duplas escolhidas são as hachuradas na Tabela 4.7. A dupla M5 vs. M20 não foi escolhida porque os usuários atestaram que sentiram baixa diferença entre os dois métodos.

Tabela 4.7: Duplas de métodos escolhidas para análise qualitativa. A tabela é uma matriz simétrica, e somente três duplas foram escolhidas. Os resultados correspondentes em gráficos de pizza estão nas Figuras 4.9, 4.10 e 4.11.

Método	D	R5	R20	M5	M20
D	-	1	0	2	1
R5	1	-	2	5	-
R20	0	2	-	-	4
M5	2	5	-	-	2
M20	1	-	4	2	-

Como resultados gerais das entrevistas, existe uma tendência de preferência aos métodos que interferem menos na direção do piloto, principalmente no percurso em oito, em que o perfil do piloto tem mais impacto no algoritmo de correção.

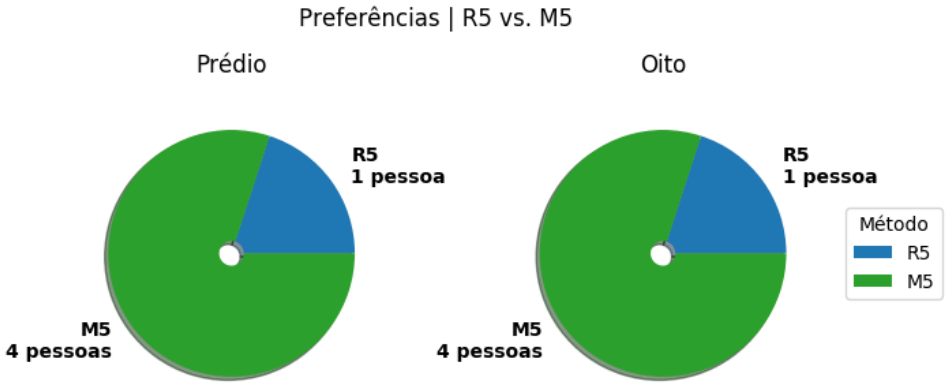

Figura 4.9: Preferências entre R5 e M5. A correção por média foi a preferida nos dois percursos. Fonte: o autor.

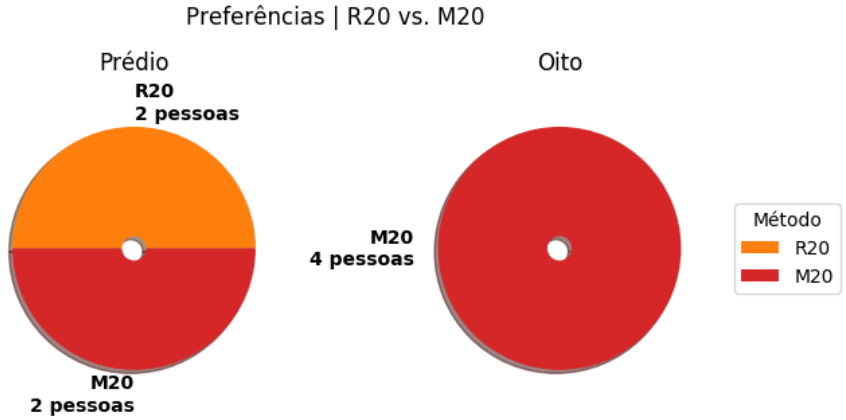

Figura 4.10: Preferências entre R20 e M20. No prédio houve empate, mas no oito a correção por média foi a preferida. Fonte: o autor.

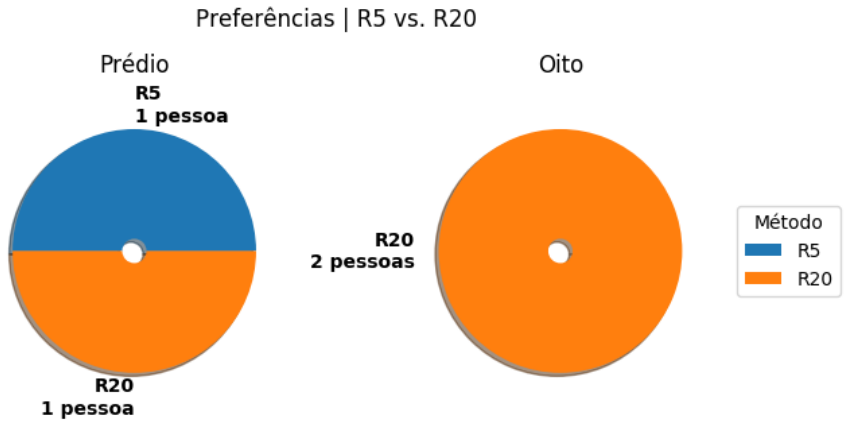

Figura 4.11: Preferências entre R5 e R20. No prédio houve empate, mas no oito o limiar de 20% foi o preferido. Fonte: o autor.

4.4 Análise de casos específicos

Durante o acompanhamento dos testes, observou-se que o perfil de pilotagem dos usuários variou bastante. Por perfil de pilotagem entende-se a maneira como são feitas as curvas, principalmente no percurso em oito, em que o *timing* das curvas é muito importante. A seguir, temos uma análise de alguns casos específicos de pilotagem no percurso em oito.

Os gráficos analisados nas seções subsequentes consistem nos comandos de velocidade à esquerda e da trajetória à direita. Os gráficos estão codificados por cores de forma que a curva de saída do algoritmo à esquerda, que é a curva com cores quentes, se relaciona com o percurso à direita. Além disso, o ponto roxo indica o início do teste, e o ponto azul indica o fim do teste.

4.4.1 Usuário A

No primeiro caso, com o usuário A, temos a realização dos testes indicados na Tabela 4.8, na ordem D → M5 → R5. O usuário A teve dificuldades em seguir o oito sem correção e dos três métodos testados, preferiu dirigir com correção por rede. Nas Figuras 4.12, 4.13 e 4.14 temos os gráficos do caminho percorrido e dos comandos dados pelo usuário para cada teste da Tabela 4.8.

Aqui, os dados seguem o padrão esperado. A pilotagem sem correção resulta em maiores am_z e αm_z, ou menor suavização. Já a correção por média resulta em suavização intermediária e a correção por rede resulta na maior suavização. Pode-se observar nos gráficos de trajetória que as curvas se tornam mais bem definidas conforme a interferência da rede aumenta. Além disso, o algoritmo tenta manter os comandos do usuário A constantes, principalmente os de velocidade linear. Isto indica que a rede conseguiu aprender e cumpriu bem o objetivo de auxiliar na pilotagem.

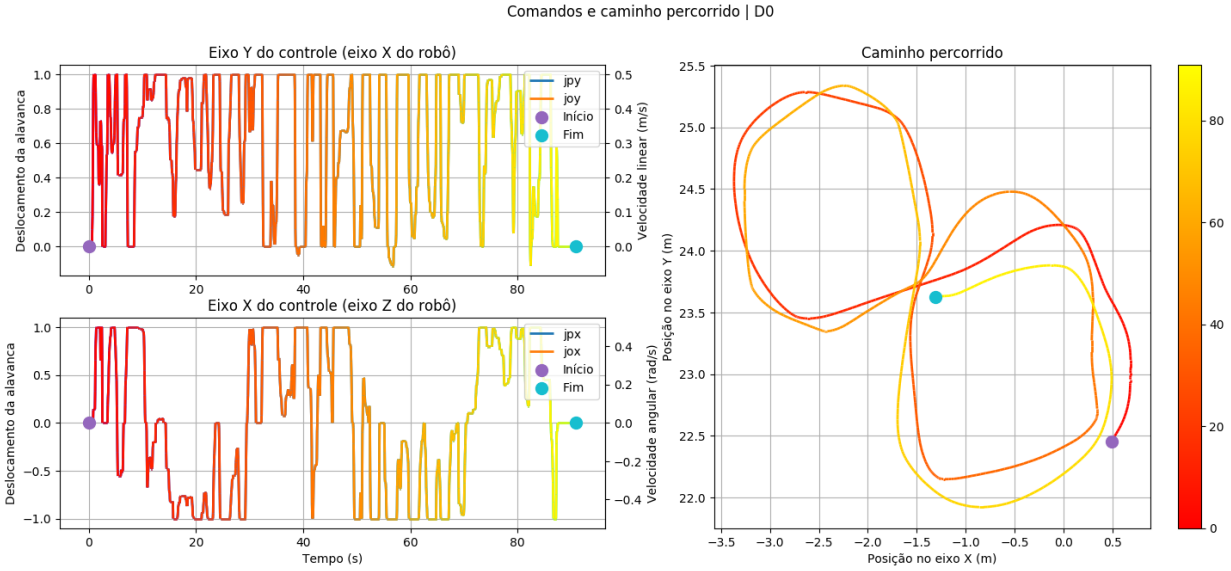

Figura 4.12: Gráficos do usuário A para testes sem correção. Os sinais de comando oscilam bastante, o que é visível no percurso. Fonte: o autor.

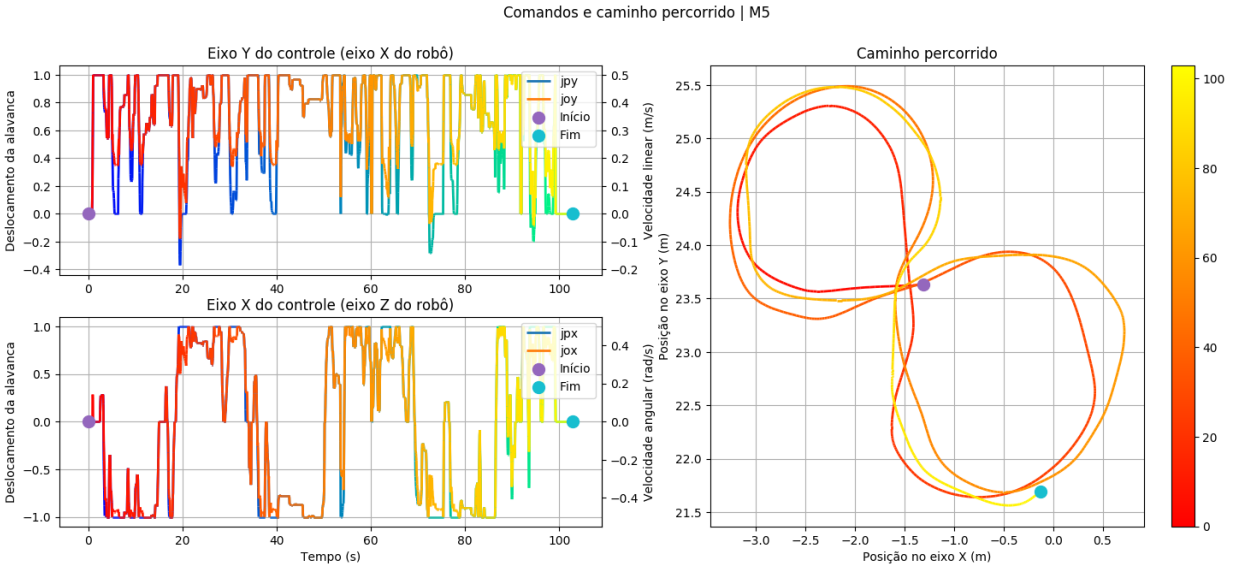

Figura 4.13: Gráficos do usuário A para testes com M5. Pode-se perceber que comandos estão mais estáveis, que resulta numa trajetória mais suave. Fonte: o autor.

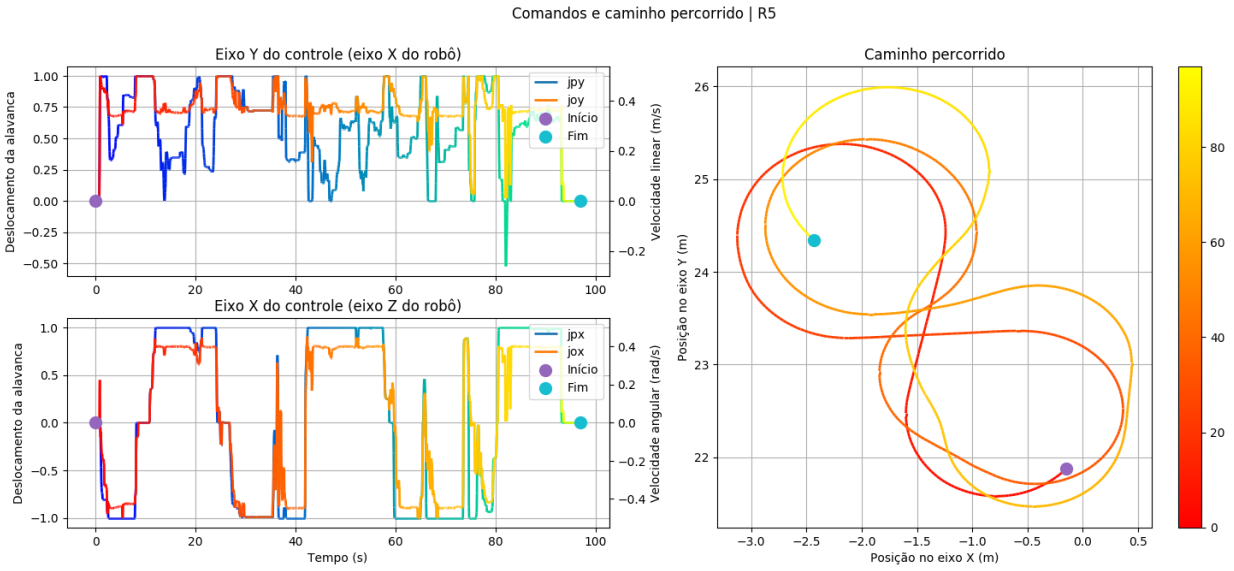

Figura 4.14: Gráficos do usuário A para testes com R5. Este foi o método preferido do usuário, e pelos comandos, percebe-se que o algoritmo manteve as velocidade linear e angular razoavelmente constantes, que resulta na trajetória mais suave dos três métodos. Fonte: o autor.

Tabela 4.8: Resultados dos testes para o usuário A. O método R5 foi o preferido do usuário e resultou em maior suavização, indicado pelas menores acelerações linear e angular.

Método	pa (%)	$diffm$ (%)	am_x (m/s^2)	αm_z (m/s^2)
D	-	-	0.1904	0.2902
M5	27.77	4.83	0.1582	0.2791
R5	78.47	10.42	0.0638	0.1707

4.4.2 Usuário B

Para o segundo caso, com o usuário B, temos a realização dos testes indicados na Tabela 4.9, na ordem M5 → R5, e os gráficos correspondentes nas Figuras 4.15 e 4.16, como anteriormente. O usuário B conseguiu seguir o oito com certa facilidade na correção por média mas com a correção por rede sentiu mais dificuldade, e por isso acabou preferindo a correção por média.

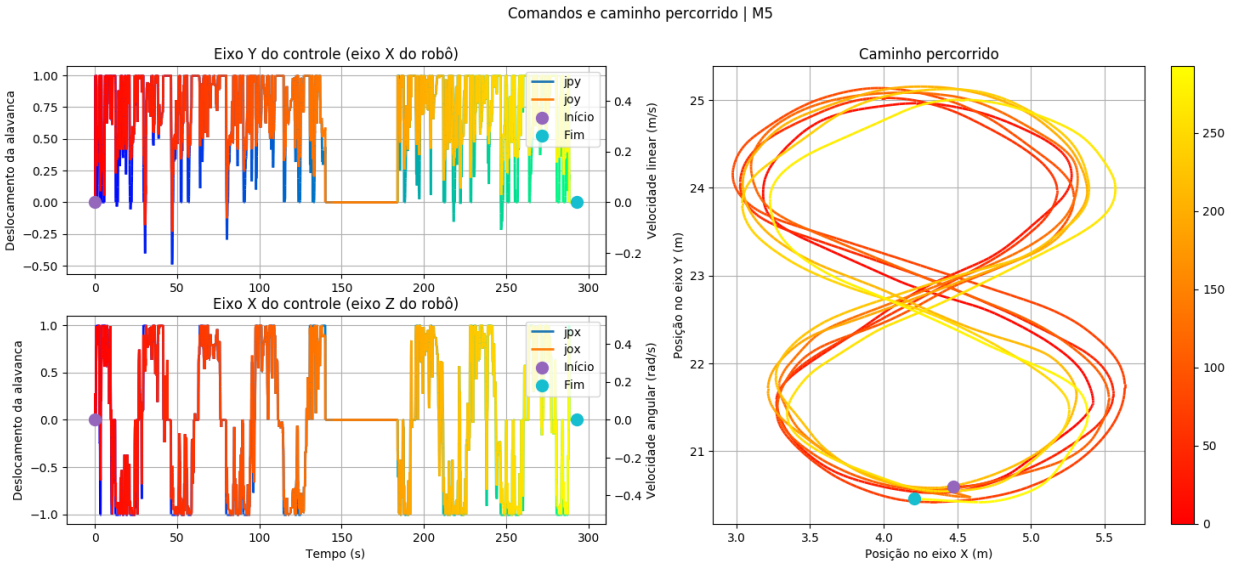

Figura 4.15: Gráficos do usuário B para testes com M5. Apesar de a velocidade linear estar mais instável, a trajetória do usuário está bastante uniforme, com as voltas bem alinhadas. Durante o teste, o sentido das voltas foi invertido, evidenciado pela pausa nos comandos. Fonte: o autor.

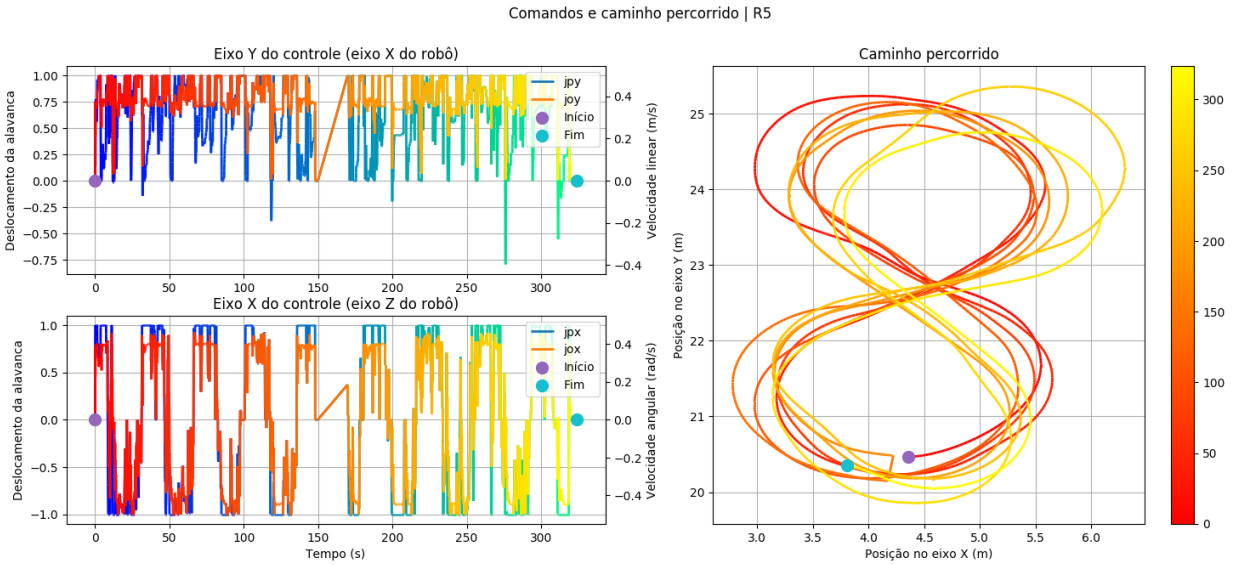

Figura 4.16: Gráficos do usuário B para testes com R5. A velocidade linear neste caso está mais estável, mas como o perfil de pilotagem do usuário é diferente do perfil da rede, a trajetória está menos uniforme. Fonte: o autor.

Tabela 4.9: Resultados dos testes para o usuário B. O método preferido foi o M5, que confere mais liberdade na pilotagem do robô.

Método	*pa* (%)	*diffm* (%)	am_x (m/s^2)	αm_z (m/s^2)
M5	21.66	4.05	0.1477	0.2858
R5	62.71	9.87	0.0958	0.2311

Novamente, os dados coletados seguem o padrão esperado, em que a correção por rede resulta em maior suavização. Entretanto, na trajetória do usuário B, o usuário saiu algumas vezes do trajeto, que pode ser notado na Figura 4.16. Nos comandos, há uma tentativa de impor suavidade pela rede, mas o algoritmo encontra alguma resistência da parte do usuário.

Observando a tabela do usuário B, temos que a mesma correção por rede acarretou em menos suavização do que no caso do usuário A. Isto possivelmente se deve à dissimilaridade do perfil do usuário B para com o perfil da rede. O usuário teve que constantemente ajustar sua pilotagem por conta das sugestões do algoritmo, e isso acabou aumentando a aceleração média.

4.4.3 Usuário C

Para o terceiro caso, com o usuário C, temos a realização dos testes indicados na Tabela 4.10, na ordem M20 → R20, e os gráficos correspondentes nas Figuras 4.17 e 4.18, como anteriormente. O usuário C tinha um perfil de pilotagem bastante diferente do esperado, que pode ser visto na trajetória da Figura 4.17. A imposição de suavidade também não agradou ao usuário C, que preferiu a correção por média.

Para o usuário C, am_z e αm_z tem grandeza ligeiramente maior do que para os usuários A e B. Além disso, αm_z para os dois métodos tem valores próximos. Durante o acompanhamento dos testes, pode-se perceber que o usuário teve muita dificuldade em pilotar do jeito como ele gostaria. Sempre que o usuário tentava girar o robô em torno do próprio eixo, o algoritmo impedia, e só permitia que as curvas fossem feitas se existisse velocidade linear.

Isto evidencia uma característica importante da base de dados. Em nenhum momento da coleta de dados o robô foi comandado para girar em torno do próprio eixo, com v_x 0 e $|\omega_z > 0|$, ou comandado de ré, com velocidade linear negativa, ou $v_x < 0$. Assim, a rede não aprendeu este tipo de movimento e o algoritmo de correção por rede nunca sugeriu este tipo de movimentação.

Tabela 4.10: Resultados dos testes para o usuário C. O usuário preferiu o método M20, novamente por conta da maior liberdade que o método confere durante a pilotagem.

Método	*pa* (%)	*diffm* (%)	am_x (m/s^2)	αm_z (m/s^2)
M20	0.44	5.09	0.1606	0.3335
R20	18.15	10.28	0.1436	0.3405

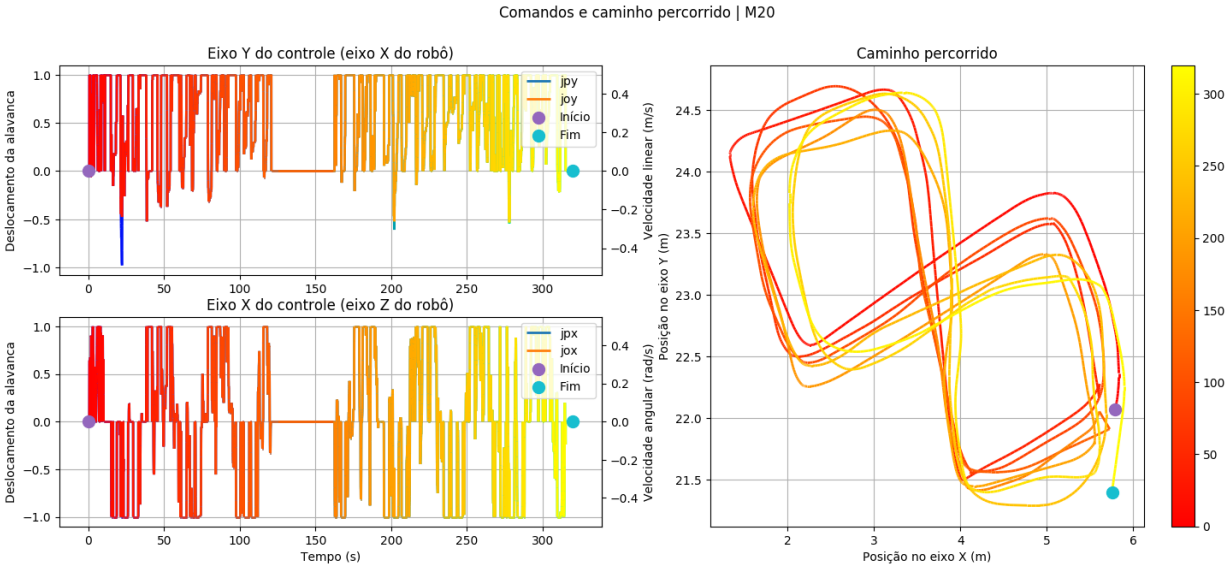

Figura 4.17: Gráficos do usuário C para testes com M20. Neste caso, o perfil do usuário é bastante diferente do perfil da rede, uma vez que o usuário tentou fazer o oito com linhas retas e curvas em torno do eixo do robô. Fonte: o autor.

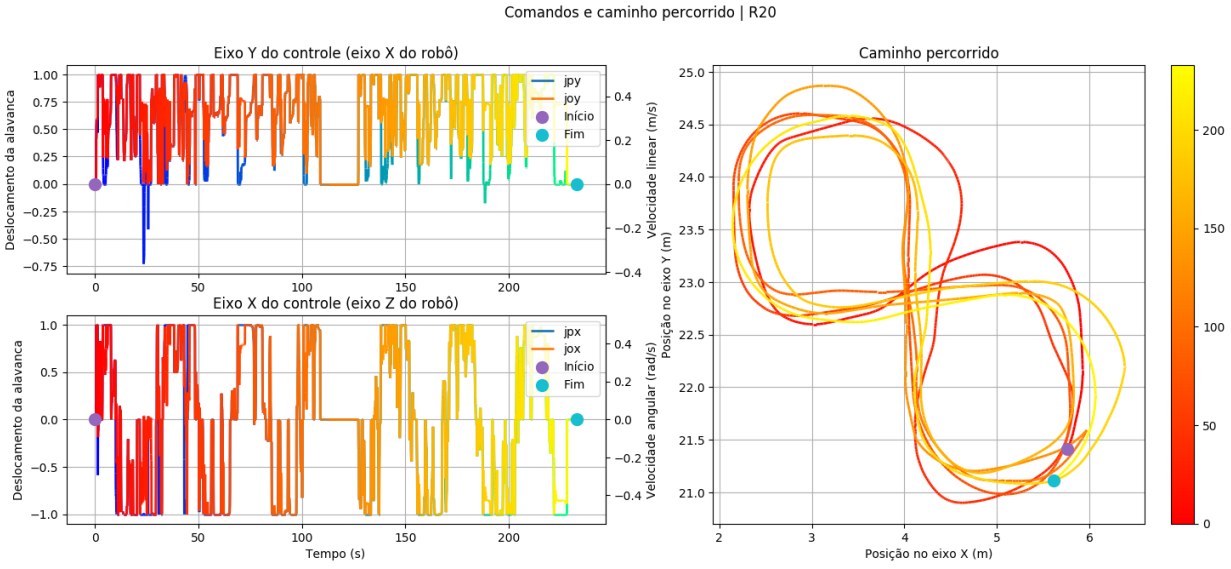

Figura 4.18: Gráficos do usuário C para testes com R20. Há uma tentativa do algoritmo de conformar o usuário ao perfil da rede, evidenciado pelo formato mais arredondado do oito. Isto, no entanto, fez com que o usuário não gostasse deste método. Fonte: o autor.

4.5 Discussões gerais

Verifica-se nas Tabelas 4.1 e 4.2 que o aprendizado aconteceu corretamente, as arquiteturas propostas aprenderam bem a sequência de dados, e as arquiteturas $N = (96, 64, 32)$ e $N = (128, 96, 64, 32)$ tiveram melhor desempenho.

Nas Tabelas 4.5 e 4.6, pôde-se verificar que a correção por rede tende a suavizar mais a direção do piloto, principalmente com limiar de 5%. Por conta das diferenças de perfil e maior dificuldade do percurso no oito, na tabela temos maior variabilidade nos dados. Pode-se perceber que a rede estava atuando como desejado, suavizando os comandos de velocidade, ainda que alguns usuários não gostassem da correção.

Nas entrevistas, de acordo com as Figuras 4.9, 4.10 e 4.11, observou-se uma tendência de preferência aos métodos que interfiram menos na pilotagem. Os casos em que os usuários gostaram de pilotar com a rede foram os casos em que o seu perfil de direção era parecido com o do banco de dados. No entanto, um motorista que aprendeu a dirigir um carro incorretamente também oferece mais resistência a correções quando aprende a dirigir corretamente em uma auto-escola. Assim, pode-se argumentar que a preferência por liberdade na pilotagem pelos usuários não implica rejeição dos resultados obtidos. Neste trabalho, a definição do que é certo e errado está partindo de uma definição arbitrária de suavidade, mas a analogia ainda vale.

4.5.1 Melhorias e trabalhos futuros

Conforme já mencionado, durante o acompanhamento dos testes, foi observado que o perfil de pilotagem variou bastante entre os usuários. Qualidades como a familiaridade do usuário com um controle de videogame, a experiência com jogos e a idade afetam o perfil de pilotagem. Algumas pessoas apresentaram pouca dificuldade para guiar o robô, mesmo no percurso em oito, uma vez que a velocidade do robô era de $0.5m/s$, mais lento que a caminhada normal de um humano. Devido a pequena quantidade de pessoas que validaram o algoritmo, os resultados indicam apenas tendências, e não conclusões estatísticas de alta confiabilidade.

Para a base de dados, temos para o percurso em oito um perfil de velocidade linear indesejado, que pode ter injetado ruído nas predições da rede, e pode-se argumentar que a quantidade de dados coletados não foi volumosa o suficiente. Ambos os aspectos podem ser melhorados em trabalhos futuros. Além disso, a base não contém movimentos em que o robô gira no próprio eixo ou se movendo de ré, que pode ou não ser uma adição pertinente à base de dados.

Outro trabalho futuro possível para a base dados seria a inclusão de imagens no treinamento. Isso introduziria novos problemas, no entanto. Existiria a necessidade de arquiteturas mais robustas por conta de variações de iluminação nas imagens e a base possivelmente se tornaria específica ao ambiente em que os dados foram capturados.

Para o algoritmo de correção, houve um pequeno atraso na pilotagem do robô por conta da frequência de publicação do nó *correct*. Em um trabalho futuro, seria interessante aumentar a frequência para que não sobrasse tempo útil entre as predições da rede e publicação dos comandos.

O chaveamento entre o comando do piloto e do algoritmo não é uma transição suave, principalmente no limiar de 20% de diferença. Uma melhoria possível seria um usar um filtro de Kalman com modelo rudimentar em espaço de estados para a fusão dos dados. Somado a isto, pode-se usar uma fusão personalizada a depender do perfil de pilotagem do usuário, algo que pode ser identificado e chaveado em tempo real, uma vez que um perfil muito dissimilar ao da rede acaba acarretando em uma correção desagradável ao usuário.

Uma melhoria técnica no código seria a utilização da saída do seletor como entrada da rede e não simplesmente os comandos do piloto. As redes LSTM tem memória das últimas entradas, e seria mais interessante que esta memória fosse referente ao que realmente foi enviado ao robô para que a próxima predição seja mais correspondente com a realidade, uma vez que o comando do piloto nem sempre é efetivamente enviado ao robô.

Por fim, um outro trabalho futuro poderia utilizar os algoritmos de correção aqui propostos para um segundo algoritmo completamente autônomo de pilotagem, com a finalidade de suavizar sugestões espúrias de velocidade ou mudanças muito bruscas.

Capítulo 5

Conclusões

A proposta deste trabalho foi de propor e validar modelos de redes neurais que pudessem predizer qual deveria ser o próximo comando na pilotagem de um usuário inexperiente baseado na pilotagem de um usuário experiente.

Foram definidas as técnicas de pesquisa utilizadas e os procedimentos experimentais foram listados. Além disso, foram propostas as arquiteturas das redes neurais e foi definida a estrutura do algoritmo de correção quanto aos métodos utilizados, que poderiam ser baseados em correção por rede ou correção por média.

Uma base de dados foi construída com a pilotagem de um operador experiente, e as arquiteturas propostas foram validadas quantitativamente, verificando que a validação cruzada e o aprendizado convergiram em modelos satisfatórios, e validadas qualitativamente, com testes com voluntários e entrevistas. Destes testes, foram verificadas algumas tendências, como a preferência dos usuários por liberdade na pilotagem.

Por fim, pôde-se avaliar, de acordo com a definição de suavidade deste trabalho, que os métodos de correção propostos resultam em suavização maior dos comandos de velocidade do que a pilotagem sem correção, e a correção por rede resultou em uma suavização maior do a correção por média. Isso mostra que os métodos de correção propostos cumprem seu objetivo e suavizam a pilotagem dos usuários, indicando a viabilidade do uso de redes LSTM em módulos de auxílio de direção para veículos robóticos.

5.1 Perspectivas Futuras

Para melhor desempenho da correção, podem ser feitas melhorias na qualidade e volume da base de dados, além da inclusão de imagens no conjunto de dados utilizados. Podem ser feitas melhorias também na fusão dos comandos, que poderia ser realizada por exemplo um filtro de Kalman, ou até ser personalizada em tempo real a depender do perfil de direção do usuário. Por fim, para que o algoritmo seja melhor avaliado, em testes futuros é ideal que os voluntário sejam pessoas com pouca destreza com as mãos.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] Amazon's New Robots Are Shipping Your Order This Holiday | Time. `http://time.com/3605924/amazon-robots/`. Acessado em: 7 dez. 2018.

[2] Robô Aspirador Roomba | iRobot. `https://www.irobot.com.br/Robos-domesticos/Vacuum-cleaning`. Acessado em: 7 dez. 2018.

[3] Robôs Industriais: tudo o que você precisa saber! | Pollux. `https://www.pollux.com.br/blog/robos-industriais-tudo-o-que-voce-precisa-saber/`. Acessado em: 7 dez. 2018.

[4] CAMPION, G.; CHUNG, W. Wheeled rob 17 . wheeled robots. In: . [S.l.: s.n.], 2008.

[5] GHOSH, S.; KRISTENSSON, P. O. Neural networks for text correction and completion in keyboard decoding. *CoRR*, abs/1709.06429, 2017. Disponível em: <http://arxiv.org/abs/1709.06429>.

[6] DAVIDS, A. Urban search and rescue robots: from tragedy to technology. *IEEE Intelligent Systems*, v. 17, p. 81–83, Mar-Abr 2002.

[7] BAJRACHARYA, M.; MAIMONE, M. W.; HELMICK, D. Autonomy for mars rovers: Past, present, and future. *Computer*, v. 41, p. 44–50, Dez 2008.

[8] BINNEY, J.; KRAUSE, A.; SUKHATME, G. S. Informative path planning for an autonomous underwater vehicle. *2010 IEEE International Conference on Robotics and Automation*, Mai 2010.

[9] ENDSLEY, M. R. Autonomous driving systems: A preliminary naturalistic study of the tesla model s. *Journal of Cognitive Engineering and Decision Making*, v. 11, p. 225–238, Set 2017.

[10] DUDEK, G.; JENKIN, M. *Computational Principles of Mobile Robotics*. 2. ed. [S.l.]: Cambridge University Press, 2010.

[11] EVERETT, H. R. *Sensors for Mobile Robots*. 1. ed. [S.l.]: A K Peters, Ltd, 1995.

[12] How do Roomba robot vacuum cleaners work? `https://www.explainthatstuff.com/how-roomba-works.html`. Acessado em: 7 dez. 2018.

[13] PARK, J. H. Impedance control for biped robot locomotion. *IEEE Transactions on Robotics and Automation*, v. 17, p. 870–882, Dez 2001.

[14] BOEF, A. J. den. Interferometric laser rangefinder using a frequency modulated diode laser. *Applied Optics*, v. 26, p. 4545–4550, Nov 1987.

[15] BOEF, A. J. den. Detecting moving objects using a single camera on a mobile robot in an outdoor environment. *8th Conference on Intelligent Autonomous Systems*, Mar 2004.

[16] TOLANI, D.; GOSWAMI, A.; BADLER, N. I. Real-time inverse kinematics techniques for anthropomorphic limbs. *Graphical Models*, v. 62, p. 353–388, Set 2000.

[17] ZHAO, J.; BADLER, N. I. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics (TOG)*, v. 13, p. 313–336, Out 1994.

[18] KING-HELE, D. Erasmus darwin's improved design for steering carriages—and cars. In: THE ROYAL SOCIETY. [S.l.], 2012. v. 56.

[19] THRUN, S.; BURGARD, W.; FOX, D. *Probabilistic Robotics*. 1. ed. [S.l.]: MIT Press, 2005.

[20] THRUN, S. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, Jul 1998.

[21] HUANG, A. S. et al. Visual odometry and mapping for autonomous flight using an rgb-d camera. *Robotics Research*, v. 100, p. 235–252, Aug 2017.

[22] BISWAS, J.; VELOSO, M. Depth camera based indoor mobile robot localization and navigation. *2012 IEEE International Conference on Robotics and Automation*, Mai 2012.

[23] DAVISON. Real-time simultaneous localisation and mapping with a single camera. *Proceedings Ninth IEEE International Conference on Computer Vision*, Out 2003.

[24] GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. 1. ed. [S.l.]: MIT Press, 2016. `http://www.deeplearningbook.org`.

[25] CHOLLET, F. *Deep Learning with Python*. 1. ed. [S.l.]: Manning Publications, 2017.

[26] KONONENKO, I. Machine learning for medical diagnosis: history, state of the art and perspective. *Artificial Intelligence in Medicine*, v. 23, p. 89–109, Ago 2001.

[27] AGARAP, A. F. Deep learning using rectified linear units (relu). *CoRR*, abs/1803.08375, 2018. Disponível em: <http://arxiv.org/abs/1803.08375>.

[28] GLOROT, X.; BENGIO, Y. Understanding the difficulty of training deep feedforward neural networks. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. [S.l.: s.n.], 2010. p. 249–256.

[29] CUN, Y. le. A theoretical framework for back-propagation. *Proceedings of the 1988 Connectionist Models Summer School*, p. 21–28, 1988.

[30] BAR, Y. et al. Deep learning with non-medical training used for chest pathology identification. *Medical Imaging 2015: Computer-Aided Diagnosis*, v. 9414, Mar 2015.

[31] REN, S. et al. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems 28*, p. 91–99, Out 2015.

[32] HINTON, G. et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, v. 29, p. 82–97, Out 2012.

[33] KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. *NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, p. 1097–1105, Dez 2012.

[34] MIKOLOV, T. et al. Extensions of recurrent neural network language model. *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Mai 2011.

[35] GRAVES, A.; MOHAMED, A.; HINTON, G. E. Speech recognition with deep recurrent neural networks. *Computing Research Repository (CoRR)*, abs/1303.5778, 2013. Disponível em: <http://arxiv.org/abs/1303.5778>.

[36] HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural Computation*, v. 9, p. 1735–1780, Nov 1997.

[37] PAN, S. J.; YANG, Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering*, v. 22, p. 1345–1359, Out 2010.

[38] DENG, J. et al. Imagenet: A large-scale hierarchical image database. *2009 IEEE Conference on Computer Vision and Pattern Recognition*, Out 2009.

[39] SHI, Z. et al. Lstm-based flight trajectory prediction. *2018 International Joint Conference on Neural Networks (IJCNN)*, Jul 2018.

[40] KINGMA, D. P.; BA, J. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2015. Disponível em: <http://arxiv.org/abs/1412.6980>.

[41] Repositório no GitHub deste trabalho. `https://github.com/thvlio/tg2`. Acessado em: 2 jul. 2019.

ANEXOS

I. PROGRAMAS UTILIZADOS

A base de dados, os modelos obtidos e os programas utilizados para treinamento e teste dos modelos estão disponíveis em [41]. A estrutura do repositório está descrita no README.

Em *config.py* estão todas os parâmetros e hiperparâmetros utilizados por outros códigos. Os *scripts* *.sh são referentes à separação e extração de dados e resultados armazenados em arquivos *.bag*. Os programas *train.py*, *compare.py* e *test.py* foram usados para realizar o treinamento e teste das redes neurais. Os programas *cv_*.py* são referentes à validação cruzada. Os programas *send_*.py* são *scripts* utilizados na plataforma DevCloud, em que vários treinamentos e testes podem ser instanciados ao mesmo tempo em servidores diferentes.

Em *data/* estão algumas das bases de dados criadas para este trabalho. Dentre elas, a base *predio/* refere-se ao percurso nos corredores do prédio e a base *oito/* refere-se ao percurso em oito.

Em *models/* estão os modelos obtidos nos experimentos. Os nomes das pastas se referem às pastas presentes em *data/*, e a pasta *cross-validation* contém os resultados das validações cruzadas.

Dentro de *ros/* está o pacote criado para implementação dos algoritmos de correção.