TRABALHO DE GRADUAÇÃO

APRENDIZADO DE COMPORTAMENTOS REATIVOS UTILIZANDO REDES NEURAIS CONVOLUCIONAIS EM ROBÔS MÓVEIS

Túlio Mariano da Silva Lima

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 DE COMPORTAMENTOS REATIVOS UTILIZANDO REDES NEURAIS CONVOLUCIONAIS EM ROBÔS MÓVEIS

Túlio Mariano da Silva Lima

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

Banca Examinadora

Profa. Dra. Carla Koike, CIC/UnB _____
Orientador

Prof. Dr. Flávio Vidal, CIC/UnB _____
Co-orientador

Prof. Dr. Marcus Lamar, CIC/UNB _____

Prof. Dr. Adolfo Bauchspiess, ENE/UNB _____

Brasília, julho de 2019

FICHA CATALOGRÁFICA

TÚLIO, MARIANO DA SILVA LIMA
Aprendizado de Comportamentos reativos utilizando Redes neurais convolucionais em Robôs móveis.

[Distrito Federal] 2019.

XIII, 60p., 297 mm (FT/UnB, Engenheiro, Controle e Automação, 2019). Trabalho de Graduação – Universidade de Brasília.Faculdade de Tecnologia.

1. Redes Neurais 2.Robótica
3. Comportamentos Reativos

I. Mecatrônica/FT/UnB

REFERÊNCIA BIBLIOGRÁFICA

LIMA, TÚLIO MARIANO DA SILVA, (2019). Aprendizado de Comportamentos reativos utilizando Redes neurais convolucionais em Robôs móveis. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT.TG-$n°3$, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 60p.

CESSÃO DE DIREITOS

AUTOR: Túlio Mariano da Silva Lima

TÍTULO DO TRABALHO DE GRADUAÇÃO: Aprendizado de comportamentos reativos utilizando redes neurais convolucionais em robôs móveis.

GRAU: Engenheiro ANO: 2019

Túlio Mariano da Silva Lima

SQN 406 Bloco A, nº 208, Asa Norte.

70847-010 Brasília – DF – Brasil.

Dedicatória

Aos meus pais e minha irmã.

Túlio Mariano da Silva Lima

Agradecimentos

Agradeço minha família, por sempre terem me apoiado nos momentos mais difíceis da minha graduação. Em especial minha mãe, Cândida, por quem eu tenho um profundo carinho, e que junto de meu pai, Lucenildo, sempre fizeram o possível para me dar uma boa educação. Agradeço minha irmã, Fernanda, pelo apoio emocional e pelas molecagens. Aos meus melhores orientadores, Professora Dra. Carla Koike e Professor Dr. Flávio Vidal, agradeço por todo o suporte que me foi dado e aproveito pra dizer que foi fundamental para apresentação desse trabalho.
Por fim, também deixo aqui os agradecimentos aos meus amigos de Brasília e da UnB, com os quais tive amplas alegrias.

Túlio Mariano da Silva Lima

RESUMO

A robótica, juntamente com a Inteligência artificial, tem apresentado diversas soluções inovadoras para o auxiliar o homem em suas tarefas. Nas implementações em robótica móvel, que envolvem a exploração de ambientes, existem muitos desafios e que são difíceis de serem alcançados sem fazer uso de técnicas de *machine learning*. Esses desafios são solucionados com o uso de redes neurais, no ritmo que também evoluem em complexidade, e possibilitam que a tarefa de programar tome um rumo mais associado a orientar e ensinar máquinas a realizarem tarefas. Com isso, esse trabalho apresenta uma implementação utilizando uma rede neural convolucional para reproduzir comportamentos reativos em um robô móvel, a partir da estimativa do ângulo de esterçamento, ou sentido de orientação. Esses comportamentos se traduzem em reações ao que foi percebido, com a ajuda de imagens, e com os treinamentos realizados para a rede proposta foi obtido um resultado que orientou o robô satisfatoriamente (de forma reativa) durante sua navegação autônoma.

Palavras Chave: Robótica, Redes Neurais, Aprendizado de Máquina, Comportamentos Reativos, Pioneer.

ABSTRACT

Robotics, along with Artificial Intelligence, has presented several innovative solutions to assist man in his tasks. In mobile robotic implementations that involve the exploration of environments, there are many challenges that are difficult to achieve without using machine learning techniques. These challenges are solved with the use of neural networks, in the rhythm that also evolve in complexity, and allow the task of programming to take a more associated course to guide and teach machines to perform tasks. Thus, this work presents an implementation using a convolutional neural network to reproduce reactive behaviors in a mobile robot, from the estimation of the steer angle, or direction orientation. These behaviors translate into reactions to what was perceived, with the help of images, and with the trainings performed for the proposed network, a result was obtained that guided the robot satisfactorily (in a reactive mode) during its autonomous navigation.

Keywords: Robotics, neural networks, machine learning, reactive behaviors, Pioneer.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Símbolos

ω	Parâmetros do Modelo - *Vetor de Pesos*	
x	Dado de Entrada - *Vetor de Entrada*	
$\hat{y}$	Valor estimado - *Vetor de Saída*	
y_{teste}	Amostras de teste - *Vetor de amostras almejadas para teste*	
$Temp$	Temperatura	
$STemp$	Sensação Térmica	
$Temp_{teste}$	Amostras de temperatura em testes	
η	Taxa de Aprendizado	
K	filtro de convolução	
p	conjunto dos filtros K	
I	Imagem	
m, n	Dimensões da Imagem	
$kernel$	matriz que constitui o filtro	
X	Eixo X	
Y	Eixo Y	
V	Velocidade Linear	[m/s]
W	Velocidade Angular ou Esterçamento	
$\times$	Multiplicação	
$*$	Convolução	

Grupos Adimensionais

i, k	Contador

Subscritos

$teste$	Conjunto de dados de teste
i	Índice dos conjunto de dados
$esquerda$	Rodas da Esquerda do Robô
$direita$	Rodas da Direita do Robô

Sobrescritos

$\hat{}$	Valor estimado
τ	Tarefa relacionada à regressão
$'$	Eixo Relativo

Siglas

IFR	Federação Internacional de Robótica - *International Federation of Robotics*
IA	Inteligência Artificial
T	Tarefa
P	Desempenho
E	Experiência
MSE	Erro médio quadrático
CPU	Unidade Central de Processamento - *Central Processing Unit*
GPU	Unidade de Processamento Gráfico - *Graphics Processing Unit*
TPU	Unidade de Processamento de Tensores - *Tensor Processing Unit*
LIDAR	*Light Detection And Ranging*
SENSE	Percepção
PLAN	Planejamento
ACT	Ação
PWN	*Pulse Width Modulation*
ARIA	*Advanced Robotics Interface for Applications*
ROS	*Robot Operating System*
ML	*Machine Learning*
DL	*Deep Learning*
RN	Rede Neural
ELU	Tipo específico de camada de ativação - *Exponential Linear Unit*

Capítulo 1

Introdução

1.1 Contextualização

No cenário desta década, as tecnologias em software, que fazem uso programas que 'aprendem' como executar tarefas, vem ganhando espaço na forma de produtos, apesar dessa ideia não ser tão recente e existir pelo menos desde a década de 50[14]. Dentre essas técnicas, a abordagem feita por redes neurais artificiais tem ganhando destaque na sua proposta de automatizar esforços repetitivos e também para facilitar a programação de funções operacionais. Dessa forma, é possível instruir um computador como reagir através da observação de padrões, o que permite uma maior diversidade e complexidade no desenvolvimento de $'robôs\ inteligentes'$.

Robôs foram apresentados ao mundo com o conceito de uma criatura mecânica, feita à imagem do ser humano, que funciona de maneira autônoma, inclusive capaz de realizar tarefas feitas por pessoas [15]. Entretanto sua implementação mostra que existem diversas limitações relacionadas à inteligência e à capacidade de solução de problemas; e geralmente de emprego em situações onde existe riscos elevados ou mesmo em atividades repetitivas e maçantes. Atualmente, a imagem associada a essas máquinas tem relação concreta associada principalmente à práticas realizadas dentro de fábricas, linhas de montagem, e constituem uma peça essencial dentro de cadeias produtivas de grande demanda. Segundo a estimativa apresentada pela Federação Internacional de Robótica (IFR), existe uma demanda crescente pela adoção de robôs no setor da indústria mundial [16], e ilustra a quão solicitado são as aplicações dessas máquinas.

De maneira lúdica, o cinema apresenta os robôs como figuras ultra-inteligentes, dedicadas e até cômicas; mas atualmente a prática revela um cenário longe desse que foi idealizado[15]. Ilustrando o cenário ideal com a implementações robóticas que solucionem problemas de maneira autônoma e façam escolhas inteligentes, tais como em filmes de ficção científica, a exemplo dos famosos $R2D2$ na saga $Star\ Wars$ e David no filme $I.A.$

Por essência, o que torna o robô diferente de um computador é justamente a sua capacidade de tomar decisões inteligentes e de possuir uma estrutura que lhe permite atuar diretamente no ambiente. Logo, máquinas que realizam movimentos repetitivos, como robôs de solda, não são exatamente inteligentes até que os mesmos tenham a capacidade de usar o computador tal como

um cérebro humano.

O estudo de como a inteligência se dá na natureza é inspirado principalmente pelo o que é observado em animais, em especial nas formas e tipos de comportamentos [15]. Essa inspiração fornece a base do que inicialmente motivou as pesquisas de inteligência artificial; e no ponto que tange essa inteligência programada, tanto a computação como as aplicações de IA's tem crescido bastante com o advento de técnicas de aprendizado e hardware adequado. A robótica, que também vê os comportamentos (*behaviors*) como compositores de inteligência, teve seu desenvolvimento alavancado pelo uso de implementações de IA na construção de sistemas autônomos, o que também permitiu realizações mais próximas de que é visto como inteligente; logo o problema tomou a direção mais ligada a como o computador aprende e o que ele pode aprender.

Inicialmente, em 1970, os estudiosos dessas áreas correlatas buscaram por implementações de comportamentos bioinspirados, com elaborações simples que imitam reflexos observados na natureza, apresentado no trabalho proposto anteriormente por Grey Walter [17] (1950). A abordagem por IA nesses problemas permite um grau de complexidade maior nas implementações robóticas, e atualmente as tarefas possibilitadas pela construção de soluções robóticas em IA tem objetivos grandes, e de complexidade não envolvidas apenas de reações, como exemplo do iCub [18] que aprendeu o tiro com arco e flecha [19].

No que diz respeito as técnicas de aprendizado de máquina, as redes neurais convolucionais apresentam bons resultados na resolução de problemas que envolvem grande quantidade de dados, principalmente quando se trata de imagens. E assim, motivado pelas diversas questões relacionadas à execução de tarefas em um robô e pelo impacto da implementação, este trabalho desenvolveu uma metodologia para uma arquitetura de robótica móvel, a qual será descrita a seguir.

1.2 Definição do problema

No que envolve especificamente os comportamentos reativos, que são baseados na função de apresentação de reflexos para o que foi percebido[8], existe uma dificuldade associada à sua concepção (por código ou hardware) e que é diretamente ligada à complexidade da função. O computador por trás do algoritmo que guia um robô para imitar um inseto, por exemplo, é programado para traduzir 'sentidos' em 'reflexos'; e exatamente desse ponto de partida que tem-se um grande problema quando existe a necessidade de programar um robô para agir de maneira mais elaborada, como um humano, para interagir com o ambiente e realizar passos de dinâmica complexa.

Considere, por exemplo, o robô humanoide Asimo[20], que possui uma grande quantidade de juntas rotativas que compõem 34 graus de liberdade e que executa a tarefa de caminhar com uma certa velocidade. Essa atividade pode parece simples aos olhos leigos mas que envolve um detalhamento cinemático de como o conjunto interage e das informações dos diversos sensores disponíveis, que por sua vez exigem um grande poder de processamento e dessa forma tornam o trabalho mais custoso ou *'hard coded'* (direcionado) para um modo de funcionamento.

Além do intuito de entender como o processo de aprendizado é consolidado, os estudos em

inteligencia artificial focam em maneiras de simplificar as formas de interação com o computador e principalmente como o computador possa vir a relaciona-se com o mundo. E dada as dificuldades apresentadas por implementações de grande complexidade, uma tática inteligente é aplicar técnicas de aprendizado de máquina para assimilação da 'inteligência', sem que seja necessário propriamente codificar comportamentos e de uma maneira mais direta, com apresentação de objetivos.

Partindo desses princípios, um robô móvel como o Pioneer e uma rede neural, juntos, podem desempenhar comportamentos reativos e apresentar soluções para problemas de robótica que não são triviais. Dessa forma o robô pode ser orientado por uma rede neural, de forma que esta faça uso da informação de câmera como estímulo e apresente reflexos como saída.

Fruto dessa tática, os questionamentos de partida podem envolver: o que é necessário fornecer para o algoritmo de aprendizado para que este aprenda o que é desejado? Que dados devem ser fornecidos nesse processo? Quais comportamentos e quais sensores estão ligados nesse processo? Como avaliar os resultados?. E considerando os pressupostos adotadas nesse trabalho, que envolvem o uso da plataforma de exploração (Pioneer 3AT[10]) e rede neural convolucional, é desejado o empreendimento de um sistema que possa aprender a reagir e mimetizar comportamentos apresentados por um piloto, de maneira análoga ao que é feito por sistemas de guiagem de automóveis.

1.3 Objetivo e Metodologia

Os objetivos e metodologia deste trabalho foram propostos de forma a delimitar o problema a ser tratado neste manuscrito e para a melhor obtenção da solução, conforme apresentado nas subseções à seguir.

1.3.1 Objetivo Principal

Desenvolver uma abordagem metodológica a qual permita-se incorporar comportamentos reativos em um robô móvel, utilizando redes neurais convolucionais. Espera-se que a partir desta implementação, permita-se que a arquitetura robótica adquira comportamentos reativos para a navegação autônoma em ambientes.

1.3.2 Metodologia

Os passos utilizados para compor a metodologia, que está direcionada a conquista do objetivo principal, são descritos a seguir:

- Estudo e proposição de modelo de uma arquitetura convolucional para treinamento e estimação de Ângulo de esterçamento;
- Elaboração de base de dados anotadas de imagens de navegação em robótica móvel, para cada comportamento;

Preparação de ambiente de coleta entre computadores;

Redimensionamento e ajuste de indexes, nos dados.

- Definições de comportamentos reativos primários para o processo de navegação de robôs móveis;
- Execução de treinamentos compostos por números de épocas distintas;

  Testar um comportamento simples de maneira isolada;

  Compor comportamentos;

- Avaliação de erros, para metrificar os resultados e verificação da capacidade de generalização, e análise de comportamentos não esperados;
- Teste de controle autônomo;
- Avaliação de erros ou atrasos provenientes dos equipamentos;

1.4 Justificativa

No que envolve as redes neurais, esse trabalho apresenta uma forma de aplicar/mimetizar comportamentos reativos com redes convolucionais, assim como compor e decidir quais reações são mais adequadas, utilizando-se de um robô Pioneer. Com isso, contribuir com uma abordagem na implementação de sistemas bioinspirados.

A principal inspiração revela-se na abstração que a rede pode proporcionar, a respeito da programação de tarefas que serão executadas por um robô e da maneira alternativa de controle.

Para a robótica móvel, o trabalho apresenta maiores horizontes de implementação, e permite que a predição/composição realizada pela rede ajude na solução de desafios, como por exemplo, a guiagem autônoma de veículos.

1.5 Apresentação do manuscrito

No decorrer do texto, os próximos capítulos abordarão as questões primordiais que são relacionadas a cada parte deste trabalho, na ordem descrita. O Capítulo 2, ilustra um panorama inicial das técnicas de aprendizado de máquina e *Deep learning*, incluindo os conceitos, formas comuns, aplicações, motivação, formulação e as possibilidade resultantes.

No Capítulo 3, é apontado o embasamento teórico relacionado especificamente à comportamentos reativos, bem como o contexto da robótica onde surgiu o conceito e associado a aplicação. Enquanto o Capítulo 4 apresenta as principais características dos materiais compostos nos experimentos e como eles estão posicionados, ou melhor utilizados.

Com o Capítulo 5, busca-se ilustrar como as técnicas e a estrutura, anteriormente comentadas nos capítulos passados, foram organizados; compreendendo como foi dada a coleta de dados, a

definição de quais comportamentos foram almejados e como foram executados, a estrutura da rede utilizada, os treinamentos e por fim como as análises serão feitas.

Os últimos Capítulos, 6 e 7, apresentam os resultados obtidos nos testes com as devidas considerações e uma breve conclusão sobre o que foi apresentado (incluindo as perspectivas de melhorias futuras), respectivamente.

Capítulo 2

Machine Learning

2.1 Aprendizado de máquina

Machine Learning e *Deep Learning*, traduzidos como Aprendizado de Máquina e Aprendizado profundo, estão intimamente relacionados, sendo o segundo um tipo específico do primeiro. Essencialmente, *Machine Learning* é conhecido por "uma técnica de estatística aplicada com o uso de computadores para estimar funções e prover intervalos confiáveis ao redor dessas funções.[3]", nas palavras de Ian Goodfellow.

2.1.1 Conceito

De uma maneira geral, *Machine Learning* é o campo de estudo multidisciplinar dedicado ao entendimento de como máquinas podem aprender a tomar decisões com base em informações, inspirado pelo modo como a mente humana funciona[5]. Em outras palavras, estuda como um computador pode aprender a realizar uma tarefa sem que o mesmo seja explicitamente programado para tal. Essa tarefa, geralmente é dada por um propósito geral que é 'aprender' uma função, e o aprendizado consiste no uso de algoritmos que fornecem uma função para seu respectivo conjunto de dados. A partir da obtenção da função é possibilitado ao computador fazer inferências para casos não apresentados, e ainda metrificar quão bom foi o papel da função em estimar respostas.[3]

Tom Mitchell[21] trata a definição de aprendizado de uma maneira sucinta, que fornece base para uma variedade de trabalhos na área, da seguinte forma a partir de uma tradução livre:

> *"Um programa de computador é dito para aprender com a experiência E com relação a uma classe de tarefas T e uma medida de desempenho P, se seu desempenho em tarefas T, como medido por P, melhora a experiência E "*

Dessa forma ele define a capacidade do algoritmo em melhorar o seu desempenho no processo de aprendizado e introduz as métricas para análise dos algoritmos.

Tarefas(T) compreendem as ações finais realizadas pelos algoritmos, e não o aprendizado da

tarefa em si, tal qual o como o sistema deve proceder relativo aos exemplos fornecidos. Performance(P) ou desempenho, compõe um parâmetro que avalia a habilidade dos algoritmos para aprender, e está diretamente ligada a tarefa em si, é metrificada pela acurácia e pela taxa de erro que se traduzem na capacidade do modelo de fornecer uma resposta correta e errada, respectivamente. Por fim, experiência(E) engloba o conjunto de dados utilizados no aprendizado, assim como a forma como estão organizados, e isso reflete diretamente na resposta do algoritmo.

A implementação desse aprendizado de máquina permite resolver problemas que não são fáceis de serem solvidos do ponto de vista da criação de uma lógica computacional e nem são ajustados por funções triviais. Dado essa visão geral, existem peculiaridades em algoritmos que podem ser do tipo Supervisionado ou Não Supervisionado, diferenciando como os dados serão usados e as finalidades práticas (tarefas) as quais se aplicam, os quais são detalhados a seguir.

2.2 Supervisionado e Não Supervisionado

O uso dos dados (Experiência) durante a etapa de treinamento pode resultar em dois principais caminhos, e consequentemente performar tarefas diferentes. Ambas tem sua utilidade e uma diferença muito sutil, e nada formal, mas tradicionalmente definidas em supervisionado e não supervisionado[3].

Com isso o aprendizado dado pela forma Supervisionada apresenta um conjunto de dados que relaciona entradas com saídas, geralmente por rótulos (providos por um instrutor), e tem como objetivo aprender a diferenciar as entradas informando a saída de acordo com a experiência adquirida. Com por exemplo, dado um conjunto de dados que relaciona temperaturas com a sensação térmica de várias pessoas ($Temp$ e $STemp$), para o qual cada elemento $Temp_i$ tem um respectivo elemento em $STemp_i$. Existe uma relação ainda não conhecida entre conjunto de entrada e saída, que caracteriza o aprendizado supervisionado, e é desejado que o computador aprenda estimar esta $'função'$ que relaciona os dois, com base exclusivamente no que foi sugerido; para posteriormente apresentar o valor de sensação térmica $\hat{STemp}$ para a temperatura qualquer $Temp_{teste}$.

Enquanto o treinamento que faz uso de base de dados com várias características, mas com a diferença das informações não necessariamente estarem relacionadas por tarjas, é caracterizado como não supervisionado; e tem como objetivo extrair informações de como os dados randômicos estão distribuídos ou mesmo separa-los em grupos. Considere por exemplo, um conjunto de dados I que contém as informações (sem rótulos) de interações de pessoas em uma rede social, e para o qual é desejado identificar a relação desses grupos e algumas outras informações; dada a incerteza de uma saída definida ou pré-determinada, e o formato dos dados, esse caso constitui um exemplo de aprendizado não-supervisionado

Existem outras formas de implementação que envolvem técnicas mais sofisticadas de aprendizado, como é o caso do aprendizado por reforço, onde existe uma realimentação entre a implementação de *machine learning* e a experiência adquirida, e tem como base o ajuste do desempenho a partir de recompensas ou punições, além de não ter uma base fixa de dados. Isso vem permitindo

aos sistemas que utilizam essa técnica, uma qualidade melhor de implementação e a superação de desafios complexos, como por exemplo em jogos, na qual o algoritmo consegue aprender a jogar com base em várias tentativas[3].

2.2.1 Tarefas e Desempenho

A tarefa está ligada diretamente a como o sistema processa as informações e as suas respostas, e pouco de como o sistema em si irá aprender[3]. Assim, os limites da execução computacional de uma tarefa que são relacionados às barreiras estruturais, custo e a criatividade do programador não cabem dentro do conceito desse ponto, mas apenas do processo de aprendizado, pois só a definição da tarefa é o parâmetro de interesse.

O desempenho não é relacionado a como o aprendizado ocorre, apesar de ser usado no ajuste do aprendizado, e define uma métrica genérica (erro de generalização[22]) que avalia as tarefas aprendidas. Entretanto, é importante destacar que nem sempre é simples definir uma métrica, especialmente em casos de aprendizado não supervisionado, pois a própria é muito particular de cada aplicação e depende dos pesos atribuído aos erros e aos acertos.

2.3 Ajuste, Sobre-Ajuste, Sub-Ajuste e Capacidade

Antes de apresentar formalmente o conceito do ajuste, será introduzido uma exemplo que ilustra de forma simples como o ajuste se dá no caso de uma regressão linear, e por fim é feita uma comparação com os casos de sobre-ajuste e sub-ajuste.

A Regressão Linear ilustra como as métricas de aprendizado estão correlacionadas e fornece um contato sobre o ajuste da capacidade do algoritmo de prever os dados relativo a situação abordada[3]. Nesse problema, temos um algoritmo que faz uso de um conjunto real finito de valores para $'aprender'$ como estimar valores, tal como uma função linear. Expresso por:

$$\hat{y} = \omega^{\tau} x \tag{2.1}$$

Na qual $\hat{y}$ é um valor estimado pelo modelo e x o valor de entrada, e isso constitui a tarefa que será realizada.

ω por sua vez é o vetor com os valores de peso que cada parâmetro tem, e ele é obtido via treinamento, a partir de um conjunto finito de entradas selecionadas. Esse ajuste de ω ocorre através de correções calculadas para diminuir o valor do erro quadrático médio, de forma que ao final da execução do algoritmo é desejado ter o melhor vetor de pesos que ajustam saída e entrada. Logo, se as informações estão relacionadas linearmente como o da função dada acima, o algoritmo será capaz de estimar os pesos da função, e de posse desses valores é possível obter um sistema que simula o real, através de estimativas.

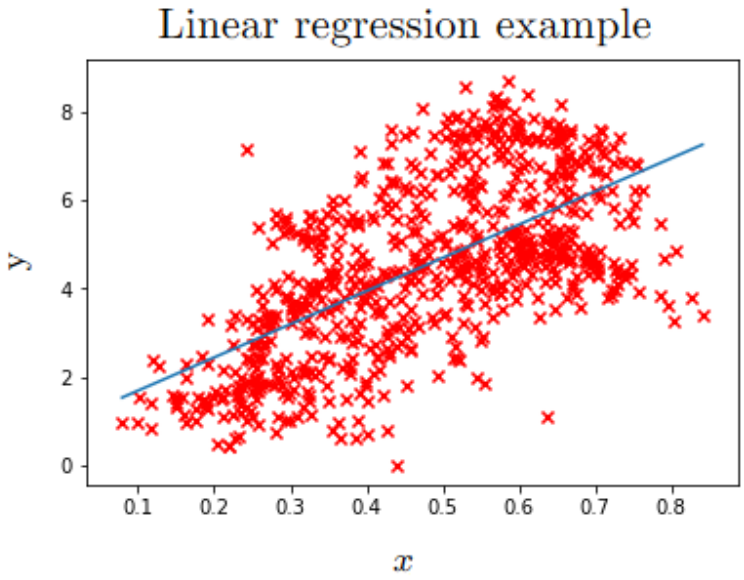
O erro quadrático médio é entendido como variável que mensura a performance por meio de um conjunto de teste separado do conjunto total x. Esse erro é calculado durante o treinamento

para melhor ajustar os pesos, e durante os testes para comparação entre o estimado pelo sistema treinado e o esperado, como na fórmula seguinte:

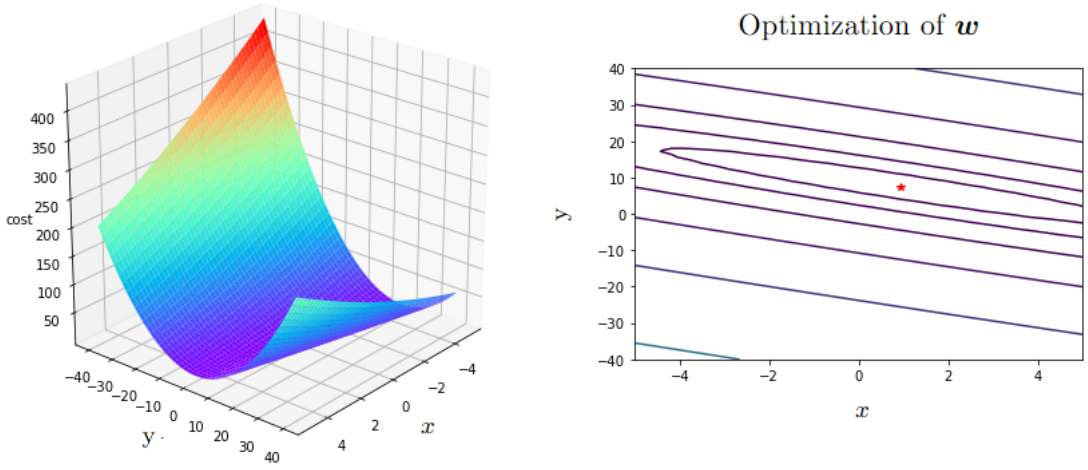
$$MSE = \frac{1}{m}\sum_{i}(\hat{y} - y_{teste})^2 \tag{2.2}$$

O propósito do algoritmo iterativo de aprendizado de máquina é minimizar o valor desse erro, de forma $\hat{y} = y_{teste}$

Nesse caso, a regressão linear compreende um algoritmo que fazem uso do uma base de dados fixa, a qual não interage diretamente com o ambiente enquanto processada. E como resultado é desejado obter as curvas expressas na Figura 2.1 .



(a) exemplo de regressão linear para um conjunto de dados



(b) curva de nível do custo pra o treinamento e ponto de menor custo respectivo.

Figura 2.1: Regressão Linear [1].

Existe uma forma de calcular os pesos da função de regressão de forma não iterativa, dessa forma é desejado realizar um único cálculo que encontra o ponto de mínimo na função quadrática da figura 2.1(b). Entretanto, esse meio não será o foco do apresentado pois a complexidade do cálculo aumenta conforme o conjunto de dados aumentarem, e com isso o custo computacional do método iterativo se torna mais palpável. Outro detalhe que pode não ser evidente e deve ser entendido, é o de que a função de primeira ordem definido pela equação 2.1 e exemplificado pela figura 2.1(a), pode ser ajustada pela adição de um termo constante para permitir a translação da curva ao longo do eixo y.

Seguindo o proposto, serão apresentado conceitos relacionados ao ajuste após o treinamento, começando pela generalização que é a capacidade de um algoritmo realizar uma boa predição para dados de entrada que não foram fornecidos durante treinamento. Nesse contexto são considerados o erro de validação que é utilizado para ajuste do modelo (otimização de ω) e o erro de teste que é medido após o treinamento e metrifica o conceito de generalização.

Durante implementação do algoritmo algumas características do modelo, relacionadas ao erro de treinamento e a ordem do ajuste escolhido, resultam em consequências diferentes em reflexo de um sobre-ajuste e um sub-ajuste. Assim, para um dado modelo de aprendizado, diz-se que um modelo está sub-ajustado quando o valor do erro de treinamento é grande; e define-se que o modelo está sobre-ajustado quando o mesmo apresenta um erro de treinamento pequeno porém com ordem (complexidade) bem maior que realmente necessária, e que pode incrementar erros para estimativa de dados de teste, ou seja, quando o erro de treinamento e de teste tem grandes diferenças. Em resumo, é sempre interessante manter o equilíbrio entre a hipótese a ser aprendida e o erro obtido, resultando em um ajuste aceitável e que melhor descreva o problema [3].

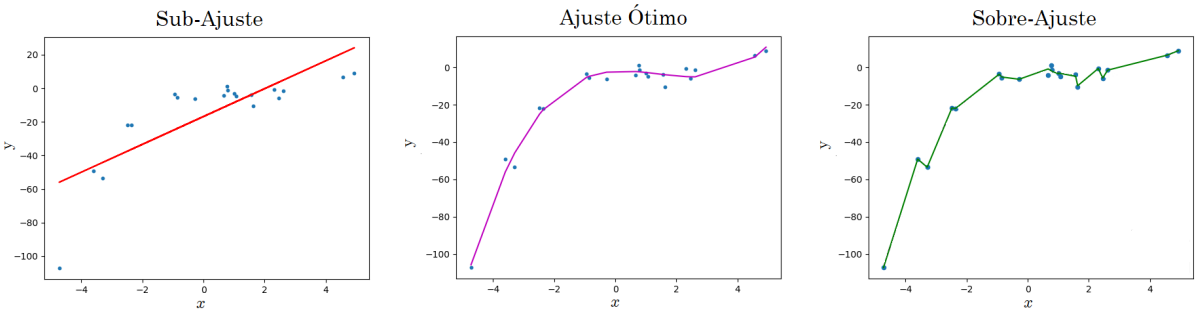


Figura 2.2: Ajustes para regressão polinomial [2].

Esse equilíbrio é dado pelo ajuste da capacidade do algoritmo, que nesse caso é a ordem da função, e é o reflexo de uma boa escolha de hipótese. Na Figura 2.2, a primeira curva revela o ajuste feito por uma regressão de ordem 1 que não adapta bem aos dados, pois falta complexidade nessa hipótese para solucionar o problema para esses dados. Na terceira curva, a complexidade da hipótese supera as características dos dados e assim fornece uma curva que cobre todos os dados de treinamento, mas que não garante boa generalização para valores intermediários. E por fim, para o conjunto de amostras apresentadas, é evidente que a função expressa na curva 2 é suficiente para obter um bom resultado, mas obter esse ajuste nem sempre é simples pois é necessário encontrar a hipótese que melhor equilibra o erro de treinamento em relação ao erro de generalização.

Cada uma das curvas descritas na última figura são esboçadas para uma equação de tamanho fixo, por isso são definidas como modelos parametrizados pois tiveram a equação definida antes do consumo dos dados. Assim, o oposto são modelos não parametrizados, ou seja, que não são fixos e permitem que o ajuste de hipóteses seja feito com base no menor erro de generalização, e por consequente o melhor ajuste de capacidade. A figura 2.3 ilustra a curva dos erro de treinamento e de generalização(teste) relacionado à capacidade, e também mostra que existe pelo menos uma solução que melhor se adapta ao conjunto de teste e de treinamento, mas essa solução pode não ser a que foi obtida com menor erro de treinamento e nem com uma maior capacidade. Dessa forma, a ordem da equação deve ser ajustada para o problema que envolve os dados, e deve ser encontrado o menor Erro de Bayes, definido pela diferença entre o predição do algoritmo e a distribuição real [3].

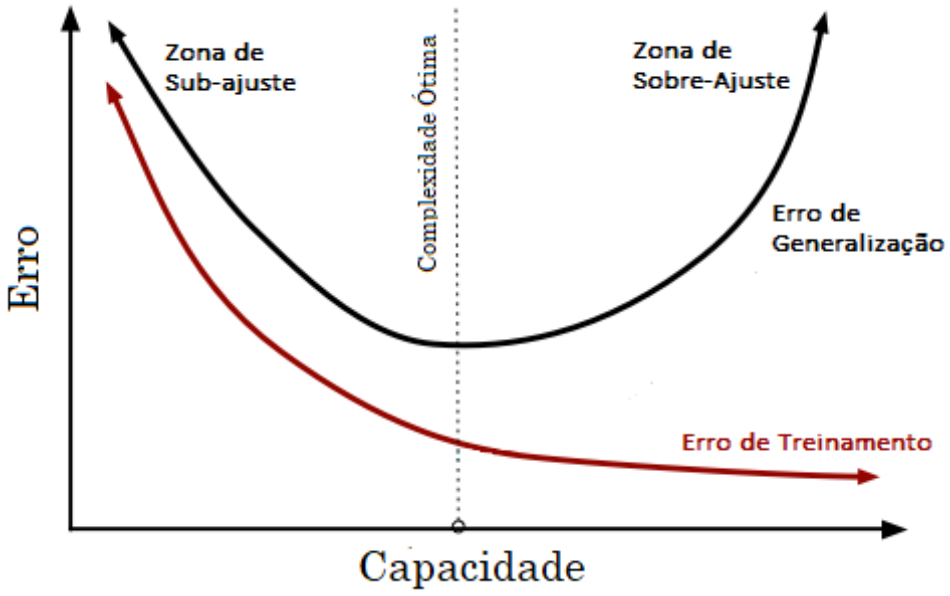


Figura 2.3: Ajuste de capacidade vs Erro [3].

2.4 Aplicações

Os algoritmos mais comuns de problemas solucionados com o uso de *machine learning* performam tarefas como regressão, classificação e segmentação. A regressão consiste em encontrar um número que representa um curva dado um conjunto de pontos, tal que esse número seja uma variável que integra um modelo que descreve a dispersão, e essa 'curva' irá depender da ordem dos dados apresentados durante o treinamento. Uma aplicação dessa técnica é a estimativa de valor imobiliário. A classificação é geralmente utilizada com a finalidade de separar dados em subconjuntos, fornecendo uma categoria como saída, e permite ainda uma variação para detecção de anomalias que se traduz em catalogar ocorrências atípicas, enquanto a segmentação permite catalogar dados em conjuntos semelhantes não conhecidos. Ambas são muito semelhantes em resultados, no sentido que diferenciam os dados, porém diferem na forma que são implementa-

das uma vez que uma se dá de forma supervisionada (classificação) e a outra de maneira não supervisionada (segmentação).

Vários outros algoritmos como de transcrição de texto por imagens ou áudio, detecção de anomalias, eliminação de ruídos, amostragem e até estimativa de função de densidade de dados, são exemplos mais sofisticados do que já é feito atualmente com uso de *machine learning*. Esses casos ilustram como problemas computacionalmente dispendiosos podem ser resolvidos, fazendo valer o uso da técnica de aprendizado de máquina com princípios de base comum, mas adaptados à cada situação.

2.5 Deep Learning

Deep Learning é um ramo da grande área chamada *Machine Learning*, que em tradução livre significa aprendizado profundo, e é assim nomeado devido envolver várias camadas de processamento, além de ter suas origens inspiradas por neurônios biológicos e na forma como eles interagem. Esse ramo também leva a denominação de 'redes neurais', devido as composição de várias unidades de processamento em cada camada e das relações estabelecidas entre elas, novamente inspirado pelo o que se conhece do funcionamento de uma rede neuronal.[3] As camadas estão sobrepostas de forma que a saída de uma é a entrada da próxima, com os dados podem fluindo em um ou mais sentidos a depender da aplicação[3], e foram criadas com a proposta de resolver algumas das limitações que as implementações tradicionais falham, principalmente quando envolve problemas não lineares (XOR[23]) ou com diversas dimensões, o que também acarreta em um ganho de complexidade. Essa composição fornece a possibilidade de agrupar diferentes funções em uma mesma implementação, onde a união dos fatores é a ideia por trás disso, na qual características são potencializadas em cada camada da hierarquia e podem ser, ou não, realimentadas com *feedback* para os nós. Com destaque pra informação de que as camadas que estão nas bordas, caso da camada de entrada e de saída, são as únicas a serem avaliadas durante um treinamento.

2.5.1 Inspiração Fisiológica

Com os avanços nos estudos em como o sistema neurológico humano funciona, foi possível mapear a estrutura e um fluxo de informação dado por correntes elétricas, apresentado pela estrutura de um neurônio biológico dado na figura 2.4 . Os estímulos elétricos, que causados pelas correntes elétricas, são provenientes de outros neurônios e compõem 'dados' para integração espaço/temporal feita por uma membrana que possui memória sináptica, que por fim ativa um impulso na saída quando um limiar de estímulos é atingido.

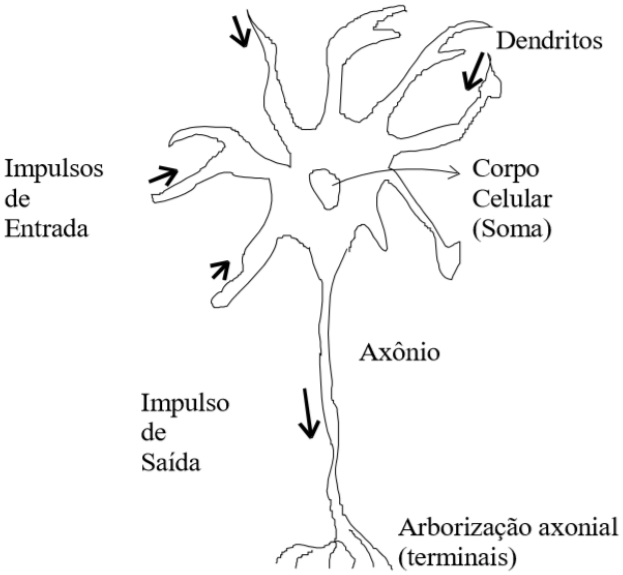


Figura 2.4: Neurônio Biológico [4].

2.5.2 Neurônio Artificial

Proposto inicialmente por McCulloch e Pitts em 1943 [24], com a definição matemática de um neurônio booleano e posteriormente melhor adaptado com o modelo do perceptron (Rosenblatt [25][26]), foram consolidados os primeiros passos para a definição de como um neurônio é aplicado em computador e um vislumbre do que um modelo genérico que a princípio poderia aprender qualquer coisa, mas que na prática teve diversas limitações.

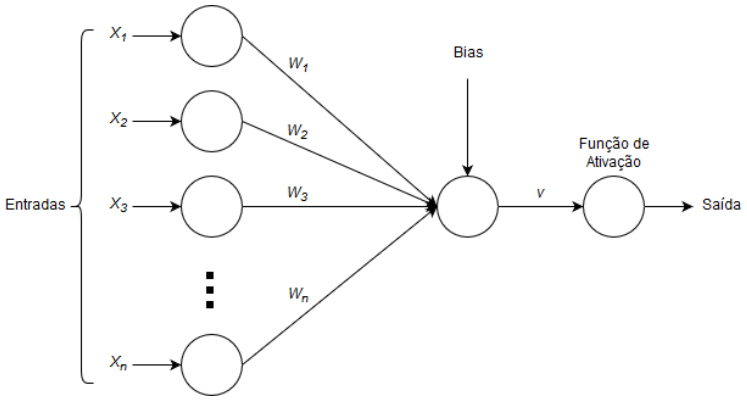


Figura 2.5: Representação de um neurônio artificial ou Perceptron [5].

Só depois de muitos anos, foi mostrado por Rumelhart, Hinton e Williams, em 1986 [27] , um outro modelo que realmente poderia resolver problemas não lineares e envolvia a *backpropagation*, que permite o ajuste dos pesos de cada neurônio. Esta implementação envolve a composição de vários neurônios em pelo menos duas camada, onde a entrada da 2ª camada é alimentada por todas as informações provenientes dos neurônios imediatamente antecedentes e assim formam uma rede neural artificial.

A fórmula que define a saída v da figura 2.5, é matematicamente expressa por:

$$y = f(v) = \sum_{i=1}^{m} \omega_i x_i + b \tag{2.3}$$

Com m igual a quantidade de entradas do neurônio e b dado pelo valor de *bias*.

Apesar de v ser dado como saída nessa ultima equação, o verdadeiro valor de saída é fornecido por y que é o resultados de uma função de ativação aplicada à v. Enquanto, ω é um vetor de pesos que é ajustado a cada iteração do algoritmo de aprendizado, e esse ajuste ocorre pelo seguinte cálculo:

$$\omega(n+1) = \omega(n) + \eta\left[d(n) - y(n)\right] x(n) \tag{2.4}$$

A diferença entre $d(n) - y(n)$ representa o erro entre o valor desejado d e o estimado y. E η simboliza um parâmetro que é entendido como a taxa de aprendizado, que é compreendido por uma faixa de valores que vai de $0 < \eta \leq 1$ e pode ser variado durante o treinamento, de acordo com as consideração estudadas por Lippmann (1987)[28].

2.5.3 Redes Neurais Profundas

Cada neurônio, como é chamado o nó de uma rede neural, atribui um peso à tarefa que está realizando, e cada camada pode conter uma quantidade grande de nós. As pontes entre os nós são chamadas de *'edges'*, análoga as sinapses no cérebro animal, e que tem pesos atribuídos. Geralmente cada camada tem uma finalidade no trato com os dados e os pesos são definidos relativos ao propósito, portanto cada neurônio aprende como tratar o dado da melhor maneira e o conjunto a fornecer o melhor resultado[5].

Ter muitos nós eleva a quantidade de operações que devem ser feitas paralelamente, por conseguinte o custo computacional, e requer um *hadware* específico; por esse motivo as GPU's são usadas para processamento de técnicas de aprendizado de máquina dada o seu bom desempenho para realizar treinamentos de redes neurais em um conjunto de dados extenso, de maneira simultânea/paralela [22]. Durante um bom tempo, o problema de usar redes neurais estava relacionado ao hardware da época não concluir o treinamento em tempo hábil, mas tudo isso tornou-se possível com a aparição de soluções para processamento mais 'rápido' e também de unidade de processamento dedicado (TPU's). Com esses avanços, os estudo em *deep learning* puderam ser aprofundados e o estado da arte encontrasse no ponto que as implementações tornaram-se populares, efetivas e superam a capacidade do ser humano, indo desde do reconhecimento de objetos dentro de imagens, passando pelo reconhecimento de voz e até ganhar uma partida de Go [29].

A figura a seguir mostra um exemplo de topologia de rede que faz uso do perceptron em várias camadas.

A camada que recebe os dados é chamada de 'camada de entrada' e a que fornece a saída é denominada 'camada de saída', enquanto as demais são ditas 'camadas ocultas'. Os dados fluem

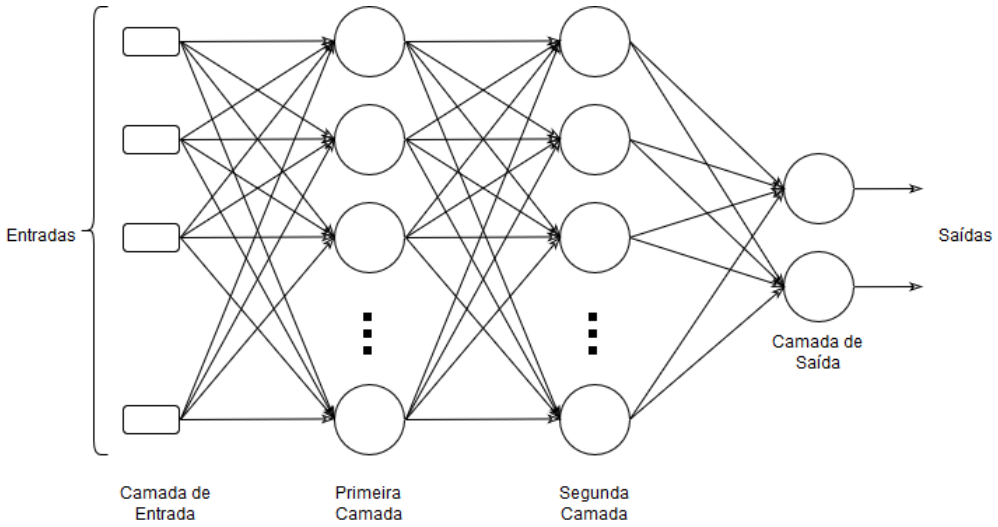


Figura 2.6: Exemplo de arquitetura multicamadas de perceptrons (*Multilayer Perceptron*) [5].

na direção da camada de saída, enquanto a informação de erro que proporciona o ajuste flui no sentido contrário à saída, ou seja, rumo à camada de entrada [5].

Em um rápido paralelo com os tipos comuns de arquiteturas de redes neurais profundas temos as redes *feed-forward* que são compostas de neurônios que descrevem funções não lineares dos dados para camada seguinte, e portanto mais simples. Enquanto que redes recorrentes são mais realistas quando comparada com neurônios biológicos pois apresentam complexidade estrutural superior e conseguem descrever mais fluxos de caminho dos dados. E por fim as rede conectadas simetricamente, que são um tipo de rede recorrente que tem pesos iguais para todos as conexões entre os nós [3].

2.5.3.1 Redes Neurais Convolucionais

Convolutional networks, também conhecidas como redes neurais convolucionais são um tipo especializado de rede neural para processamento de dados com topologia espacial, onde esses dados podem estar em duas dimensões ou mais, contanto que os dados estejam amostradas em um intervalo regular. Esses redes tem sua origem associada, inicialmente, ao trabalho de Hubel e Wiesel[3][30], que envolve a observação da atividade de neuronal de gatos, entretanto, atualmente são muito mais relacionadas aos trabalhos de LeCun[31], quando a aplicação mostrou-se factível e útil; e são assim denominadas devido à presença da operação de convolução dentre as suas camadas.

Convolução nesse contexto não compreende as mesmas operações utilizadas para tempo contínuo da engenharia ou matemática, e sim um processo discretizado de fazer convolução, descrita pela seguinte formula:

$$(p * q)(t) = \sum_{\tau=-\infty}^{\infty} p(\tau)q(t-\tau) \tag{2.5}$$

que é aplicada no domínio espacial em que se encontram as imagens.

Essa operação compreende um processo de filtragem da imagem com o objetivo de extrair informações do mesmo, e é realizada pela aplicação de um *kernel* em cada pixel da imagem[3]. Exemplos clássicos de operações de convolução no tempo são os filtros de realce e suavização de imagens(filtros de primeira ordem) e filtro Laplaciano (filtro de segunda ordem). Computacionalmente, a convolução feita em computador segue a fórmula dada por:

$$I'(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n) \tag{2.6}$$

Na fórmula dada acima, I representa a imagem com $m \times n$ pixeis, sendo utilizada para aplicar um filtro K que vai fornecer uma imagem filtrada I'.

Essas redes são especialmente utilizada para classificação de padrões e formas em imagens, dada sua robustez em lidar com translação, inclinação e escala. E essa 'habilidade' é provida principalmente pela operação de convolução, que elucida as características da imagem de entrada, junto da operação de sub-amostragem que reduz as dimensões dos dados. Essa redução sem a necessária perda de características dos dados foi o principal possibilitador do sucesso dessas redes, pois possibilita um treinamento em tempo hábil do qual uma rede neural artificial feita apenas de perceptrons não consegue competir, além da complexidade associada ao ajuste de erro de uma quantidade grande de neurônios[22].

Na prática, uma camada convolucional atua com vários *kernels* que evidenciam as características da imagem e são ajustados durante treinamento para as várias imagens que possuírem formas em comum. Os *kernels* são compostos por $m \times n$ neurônios que correspondem as dimensões dos p filtros da camada, e nesse caso os neurônios são limitados por um campo receptivo que é relacionado ao tamanho do *kernel*. A camada de sub-amostragem permite a redução do tamanho dos dados através do uso de um *kernel*, da mesma forma que a convolução, mas aplicada apenas em pontos específicos determinados pelo passo.

Um bom exemplo é o da implementação feita por Yann LeCun, chamada de LeNet-5, e apresentada na figura 2.7.

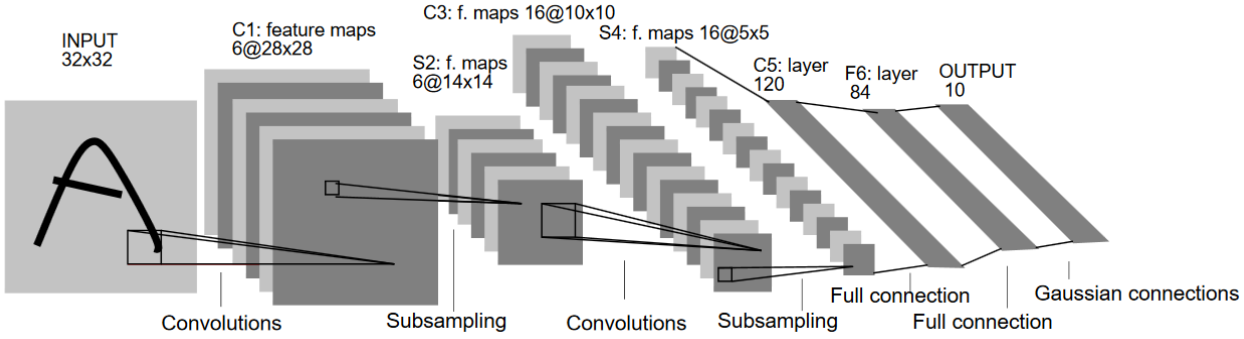


Figura 2.7: Arquitetura LeNet-5 [6].

Para a imagem de entrada com tamanho 32×32, a primeira camada tem 6 filtros com dimensões

28×28 apresentados de um *kernel* de 5×5 pixeis, logo a mesma terá $5 \times 5 \times 1 + 1(bias)$ parâmetros de cada filtro para aprender, somando um total de 156 valores. Após a convolução se uma sub-amostragem for feita com passo de 2x2 na imagem, sobre os filtros a dimensão de cada filtro será reduzida a metade saindo de $6 \times 28 \times 28$ para $6 \times 14 \times 14$, e isso reduzirá os valores que são passados para as camadas posteriores o que consequentemente irá reduzir os parâmetros ajustados pelo aprendizado.[6][32]

Contudo, se uma rede neural artificial é composta de vários neurônios, sem camadas de sub-amostragem ou de convolução, receber como entrada a mesma informação descrita na figura 2.7, a primeira camada necessitaria de $32 \times 32 \times 1$ neurônios totalmente conectados para fornecer 1 filtro com dimensões 32×32, e para todos os 6 filtros são necessários $(32 \times 32 \times 6 + 32 \times 32(bias)) \times (32 \times 32 \times 1) = 7.340.032$ parâmetros a serem ajustados em apenas um laço de execução[6]. Em resumo, essas redes geralmente são construídas com o uso de várias camadas de convolução, embora apenas uma seja necessária para definição de rede convolucional, e esses valores ilustram a inviabilidade técnica de não utilizar camadas de convolução quando deseja-se treinar uma rede neural para dados de entrada muito grandes, principalmente quando se trata de imagem, vídeo ou mesmo áudio.

Capítulo 3

Fundamentação de Comportamentos Reativos

3.1 Robótica

O termo robô, popularizado por Karel Capek em 1921, refere-se ao sistema autônomo que interage com o ambiente, consegue 'sentir' através de sensores e realizar uma tarefa para cumprir os objetivos determinados. Esse conceito refere-se a uma realização no mundo físico que antes estava limitada a mecanismos autônomos e atualmente tem suas bases movidas principalmente pela computação e pela mecânica, o que permitiu uma expansão e flexibilidade nas capacidades dessa máquina realizar tarefas.[15]

Ao robô é permitido 'perceber' um ambiente com base no conjunto de *sensores* que o ajudam a perceber o meio, tais como: ultrassom, câmeras, LIDAR, termômetro. Assim, como também é permitido atuar no meio e modifica-lo com a ajuda de *atuadores*, que geralmente são motores elétricos, ou conjuntos destes[15]. E uma vez que existe, percebe e atua, é dado o mínimo para que esta máquina possa solucionar alguns desafios por meio de seu programa, que é o controlador ou cérebro da implementação.

O programa faz o papel de orquestrar o conjunto de ações e informações do sistema, sempre com o objetivo de automatizar um procedimento e até melhorar o resultado deste. Um robô autônomo não é controlado por uma pessoa e deve tomar decisões com base no que foi programado,ou seja, deve ser auto-controlado pelas especificações do seu programa.

Assim, a Robótica é um campo que teve suas definições melhoradas na evolução do tempo, mas segundo Mataric[8] pode ser definida como a área que estuda as definições conceituais de um robô, que abrangem a concretização da automação da máquina até a sua finalidade. Dessa forma, a robótica é composta de 3 principais áreas de estudo, conhecidas como: Controle, Ciberciência e Inteligência Artificial; contudo, hoje tem a seus princípios mais ligados a teoria de controle e IA, e sem necessariamente eliminar as influências proporcionadas pela mecânica.

A teoria de controle estabelece bases de nível mais baixo de controle e consiste em uma funda-

mentação matemática dos princípios que regem a automação de sistemas, esta teve uma origem do estudo de modelos mecânicos e posteriormente incorporou princípios elétricos/eletrônicos[8]. A 'cyberciência', que caiu em desuso em prol de sistemas biomiméticos, está relacionada ao estudo de processos biológicos, principalmente seus comportamentos e sua integração, e a replicação destes em sistemas robóticos. Enquanto a Inteligência Artificial tem objetivo de prover 'inteligência' a uma máquina, que consiste no planejamento e tomada de decisão.

A primeira e mais conhecida implementação de comportamentos bioinspirados foi feita por William Grey Walter, que apresentou em seu trabalho *"A Machine That Learns"*[17] em 1950, descrita por ele como uma pequena implementação da vida, e mostrara que a composição de reações a reflexos apresentaria um comportamento parecido com o de um animal. Grey apresentou um robô parecido com uma tartaruga (Figura 3.1), com controle feito de modo reativo, e era capaz de seguir, encontrar e fugir da luz, além de desviar de objetos.

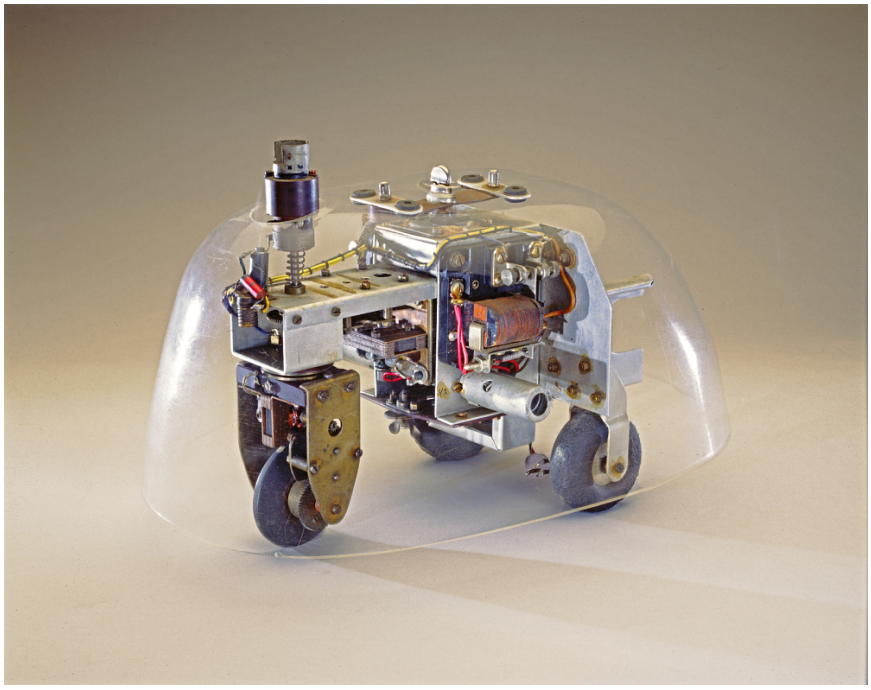


Figura 3.1: Tartaruga de Grey Walter [7]

Esse robô possui sensores de luminosidade, fotocélulas, e sensores de colisão que são integrados a um circuito elétrico analógico que 'processa' os sinais fornecidos e aciona os motores. Esse circuito, atua como o controlador que vai simular reações aos "reflexos"providos pelos sensores e tenta mimetizar comportamentos bioinspirados.

Uma apresentação mais enxuta de como essa implementação robótica pode ser feita, é realizada pela ótica da teoria de controle[8] já comentada e aqui ilustrada pela figura 3.2, onde o sistema que controla recebe uma variável de referência que nesse caso é o índice de luminosidade pela diferença do valor fornecido pelo sensor, e responde com a intensidade que foi programado para o fazer.

A inteligência artificial, que teve seu estudo impulsionado pela aplicação e problemas de con-

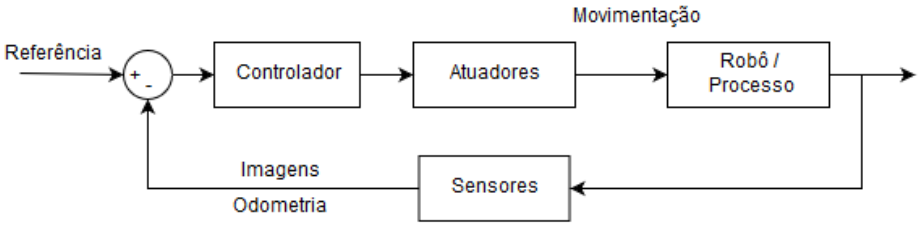


Figura 3.2: Exemplo de Controlador aplicado na robótica

texto robótico no qual também é integrante, auxilia o processo de concepção de sistemas inteligentes e trás para o jogo os mecanismos necessários para habilitar robôs a planejar e resolver problemas de maneira inteligente. As aplicações de IA estão focadas na forma como o planejamento ou controle é realizado, para abordar e solucionar problemas conhecidos, e é composto de 7 principais áreas[33] de estudo: representação do conhecimento, entendimento de linguagem natural, aprendizado, planejamento/solução de problemas, inferência, busca e visão computacional.

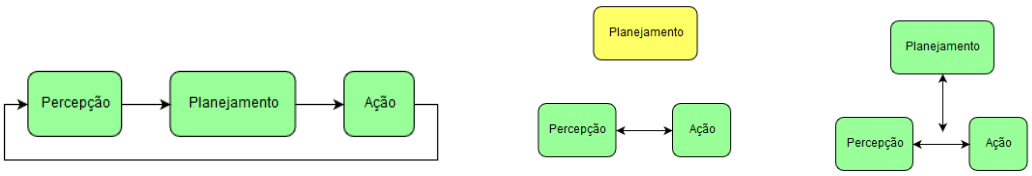
Resolver problemas de robótica envolve a escolha de técnica e das suposições para definição de uma abordagem, o que se resume ao conceito de paradigma. Em especial no caso da robótica, são destacados 3 principais tipos de controle robótico: controle reativo, híbrido e hierárquico.

Esses paradigmas são classificados, por Murphy[15][8], através da forma como é realizado a percepção do ambiente (SENSE), o planejamento de atuação (PLAN) e a ação em si (ACT). O primeiro constitui a tarefa de obter e disponibilizar, de forma útil, as informações dos sensores. O segundo, trata de obter as informações que foram disponibilizadas pelo sensores ou mesmo baseadas no conhecimento do ambiente de trabalho, e com isso indicar ações que devem ser feitas. O último e não menos importante, está relacionado a execução da ação e os comandos necessários para correta atuação.

Por exemplo, considere uma versão atualizada do robô tratado na figura 3.1 (com sensor de luminosidade, motores e um microcontrolador programável). Para esse exemplo a conversão o dado fornecido pelo sensor de luminosidade em uma variável matemática que seja útil para a lógica do programa é dada como a percepção do ambiente, enquanto a escolha da direção que o robô será orientado, com base no que foi percebido, é dado pelo planejamento. A ação nesse caso é a velocidade, o ângulo e o sentido de rotação de cada motor.

A figura 3.3, ilustra como cada paradigma robótico está relacionado a percepção do ambiente, planejamento e ação.

Note que na figura 3.3(b), a função de planejamento em amarelo não está conectada ao resto do conjunto, como na figura 3.3(c), isso não significa a ausência de participação desse bloco e tem objetivo de simbolizar apenas que não é tão determinante no processo. Esse paradigma reativo consiste na principal direção tomada nesse trabalho e será melhor descrito na seção seguinte.



(a) *hierárquico* (b) *reativo* (c) *híbrido*

Figura 3.3: Paradigmas da Robótica: uma representação gráfica de como a percepção, planejamento e ação estão relacionados em cada caso [8].

3.2 Comportamento Reativos

Os comportamentos ou *behaviors* estão relacionados ao que é observado como resultado de uma atuação, ou como visto, o ACT. Reativo é definido como aquilo que provoca uma reação ou reage, uma resposta oposta a outra, ação produzida em objeção a um estímulo exterior. Assim, a composição etimológica desses dois conceitos sugere que as ações são resultados de estímulos, ou seja, reflexos. Esse paradigma teve suas origens em 1980 e foi ocasionada principalmente pela insatisfação com o modelo hierárquico [15], e tem ampla relação a etologia (campo da ciência dedicado ao estudo comportamental de animais).

Comportamentos reativos são comumente usados para realizar controle bioinspirado, como no caso de um robô que mimetiza insetos, e consiste de um comportamento diretamente relacionado aos sensores e muito similar a reflexos, com tempo de resposta igualmente curto[8]. A agilidade de atuação é o resultado de uma lógica construída de uma malha extensa de atuação baseada apenas nas informações providas pelos sensores (não são utilizados mapas nem representações de ambiente), e podem fazer uso de tabelas de consulta predefinidas para o comportamento.

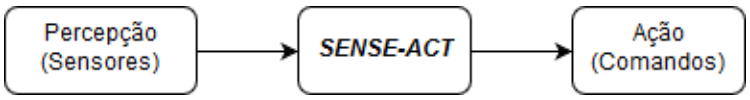


Figura 3.4: Paradigma Reativo - Fluxograma Sense-Act

A figura 3.4 ilustra, por meio do fluxograma, a forma como o controle reativo aplicado pelo bloco SENSE-ACT. Esse bloco, liga diretamente as informações de entrada dos sensores às soluções providas no espaço de respostas (comando de cada atuador), tal como uma função de transferência.

As tabelas de consulta fornecem uma conjunto de soluções para as variadas combinações de dados dos sensores, e permite obter uma solução rápida sem necessariamente planejar. Um exemplo superficial de como $lookup - tables$ funcionam é o de mapas de injeção de combustível, presente no sistema de controle de motores à combustão, que é alimentado pelas informações de rotação, temperatura, nível de CO_2 no ar e apresenta uma saída dada pela tensão pulsada (PWM) aplicada nos bicos injetores para a quantidade respectiva sugerida pela tabela.

O sistema reativo, quando não é construído por $lookup - tables$, são construídas coleções de regras paralelas que contém os reflexos desejados[15]. As regras não envolvem um planejamento

complexo da solução e tem o mesmo objetivo de agilidade de resposta que as tabelas comentadas. Entretanto, nem todos os comportamentos programados são inteiramente reativos, o que é evidente quando escolhas de resposta precisam ser feitas, e a solução pra esse problema vem por meio da composição por arbitragem e/ou fusão de comandos.

Dessa maneira dois ou mais comandos podem ser dados como possíveis reações, e uma questão crucial que pode surgir é: "qual reação é a correta?". Para isso, existem dois meios genéricos pelos quais uma solução pode ser alcançada, o primeiro consiste na arbitragem de escolha e fornece a decisão de apenas um dos comandos (normalmente comandos de maior prioridade), enquanto o segundo consiste na composição simples de comandos. A composição, não deve ter uma grande complexidade e muito menos possuir sugestão de comandos por algo parecido com uma máquina de estados.

A programação geralmente é a principal barreira na implementação de comportamentos reativos, mas isso pode ser contornado pela estruturação de premissas para solucionar pequenas tarefas[15]; premissas estas que são paralelas, concorrentes e associadas a reações específicas. Este método apresenta estrutura vertical, sugerida pela etologia, na qual as regras são compostas incrementalmente e podem estar relacionadas em caráter inibitório ou supressor. A simplicidade dos reflexos seja ele esperado ou emergente, no sentido de implementação, se dá então pela regras (premissas) e pela forma de escolher os comandos.

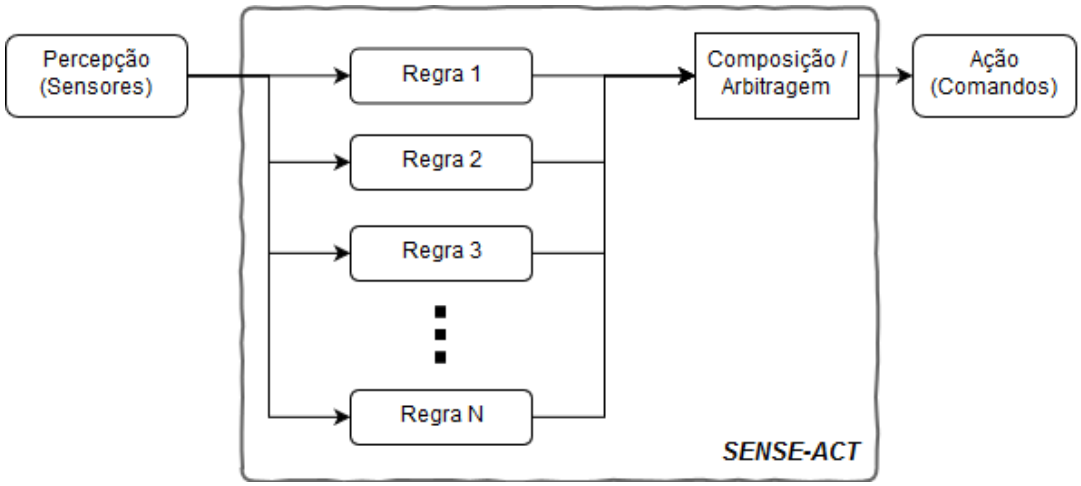


Figura 3.5: Um exemplo genérico de diagrama Sense-Act associado ao paradigma reativo

A figura 3.5, ilustra como cada conceito apresentado se encaixa do contexto robótico reativo, em especial na função de sentir e reagir/atuar. Assim as diversas regras ou premissas se encontram paralelas, e são posteriormente compostas ou selecionados.

Por fim, é de grande importância destacar que além da dificuldade de programação desses comportamentos, um outro problema relacionado a esses sistemas é o multiprocessamento de todas as regras, no que envolve a capacidade de monitorar e executar os comandos, o que pode ser uma barreira quando não se tem um hardware embarcado com capacidade suficiente de computação, seja para quantidade de regras ou para dimensão dos dados. Por exemplo, um comportamento ou reflexo que seja baseado em imagem é uma tarefa que é muito complexa de ser programada em

poucos passos, ainda mais se necessitar de tempo de resposta muito pequeno ou se o hardware é muito simples (microcontroladores). Outro aspecto está relacionado a qualidade da solução de tarefas, pois esse paradigma não é o mais indicado, e muito menos passível de prova matemática da performance do tarefa [15].

Com o intuito de sumarizar o contorno mais elegante das limitações do controle reativo[8] uma implementação desse tipo de sistema não deve possuir ou executar:

- Memória: armazenar resultados de atuações anteriores pra posterior uso e tomada de decisão.
- Auto-instrução: Aprender com o meio ou com a observação de padrões.
- Modelos de ambiente: Utilizar mapas físicos, representações virtuais do ambiente ou ter percepção total do ambiente.
- Estados Mínimos: Possuir estados (Comportamentos) dependentes tal como em uma máquina de estados.

Com as ressalvas de que o ponto vista deve ser egocêntrico a cada nova iteração, e que o diagrama de máquina de estados pode ser usado para esboçar como o conjunto de comportamentos será realizado.

Apesar de todo o esforço em categorizar corretamente o que é um comportamento reativo, esse muitas vezes se vê indissociável do paradigma híbrido, pois a própria composição de regras pode ser vista como uma forma de planejamento, e é mais complexo ainda quando envolve aplicações robóticas de IA. No caso de implementação de Inteligência artificial, a discussão é sobre questões muito mais amplas como: como as regras foram compostas? existe dependência entre as saídas? existe uma referência para tomada de decisão? ocorre planejamento?; e nem sempre todas essas questões são facilmente respondidas pois mudam a caráter da solução robótica.

Capítulo 4

Aparato Experimental

Nesse capítulo são descritos as principais ferramentas que foram utilizadas no decorrer do trabalho, seja para coleta de dados, controle de trajetória do robô ou mesmo no treinamento, assim como as principais características de *hardware* e *software* dos computadores envolvidos.

4.1 Pioneer 3AT

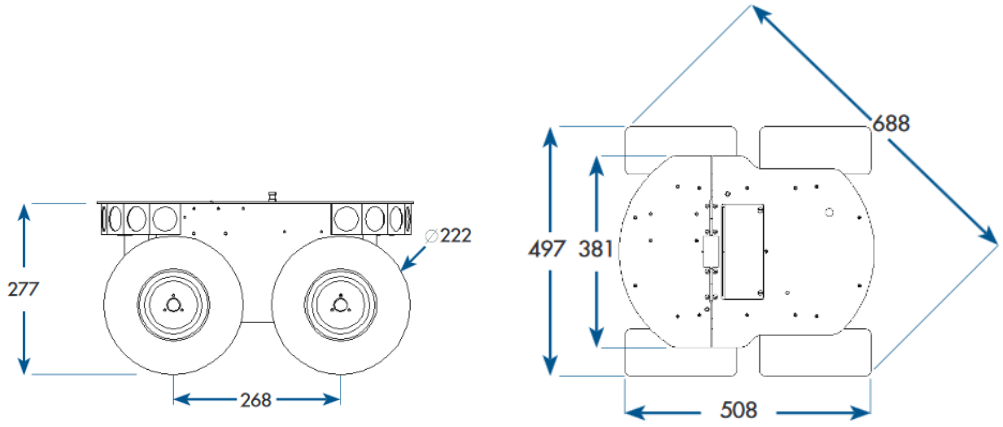
Pioneer 3AT é uma plataforma robótica desenvolvida pela Mobile Robots[10], utilizado amplamente para exploração, pesquisa e prototipagem de aplicações relacionadas mapeamento e monitoramento de espaços, navegação, manipulação e prototipagem de comportamentos robóticos. Essa plataforma dispõe 4 rodas, e 2 motores operando o par de rodas (cada lateral do robô), não holonômico, com um computador embarcado e um conjunto de baterias, além de sensores como: laser, câmera, ultrassom e odometria. O computador embarcado faz uso do Linux Ubuntu, o qual funciona normalmente como um computador, dispondo de saída de vídeo, portas USB e *Ethernet*, entrada para teclado e para mouse.

Figura 4.1: Pioneer 3AT - Robô utilizado para coleta de dados e implementação de comportamentos reativos [9].

A tabela e as figuras a seguir, sumarizam as principais características[34] dessa plataforma:

Tabela 4.1: Principais características do robô (Pioneer3AT)

Peso	12 kg
Câmera	Canon VC-C50i
Sensores	Ultrassom (Dianteiro e Traseiro)
Esterçamento	Deslizante
Raio de Curva	0
Velocidade de Translação (máxima)	$0.7m/s$
Velocidade de rotação (máxima)	$140\circ/s$



(a) vista lateral (b) vista superior

Figura 4.2: Representação gráfica das dimensões do robô. Fornecido pelo manual da fabricante (dimensões em milímetros) [10].

4.1.1 Cinemática de movimentação

O robô consegue desenvolver movimentos segundo um esterçamento $'Skid'$, como descrito na figura 4.3, e consiste de uma cinemática de 'deslizamento' a qual é possível graças ao controle feito com base na diferença entre as velocidades aplicadas nas rodas da direita e da esquerda. Isso permite ao robô mais liberdade na movimentação, como por exemplo realizar rotações sem necessariamente sair do lugar dado um ângulo esterçamento.

No plano apresentado na figura, o robô tem velocidade definidas em $X^{'}$ e $Y^{'}$ (V_x e V_y) e um ângulo de esterçamento W_z dado pelo eixo z; e essas informações são fornecidas ou informadas ao robô através de funções das bibliotecas de dinâmica de movimentação do robô (RosAria e p2os, por exemplo) e fazem a conversão direta-indireta das velocidades das rodas nas velocidades descritas espacialmente na imagem, e vice-versa.

Referente ao escopo desse trabalho, foram utilizados os comandos de esterçamento descritos em V_x, V_y e W_z para compor o comando que orienta o movimento do robô, enquanto os demais parâmetros V_z,W_x,W_y são considerados nulos ou desprezíveis nessa descrição cinemática. O valor

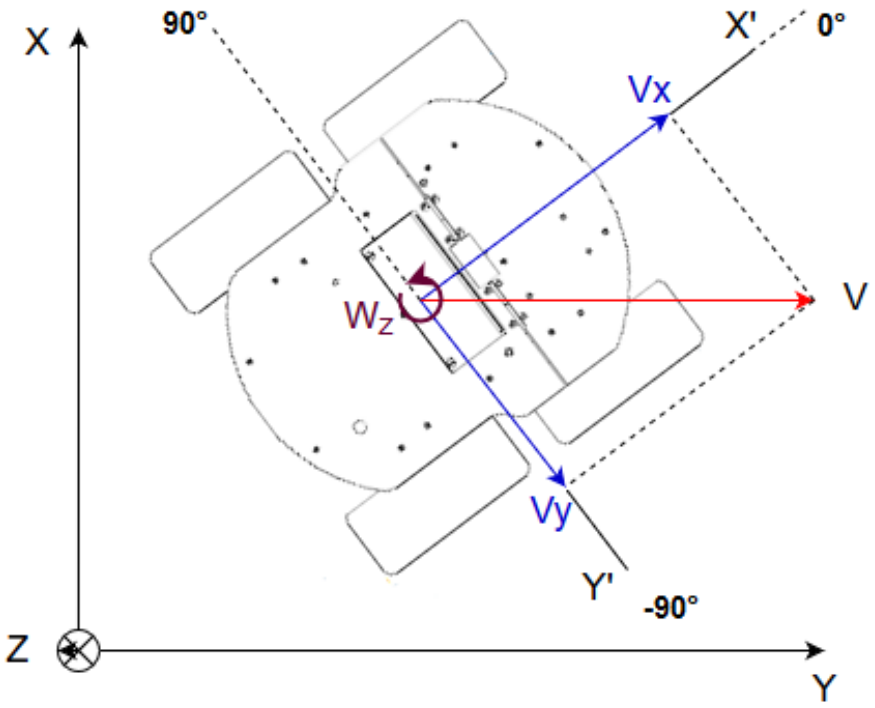


Figura 4.3: Diagrama espacial da cinemática de movimentação do Pioneer 3AT.

de V_y é diferente de zero para casos de curvas intensas e apresenta valor nulo quando em retas, com isso o conjunto de dados da aplicação apresenta esse valor como nulo para a finalidade de eliminar a dependência dessa coordenada na implementação prática.

Segundo o trabalho de Caracciolo[35], o cálculo de velocidade de rotação da rodas da direita e da esquerda pode ser dada pelas equações:

$$V_{esquerda} = V_x - (381/2) \times W_z \tag{4.1}$$

$$V_{direita} = V_x + (381/2) \times W_z \tag{4.2}$$

4.2 ROS e Gazebo

Para realizar a interface entre os comandos dos atuador e a leitura dos dados, a fabricante do Pioneer disponibiliza uma interface com estrutura cliente/servidor chamada de *ARIA* (*Advanced Robotics Interface for Applications*), assim como é definida em seu manual[36]. Entretanto, toda a interação que foi feita nesse trabalho ocorre através de duas bibliotecas de código aberto, conhecidas como *RosAria* e *p2os*, para prover diversas informações do robô em alto nível e simplificar a disponibilização desses dados. A tarefa dessas biblioteca se dá através da publicação de 'tópicos'(feitos por um 'nó'), com os comandos específicos e relacionados a atuação e percepção do ambiente, como por exemplo o controle de velocidade/aceleração e os sensoriamento da odometria. Ambas constituem pacotes da plataforma ROS, responsável pelos 'tópicos' e 'nós' que fornecem uma abstração para realização da implementação.

O nó é descrito como um processo em específico, como por exemplo, o controle de tensão nos motores ou um sensor de varredura laser, e dizem respeito a uma unidade de tarefa computada. Enquanto o tópico compreende o meio pelo qual os nós trocam informações, ou seja, os nós que produzem ou consumem informações utilizam esse barramento para conversar entre si. Desse modo, os nós funcionam como clientes e os tópicos como servidores, onde cada tópico de interesse é associado aos nós que lhe convém.

O ROS, juntamente das bibliotecas necessárias para o controle dos motores do Pioneer (p2os) tem grande importância na implementação dos comportamentos reativos no Pioneer, dada sua principal qualidade que é ter vários nós executando 'simultaneamente' e trocando mensagens entre si (nós estes que podem ser internos ao sistema e/ou externos, que é o caso da implementação da topologia de rede mestre/escravo).

O Gazebo por sua vez é uma ferramenta de simulação de ambientes e robôs para realização de prototipagem em ambiente computacional e foi utilizado para realização de alguns ensaios e teste de algoritmos antes da realização prática. Nesses pequenos experimentos, o robô foi inserido em um contexto/cenário genérico (disponibilizado pelo ambiente de simulação) e o ROS foi posicionado em paralelo, como ilustrado na figura 4.4. Esses testes, proveram um primeiro contato com o formato das mensagens trocadas entre os nós/tópicos e com a dinâmica de movimentação do Pioneer 3AT, como por exemplo o comando de velocidade recebido pelo p2os dado pela figura 4.5.

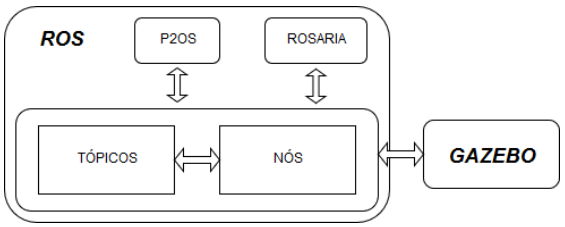


Figura 4.4: Diagrama de teste Gazebo e ROS

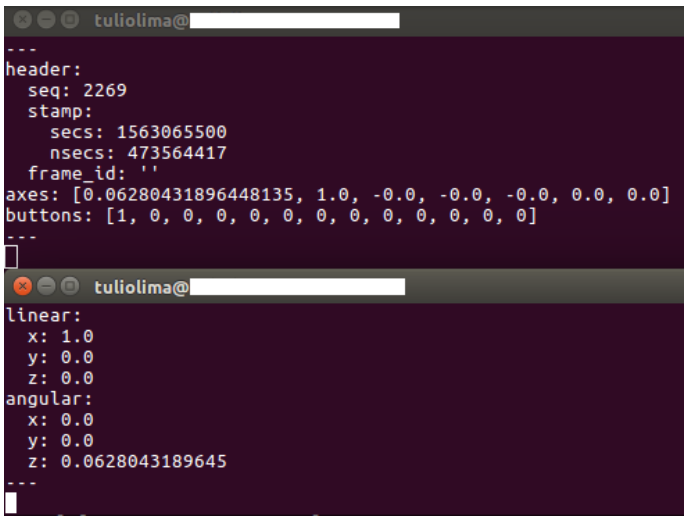


Figura 4.5: Mensagem disponibilizada no tópico do *joystick* (terminal superior) e o respectivo comando de velocidade (terminal inferior)

Seja via simulação no GAZEBO ou diretamente no Pioneer, o ROS esterá presente fazendo a abstração de alto nível entre um comando e a atuação em hardware, e assim constituiu parte fundamental da implementação deste trabalho.

4.3 *Joystick*

A orientação do robô foi realizada com a ajuda de um controle de simulador aéreo, apresentado na figura 4.6(a). Esse *joystick* apresenta diversos funções de interação, seja por botão ou mesmo por coordenadas de orientação.

(a) controle (b) Graus de liberdade

Figura 4.6: Controle *Joystick* utilizado e os respectivos graus de liberdade X e Y. Adaptado[11].

O controle *joystick* fornece um pacote de dados referentes a interação do usuário, e são interpretadas pelo programa *joystick_teleop.py* que é disponibilizado pela biblioteca *p2os_driver*. Esse programa realiza a extração dos dados percebidos em cada instante, publica em um tópico chamado */joystick* e posteriormente compõe uma mensagem para o tópico de velocidade */cmd_vel*. Esse último é observado por um nó, onde a mensagem será processada e convertida em sinais de baixo nível tais como tensão experimentada em cada motor, este nó por sua vez está diretamente ligado a atuação. A figura 4.7 descreve a relação entre o comando executado pelo piloto/agente e a saída apresentadas aos motores.

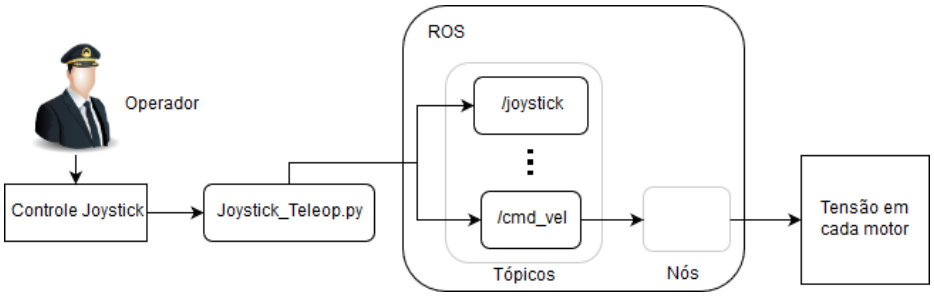


Figura 4.7: Fluxograma do uso do controle e fluxo de informações do operador até a atuação feita no motor

4.4 Câmera

A percepção do ambiente, importante para o comportamento reativo, é feita nessa trabalho pelo uso apenas de uma câmera. O equipamento utilizado é o apresentado na figura 4.8, e de maneira análoga ao *joystick*, as informações desse sensor são disponibilizadas por tópicos, na figura 4.9. Esse tópico, o qual é permitido apenas a leitura, fornece diversos formatos da informação capturada pelo sensor ótico, geralmente a mensagem é constituída por uma matriz com o tamanho da imagem, ou no melhor dos casos a informação dessa imagem comprimida.

Figura 4.8: Câmera [12]

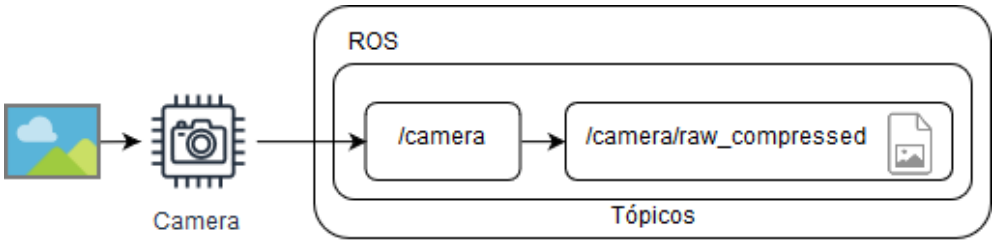


Figura 4.9: Fluxograma de captura de imagem e fluxo de informações para o tópico da câmera

4.5 Especificações dos Computadores utilizados

A seguir, é apresentado um panorama das características da máquina utilizada neste trabalho, compreendendo o *hardware*, *software* e *frameworks* implementados.

4.5.1 Hardware

As máquinas utilizadas para o treinamento e validação do modelo proposto, bem como para aquisição dos dados contém as seguintes especificações:

4.5.1.1 Computador de Treinamento

- CPU: Intel Core i3-7100 3.9 GHz 8ª Geração
- Memória(RAM): 8 GiB
- GPU: Nvidia GeForce GTX 1050Ti 4 GiB (Memória)

- Armazenamento: SSD 128 GiB

4.5.1.2 Pioneer - *Slave*

- Versão: Pioneer 3AT
- CPU: Intel Core Pentium M 1.73GHz 1ª Geração
- Memória: 496 MB
- Armazenamento: 76.5 GiB
- Controle(*Joystick*): Controle para simulador aeronáutico - *Joystick* Cobra Clone

4.5.1.3 Computador Auxiliar - *Master*

- CPU: Intel Core i5-2430 2.4 GHz 1ª geração
- Memória: 4 GiB
- Armazenamento: SSD 256 GiB

4.5.2 Software

Em parceria com o hardware descrito, foi feito o uso das seguintes ferramentas de software para cada máquina.

4.5.2.1 Computador de Treinamento

- Sistema Operacional: Linux Ubuntu 16.04 LTS 64 bits
- *Frameworks*: Keras 1.0.6 como *backend* do Theano: 1.0.4; Anaconda versão 2.7
- Linguagem: Python 2.7.6

4.5.2.2 Pioneer - *Slave*

- Sistema Operacional: Linux Ubuntu 14.04
- *Frameworks*: *ROS Indigo, Robot Operating System.*
- Linguagem: Python 2.7.6

4.5.2.3 Computador Auxiliar - Master

- Sistema Operacional: Linux Ubuntu 16.04
- *Frameworks*: ROS Kinetic, Robot Operating System, Gazebo.
- Linguagem: Python 2.7.6

Além disso foi necessário utilizar várias bibliotecas, tais como: Numpy, h5py, Matplotlib, openCV, roslib; as quais serviram para manipulação dos dados, interação com o ROS para captura e publicação de dados.

Capítulo 5

Metodologia

5.1 Estruturação

Dada a complexidade de programar um comportamento reativo e as facilidades que o aprendizado de máquina pode trazer na solução desses desafios, esse trabalho propõe a aplicação de uma rede neural convolucional para realizar o aprendizado de comportamentos reativos no Pioneer, considerando o sucesso que a mesma teve em sugerir informações de ângulo em trabalhos anteriores[32]. Para isso, o sistema precisa receber uma entrada de imagem do ambiente para então sugerir um ângulo de esterçamento conforme o que foi treinado, tal como uma reação ao que é visto no ambiente. A implementação de inteligência artificial e de aprendizado de máquina, nesse contexto, está posicionada de tal forma que a rede proposta consiga implementar as funções características do paradigma reativo, tais como prover uma resposta rápida para o estímulo e compor/determinar saídas corretas para estímulos diferentes.

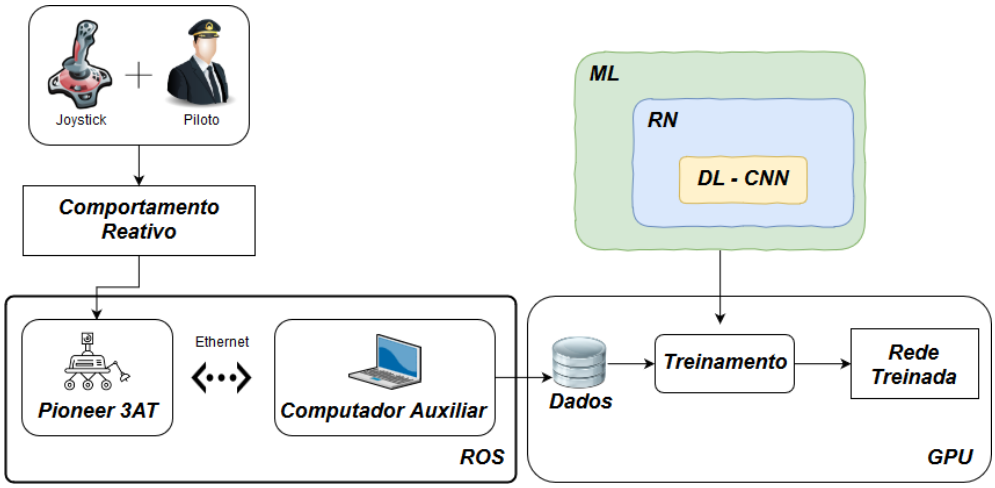


Figura 5.1: Fluxograma de implementação do trabalho até o treinamento.

Para tanto, foram definidos primeiramente os comportamentos desejados e em seguida um piloto tenta replicar como seria uma resposta ideal que espera do sistema para a situação (Figura 5.1), por fim o treinamento da rede é feito para o conjunto de dados desejados e uma série de

testes são feitos para avaliar se as metas foram atingidas.

Como segue nesse capítulo, também será descrito toda a metodologia utilizada para captura dos dados, os dados em si, além dos comportamentos reativos, a arquitetura utilizada na implementação e como ela foi posteriormente avaliada (Figura 5.2).

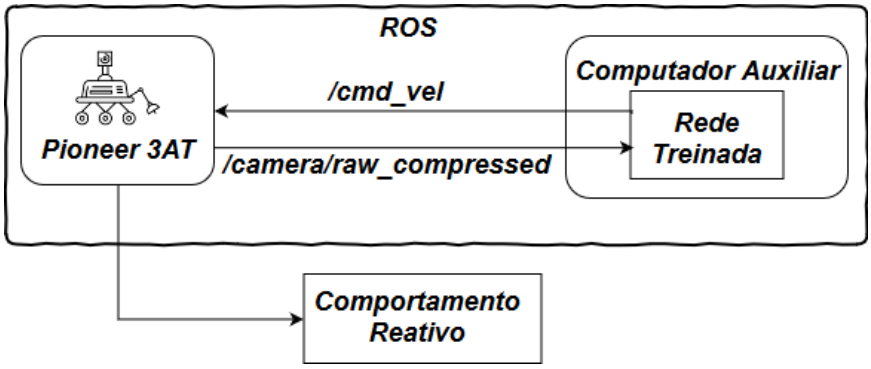


Figura 5.2: Fluxograma da implementação dos testes práticos realizados no trabalho.

5.2 Comportamentos Reativos

Como foi comentado, replicar comportamentos reativos computacionalmente com o uso de imagem pode ser muito difícil e custoso, principalmente quando se espera qualidade e uma generalização abrangente do que deve ser feito. Nesse contexto vamos implementar técnicas de aprendizado de máquina para verificar a hipótese de que um conjunto de 3 comportamentos reativos, conduzido por um piloto, podem ser 'aprendidos' por redes neurais.

Os 3 comportamentos são descritos da seguinte forma:

- Retas: Seguir a faixa de deficiente visual
- Curvas Suaves: Seguir a faixa de deficiente visual e fazer curvas quando não puder seguir em frente
- Curvas em T: Seguir a faixa de deficiente visual, fazer curvas quando não puder seguir em frente e em situações de bifurcação realizar a curva sempre à direita.

Em todos os casos foi definido uma velocidade máxima de cruzeiro para a coordenada que compreende o eixo x do controle, com valor máximo de 1.0, correspondente a 0.5 m/s desenvolvido na prática. Além da variável V_y permanecer com valor nulo, até que uma curva muito brusca seja feita, e deixa o cenário livre para que o condutor do robô realize apenas a orientação espacial, referente ao eixo z (velocidade angular, W_z), que é o dado de esterçamento efetivo.

5.2.1 Retas

Nesse comportamento, a proposta é andar sempre o mais próximo da faixa, ou seja, de modo que o robô fique localizado imediatamente acima da mesma. E assim, esse o comportamento é realizado sobre o comando de um piloto, que avalia subjetivamente ao seu modo de condução, e que sugere correção de ângulo que deve ser feita para obter sucesso em manter o robô sempre em cima da faixa.

5.2.2 Curvas Suaves

O esperado para essa situação é uma composição do comportamento anterior e um novo, que é definido pela necessidade de fazer curvas sempre que não houver mais como ir em frente e na direção que a faixa estiver disponível. Logo, o rede que será treinada para esse comportamento, deve conseguir fazer curvas suaves para as direções disponíveis caso não tenha como ir em frente, e ainda sim conseguir se manter em cima da faixa quando possível seguir em frente. As curvas que foram adicionadas nesse ponto, são descritas como suaves pois começam antes e terminam depois da identificação de uma interseção ortogonal de faixas (logo, o robô deve se distanciar da faixa com antecedência e depois retornar para faixa sem movimentos bruscos), assim o valor de esterçamento não será muito grande e a faixa estará sempre presente na imagem.

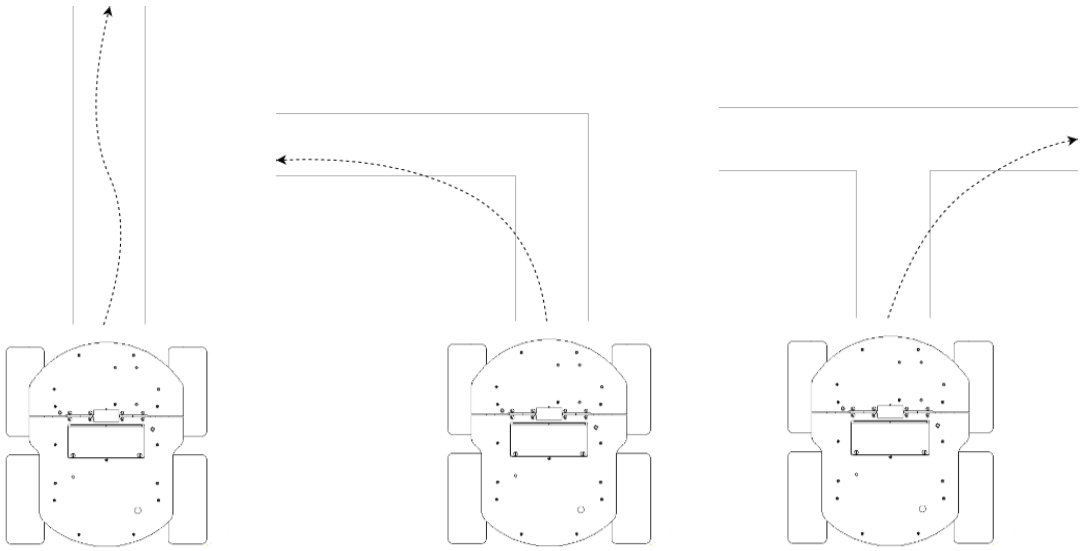
5.2.3 Curvas em T

Continuando a composição de comportamentos, o desejado nesse comportamento é de que sejam executadas todos os comportamentos feitos em curvas suaves adicionados de um terceiro que é descrito pela realização de curvas suaves à direta, sempre que a situação de bifurcação (sem possibilidade de seguir em frente) for identificada. Aqui é desejado o mesmo procedimento que são feitas as curvas suaves, só que para bifurcações, e assim o piloto deve suavemente adiantar a curva sempre para direita quando identificado a situação específica de ramificação apresentada.

5.3 Dados

Ter um conjunto de dados organizado deve ser a maior das preocupações do cientista de dados que trabalha com aprendizado de máquina, além de ter um volume de dados adequado e com informações que sejam realmente relevantes para alcançar os seus propósitos. E nesse sentido, a metodologia de como os dados foram fornecidos à rede é semelhante ao proposto na solução da *comma.ai*[37], mas com outros dados que são fruto da implementação diferente. Ou seja, o formato utilizado para armazenar os dados é o $hdf5$, que permite acesso rápido e indexado aos dados de treinamento, e nele estão contidos os dados de imagens, ângulo de esterçamento e comando de velocidade, todos alinhados por uma marca de tempo.

Apesar de fornecido o valor de comando de velocidade, apenas as imagens e o ângulo de esterçamento foram utilizados. A imagem tem formato RGB e tamanho de 240×320 pixeis, e



(a) Retas (b) Curvas Suaves (c) Curvas Suaves em T

Figura 5.3: Comportamentos Reativos - Representação gráfica de cada comportamento sugerido.

refletem a amostra do espaço visual do ponto de vista do robô, capturada em uma taxa de 30 fps (quadros por segundo). O ângulo de esterçamento compreende o valor de coordenada apresentada pelo controle e sugerida pelo piloto como sentido de movimentação, essa variável é extraída antes de ser convertida em um comando de velocidade, que por sua vez é enviado ao tópico */cmd_vel* no ROS e possui uma taxa de amostragem de 20 amostras por segundo (posteriormente o controle de dinâmica adaptado ao modelo do robô aplica a tensão correspondente de cada conjunto de motores que estão ligados as rodas).

Dado o desalinhamento dessas informações, foi necessário criar um programa que organiza esses dados temporalmente com a informação que tem menor frequência de amostragem (informação do tópico */joystick*), assim a imagem que está localizada temporalmente mais próxima ao comando do controle será associada a este por um mesmo *'index'*.

Os comandos do controle e de velocidade, que compreendem a informação útil de esterçamento, apresentam escala que vai de -1.0 à 1.0, respectivos aos sentidos máximos de direita e esquerda. Esses dados precisaram ser ajustados pois a escala do controle não apresenta uma variação grande entre as amostras que seja vista como um acerto durante o treinamento, e por isso a escala dos dados utilizados na implementação do *comma.ai* que tem intervalo de [-502.3,512.6] baseou a alteração de escala nesse trabalho para [-507.45,507.45], que é presente em todos os arquivos de dados. Assim, por exemplo, uma informação do esterçamento que se encontravam entre 0 e 1 foi redimensionadas para um fundo de escala de 0 até 507.45 . Ainda sobre esse dado, a informação de velocidade referente à V_x, que simboliza a velocidade de translação, é apresentada dentro da faixa de valores de -1.0 à 1.0, e na coleta dos dados foi utilizado o valor fixo de 1.0 (que na prática significa uma velocidade de 0,5 m/s).

Os dados compreendem informações com 108.480 amostras de retas, 126.289 de curvas suaves e 41.774 de curvas suaves em T, como resultado de cerca de 8 horas de gravação, das quais

apenas as informações mais pertinentes foram extraídas. Foram evitados amostras de momentos que poderiam adicionar erros e que não eram o propósito da implementação, como por exemplo, momentos que o robô ficou parado por mais de um minuto, na entrada e saída do local onde o robô é guardado ou em instantes em que foi necessário realizar alguma ação que não fosse correspondente a nenhum dos comportamentos reativos desejados.

5.4 Coleta de Dados

Para realizar a captura dos dados, foi utilizado um esquema escravo/mestre entre dois computadores, como no fluxograma descrito na figura 5.1, no qual o computador do Pioneer trabalha em conjunto com um computador auxiliar, através de uma rede cabeada para que o ROS que é executado no mestre possa interagir com o escravo. Dessa forma é permitido que ambos os computadores possam interagir por tópicos de mensagens, entretanto o $'master'$ aqui só atua lendo as mensagens no modo de captura. Dentre os tópicos fornecidos pelo 'escravo' estão os dados de imagens, comandos de velocidade, comandos do controle($joystick$) e até de odometria(posição), essas informações são inteiramente sincronizadas com o relógio do computador que as disponibiliza e registrados com uma marca de tempo em cada mensagem.

A nomenclatura mestre/escravo comentada aqui não tem a intenção de ser fiel ao conceito computacional de orquestrar as ordens produzidas pelo *master* e executadas pelo escravo, e dizem respeito apenas a forma como foram implementadas as referências de endereço de cada robô. O *master* apesar de não enviar comandos nesse momento, executa uma rotina que salva todos os dados importantes em um formato $'.bag'$, para que depois de salvo em disco possa ser feito os devidos processamentos que alinha os dados e salva no formato final $'.h5'$.

Foram feitos testes de captura direta no robô, mas eles não trouxeram um bom resultado, pois gera uma sobrecarga ao fraco processador do robô e ocasiona travamentos durante a coleta, então foi escolhido o formato que atendeu melhor ao experimento($master/slave$), que é realizado com o uso de uma computador auxiliar ao robô, com mais poder de processamento e mais velocidade de armazenamento, conectados por um cabo de rede *ethernet* para proporcionar menos perdas de pacotes e ter o mínimo atraso possível.

5.5 Arquitetura da Rede Neural

A rede sugerida tem formato de um rede neural convolucional profunda, que faz bastante sentido quando se trabalha com imagens pois auxiliam na extração de características ou mesmo objetos que podem ser fundamentais para orientação do robô no problema deste trabalho. Essa arquitetura também foi obtida do exemplo fornecido pela *comma.ai*, entretanto a proposta tem uma imagem de $3 \times 160 \times 320$ e nessa trabalho terá uma leve alteração da dimensão da imagem de entrada, para adaptar ao conjunto de dados capturado que é de $3 \times 240 \times 320$. A rede irá sugerir como resposta um valor de esterçamento considerando a imagem de entrada, tal como em um problema de regressão, mas com a grande diferença de envolver imagens para fazer isso.

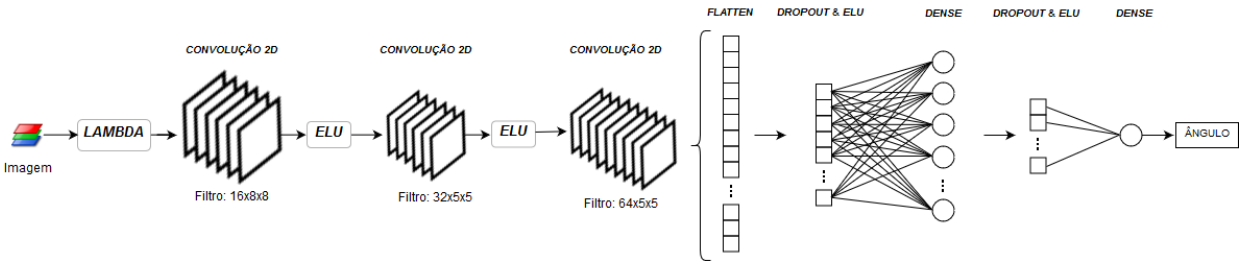


Figura 5.4: Arquitetura da Rede Neural, com a entrada composta de uma imagem RGB e a saída é dada pelo valor ângulo

No modelo sequencial feito em código, cada camada a topologia da rede é descrita da seguinte forma, respectivo a sua função:

1. Lambda:

   Nessa camada é permitido a criação de operações ou camadas que não são pré-definidas, e possibilita que sejam aplicadas sobre os dados sem requer que pesos sejam aprendidos. Assim é aplicada uma função que normaliza os dados da imagem de entrada, e para cada pixel que tem valor entre 0 e 255 essa camada apresenta uma saída de imagem de mesmo tamanho mas com informações que vão de -1 a 1.

   A função definida em Lambda é dada por:

   $$x : \frac{x}{127.5} - 1. \tag{5.1}$$

   E tem dimensões de entrada e saída com mesmo tamanho de 3x240x320.

2. Convolução 2D:

   Essa camada implementa filtros de convolução que serão computados na imagem de entrada, e permite que informações de dependência espacial relacionada as cores da imagem sejam extraídas, e trabalha bem com imagens que envolvem cores. Nessa camada foi escolhido a quantidade de filtros a serem aplicados e posteriormente aprendidos, o passo em que os filtros são aplicados, o tamanho do *kernel* do filtro e o preenchimento nas bordas.

   - Número de Filtros: 16
   - Passo dos Filtros: 4×4
   - Tamanho do 'Kernel': 8×8
   - Preenchimento de Bordas: "same", que não adiciona nem retira as bordas quando o filtro é computado.

   A entrada é dada pela camada anterior, com dimensões de $3 \times 240 \times 320$, e saída dessa camada é de $16 \times 60 \times 80$.

3. ELU:

   Essa camada é a implementação de um tipo de função de ativação, que são funções fruto de uma abstração que representa matematicamente o potencial de disparo de uma rede neuronal, ou seja, uma rede de neurônios biológicos. Os tipos mais comuns são os $ReLU$, $Sigmoid$, $BinaryStep$. Do termo em inglês $'ExponentialLinearUnit'$, descrita pela seguinte fórmula e pelo gráfico na figura 5.5. E como foi explicado por Clevert[38], em suas palavras, essa função acelera o aprendizado de redes neurais profundas e leva a taxas de acurácia maior na classificação, comparadas com a outras funções de ativação.

$$f(x) = \alpha \times (e^x - 1.),\ \forall\ x < 0 \tag{5.2}$$

$$f(x) = x,\ \forall\ x \geqslant 0 \tag{5.3}$$

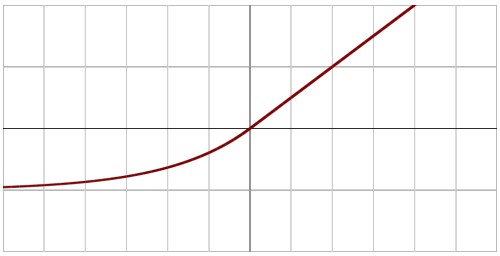


Figura 5.5: Curva da função ELU [13].

   O parâmetro de entrada é a saída da camada anterior, $16 \times 60 \times 80$, enquanto que a saída dessa camada tem as mesmas dimensões da entrada. O parâmetro *alpha*, que molda a escala dos valores negativos, tem valor 1,0 quando não especificado na chamada função.

4. Convolução 2D:

   Essa camada é implementada da mesma forma que a primeira convolução, mas diferente daquela esta tem os seguintes parâmetros:

   - Número de Filtros: 32
   - Passo dos Filtros: 2×2
   - Tamanho do 'Kernel': 5×5
   - Preenchimento de Bordas: "*same*"

   A entrada é dada pela camada anterior, com dimensões de $16 \times 60 \times 80$, e saída é de $32 \times 30 \times 40$.

5. ELU:

   Mas uma vez essa camada é aplicada com da mesma forma que a anterior e fornece como saída um dado de $32 \times 30 \times 40$, ajustados segundo a função ativação ELU.

6. Convolução 2D:

   Novamente, a camada de convolução 2D é aplicada e fornece os dados, com os seguintes parâmetros:

- Número de Filtros: 64
- Passo dos Filtros: 2×2
- Tamanho do 'Kernel': 5×5
- Preenchimento de Bordas: "*same*"

A entrada é dada pela camada anterior, com dimensões de $32 \times 30 \times 40$, e saída é de $64 \times 15 \times 20$.

7. *Flatten*:

   Essa camada realiza a operação simples de 'achatar' os dados de entrada, fornecendo como saída um vetor com todos os dados. Essa camada é usada para conectar o resultado das convoluções com a camada de 'Dense', que vão realizar o ajuste com base nesses dados.

   A entrada é dada pela camada anterior, com dimensões de $64 \times 15 \times 20$, e a saída é uma vetor de $1 \times (19.200)$

8. *Dropout*:

   Como o objetivo de evitar o problema de sobre-ajuste, implementa uma função de substituir randomicamente valores do dado de entrada por zero, dado um certo valor de taxa dos dados. É importante destacar que essa implementação só é feita durante o treinamento, e para o caso foi escolhido 0,2, que é correspondente a 20% dos dados de entrada sendo levados a 0. O parâmetro de entrada é a saída da camada anterior, $1 \times (19.200)$, enquanto que a saída dessa camada tem as mesmas dimensões da entrada.

9. ELU:

   Mais uma vez é implementada uma camada de ELU, e entrega como saída o mesmo dado de entrada ajustado por uma função de ativação que tem suas vantagens para redes convolucionais. O parâmetro de entrada é a saída da camada anterior, $1 \times (19.200)$, enquanto que a saída dessa camada tem as mesmas dimensões da entrada.

10. *Dense*:

    Também conhecida como camada totalmente conectada, compostas por neurônios artificiais ajustados por pesos e que são conectados a todos os dados de entrada fornecidos pela camada anterior, e tem como saída uma resposta sugerida pro cada neurônio.

    A entrada é dada pela camada anterior, com dimensões de $1 \times (19.200)$, e a saída é dada pela quantidade de neurônios definidos nessa camada, nesse caso 512 unidades, resultando em um vetor com as dimensões 1×512.

11. *Dropout*:

    Mais uma vez a camada de dropout é implementada, mas dessa vez 50% dos dados serão levados a 0, de forma aleatória. O parâmetro de entrada é a saída da camada anterior, $1 \times (512)$, enquanto que a saída dessa camada tem as mesmas dimensões da entrada.

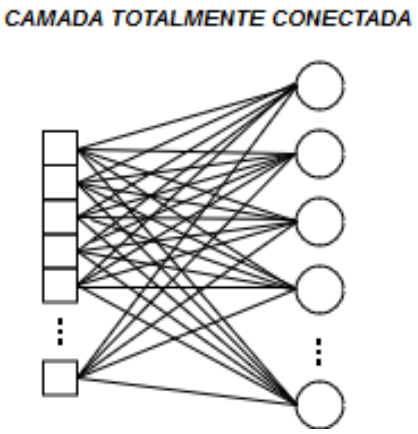


Figura 5.6: Ilustração das conexões feitas por cada neurônio na camada Dense.

12. ELU:

    Igualmente foi ilustrado nos outros casos, essa camada de ELU, aplica a sua função de ativação mas não altera as dimensões dos dados de entrada. Portanto, a entrada de $1 \times (512)$ é convertida em uma saída com as mesmas dimensões.

13. *Dense*:

    Nessa ultima camada, é implementado uma camada de apenas um neurônio que é alimentado por todos as 512 informações da camada anterior, de forma que esse mesmo neurônio apresente um resultado que é o ângulo de esterçamento sugerido. A entrada é dada pela camada anterior, com dimensões de $1 \times (512)$, e a saída é dada por apenas um valor.

5.6 Treinamento

Foram realizados os treinamentos de 3 redes com a mesma arquitetura descrita anteriormente, para os respectivos comportamentos esperados, e além disso para cada rede foram testados treinamentos com 500, 1000 e 2000 épocas, com a ideia de observar se houve sobre-ajuste. Para cada comportamento são utilizados dados diferentes, escolhidos aleatoriamente e disponibilizados na proporção de 70% de informações para treinamento, 15% para os ajustes feitos durante o treinamento e 15% para validação pós treinamento que fornecerá a métrica de erro $NRMSE$ e matriz de confusão para cada situação. Dessa forma teremos quantidades de dados diferentes para cada rede, e a tabela 5.1 descreve as quantidades de exemplos utilizados para cada etapa desse procedimento.

Tabela 5.1: Separação dos dados de treinamento

Redes Treinadas	Treinamento	Teste	Validação
Retas	75936	16272	16272
Curvas Suaves	164339	35215	35215
Curvas Suaves em T	193580	41481	41481

No treinamento existe outras variáveis que foram mantidas pela proposta do modelo fornecido pela *comma.ai*, como por exemplo o tamanho de lote de dados processados em cada passo de treinamento sugerido com o valor de 64 amostras, a quantidade de dados utilizados no teste de treinamento ao final de cada época com valor de 1.000 amostras, quantas iterações serão realizadas em cada época (com valor de 10.000), a função de custo indicada pelo erro quadrático médio (mse) e otimizador do tipo $'Adam'$ (combinação do melhor do $AdaGrad$ e $RMSProp$), esse último provavelmente foi escolhido pelos ótimos resultados que proporciona comparado com os demais métodos de otimização[39].

5.7 Análise Qualitativa

Para comparar os resultados obtidos por cada rede, em sua determinada configuração, serão apresentados agora duas métricas que auxiliam na avaliação do quão bom podem ser os resultados. Essas técnicas serão implementadas posteriormente ao treinamento de todas as redes, e completam a análise computacional. É muito importante destacar que para realização dos testes de maneira correta, os valores sugeridos da rede devem ser dimensionados na antiga forma que foram capturados, assim a rede irá sugerir valores da escala [-507.45 a 507.45] e após a conversão, a informação estará entre a faixa original que vai de [-1 a 1].

5.7.1 *NRMSE*

A primeira das técnicas é dada pela normalização da raiz do erro quadrático médio (nesse texto ele é referido como NRMSE apenas); e envolve o cálculo sobre uma série de amostras de validação e resumir em valor numérico, o quão próximo o valor de esterçamento sugerido está dos valores fornecidos nos dados utilizados, ou seja, quanto menor o erro melhor foi a capacidade da rede em generalizar soluções.

A fórmula que define esse procedimento é dada pela seguinte equação:

$$NRMSE = \frac{\sqrt{\sum \frac{\left(y_{observado} - y_{predito}\right)^2}{n}}}{y_{máx} - y_{mín}} \tag{5.4}$$

De mameira sucinta essa equação representa a raiz quadrada do erro médio quadrático, ajustados pelos valores de máximo e mínimo, e fornece uma medida normalizada que vai de 0 a 1, para ilustrar o desvio médio entre observado e predito. $y_{observado}$ é o valor de esterçamento sugerido pelo piloto e $y_{predito}$ é a sugestão fornecida pela rede para a mesma situação.

5.7.2 Matriz de Confusão

A matriz de confusão[40] permite evidenciar padrões e relações entre variáveis, de forma semelhante a matriz de correlação, e geralmente é aplicada na análise de problemas de classificação

supervisionada, com poucas variáveis. Essa matriz permite a visualização de como a rede neural performou a tarefa, e é essencial para uma análise mais geral, entretanto será utilizada uma simplificação desse modelo devido à dificuldade em ilustrar cerca de 200 casos de falsos e verdadeiros, positivos e negativos, e tem apenas o intuito de mostrar como os dados se distribuíram nas aplicações.

Nessa segunda proposta, a apresentação das métricas de avaliação da rede é feita visualmente, através da relação entre os valores que são preditos e observados (com escala de [-1,1]), na forma de uma matriz bi-dimensional com tamanho 100×100. Assim, para cada linha existe um valor predito associado, e para coluna, existe uma amostra observada correspondente; cada posição da matriz possui o valor de quantas amostras tiveram a mesma relação entre os dados, e representa em seu conjunto a relação de como os valores se distribuem, ou não. No caso desse trabalho, é esperado que essa matriz seja composta de apenas uma diagonal principal, ou algo parecido.

Devido os dados colhidos apresentarem muitas amostras com valor próximo a zero, é então adotado uma normalização das informações, para que os demais pontos não sumam na figura.

5.8 Análise Subjetiva

Para fechar o ciclo de avaliação, duas implementações visuais foram feitas em vídeos para cada comportamento e mostram como a rede deve funcionar na prática. A primeira ilustra o esterçamento sugerido sobreposto ao amostrado para a respectiva imagem, enquanto a segunda implementa os comportamentos direto no robô e integralmente sugeridos pela rede, e constitui uma validação que leva em conta todo o espaço da aplicação, inclusive um modelo dinâmico de movimentação do robô, que não foi apresentado diretamente para rede.

5.8.1 Implementação Computacional

Nessa parte, um código disponibiliza a visualização prévia de como a rede pode se comportar, e tem como entradas: a rede (com estrutura e pesos, salvos em arquivos *.json* e *.keras*, respectivamente) e o conjunto de dados, tal como os que foram apresentados. Como saída o código provê um gráfico das informações de esterçamento fornecidas e preditas, e permite observar uma prévia de como a rede se comporta para um ambiente que não foi apresentado durante nenhuma etapa de treinamento. Nesse caso, foram colhidas pequenas amostras para cada comportamento, com intuito de evidenciar o modo de operar de cada rede.

Dada a dificuldade de apresentar um vídeo mais elaborado com esse documento, para essa parte optou-se por apresentar um gráfico do esterçamento sugerido e o realizado pelo piloto, referente a 3 amostras (uma para cada comportamentos reativo) inteiramente novas e jamais apresentadas durante treinamento.

5.8.2 Testes Práticos

Sugerido como parte adicional ao trabalho, a aplicação dessa fase constituiu na criação e execução de um código em laço iterativo que captura a imagem publicada durante o instante da execução, apresenta a mesma para a rede e obtém um ângulo de esterçamento, por fim, compõe a coordenada de esterçamento do comando de velocidade que guia o robô.

Assim, é desejado observar como cada rede após treinada e validada, funciona enquanto interage com um modelo de dinâmica de movimento real do Pioneer. Essa parte carece de coleta de dados experimentais, podendo ser observado por vídeo disponibilizado no link fornecido no Capítulo 6.

Capítulo 6

Resultados

Nesse capítulo é apresentado um detalhamento das informações que foram coletados, da captura dos dados e os resultados que foram obtidos da arquitetura de rede neural convolucional utilizada, para cada comportamento, produto de testes computacionais e práticos.

6.1 Dados capturados

Foram capturados 3 conjuntos de dados diferentes, para cada tipo de comportamento desejado, e foram mesclados como disposto na tabela 5.1 para a composição de comportamentos reativos. O ambiente de coleta dos dados compreendeu todo o 2º andar do prédio do CIC, e foi restrito apenas à amostras que representam informações adequadas para treinamento.

6.1.1 Conteúdo

Os arquivos foram construídos por cerca de 8 horas de gravação, compreendendo um total de aproximadamente 300.000 amostras para casos de retas, curvas suaves e curvas em T, e compõem vários arquivos com formato '.h5' os quais indexam os comandos realizados pelo piloto para cada imagem capturada no momento. Esses dados, colhidos inteiramente durante o período da noite são alinhados pela sua marca temporal (obtida no momento que cada comando chega ao ROS), e contém vários exemplos do que pode ser entendido como um comportamento reativo, como por exemplo, manter-se sobre a faixa de deficientes visuais, ou seja, segui-la em linha reta; e além de exemplos de curvas.

Os dados foram capturados no formato de arquivo $'.bag'$ e contém na sua estrutura as imagens obtidas pela câmera, comando do controle (*joystick*), comando de velocidade e odometria. Após a realização da coleta foi necessário tratar todos os dados, filtrar as informações indesejadas, alinhar as informações temporalmente, e só então criar o arquivo '.h5' que provê uma leitura mais rápida do conteúdo no padrão atual do código. Dentre todos esses dados, os únicos que foram utilizados são os de comando de controle e imagem, dada a precariedade do que era obtido pela odometria, e ausência de marca temporal na mensagem de comando de velocidade; além de outras razões

que envolvem o objetivo do comportamento reativo que é apresentada a seguir. O alinhamento comentado, foi feito com base na informação que possui a menor taxa de amostragem, no caso a informação proveniente do tópico */joystick*.

Em resumo, esses arquivos são um registro das formas que o piloto reagiu a situação, feitos de uma forma geral pela sua noção de orientação somados de sua experiência na manobra com o equipamento, e não se baseia diretamente em uma decisão com base na imagem observada do ponto de vista do robô. O modo de pilotar, apesar de realizado por um único piloto com experiência na dinâmica do Pioneer, é muito difícil de ser repetido com grande exatidão e diz um pouco a respeito da dificuldade de manobrar com um controle enquanto deve acompanhar o passo do robô, além de ter que estabilizar o máximo um comando para evitar 'solavancos' (que intuitivamente podem prejudicar como a rede aprende a reagir).

6.1.2 Breve análise do cenário dos dados e considerações

Após a coleta ser realizada, foi feita uma breve observação do que foi coletado, e confirmou-se a presença de um atraso inerente principalmente da dinâmica e também pelo tempo de reação da malha de controle do robô. Observou-se uma pequena alteração na imagem para os primeiros instantes que o comando do controle foi publicado (geralmente 10 amostras até que seja registrado uma real movimentação na imagem, às vezes um pouco menos), entretanto e teoricamente, esse atraso deve ter mais impacto se a rede a ser treinada considerar as amostras temporalmente, o que não é o caso. Como foi comentado antes, essas situações foram recortadas do conjunto de dados e portanto não devem ser uma preocupação, pelo menos em um primeiro instante.

Essa relação dos dados no tempo é muito importante na hora da validação prática, pois se o modelo sugerir variações grandes nos dados sugeridos, a dinâmica do robô não vai acompanhar e pode tanto suavizar como intensificar, o que é esperado do comportamento reativo. Logo se durante o teste o robô fizer algum movimento brusco e perder a sua referência (a faixa) o robô estará realizando um outro comportamento para o que a rede acha que pode ser uma melhor resposta pra uma situação jamais apresentada, erro que pode ocorrer tal como em um código confeccionado por um humano ou mesmo o executado com a ajuda do controle.

Um outro atraso também foi mapeado durante o processo de coleta, que é dado pela diferença de quando efetivamente o piloto pensou em mover o controle até quando esse comando de velocidade (compondo o de esterçamento) foi publicado pelo tópico de */cmd_vel* no ROS. Então foi definido uma hipótese, considerando que esse atraso ocorre tanto para o caso de uma guiagem por código como para o controle de trajetória por uma humano, o que se espera é que o comportamento de reagir seja aprendido, incluindo o atraso em controlar os rumos tomados pelo robô. Perceba que a diferença é que o comando do controle precede o comando de velocidade, informações diferentes na prática embora aquela irá compor esta, e o que deve ser esclarecido é que elas diferem no máximo por um espaço de tempo de menos de uma unidade de amostragem.

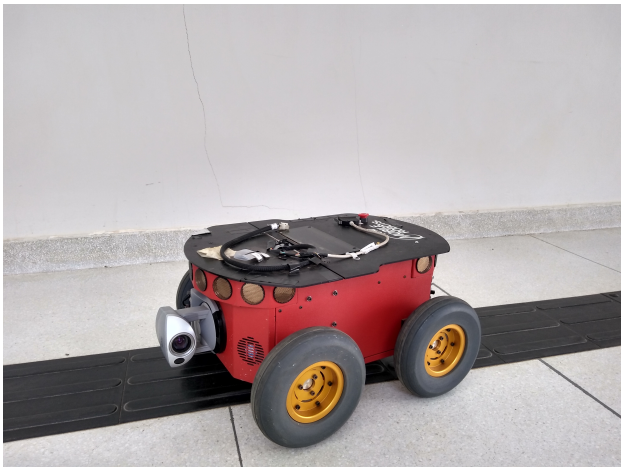
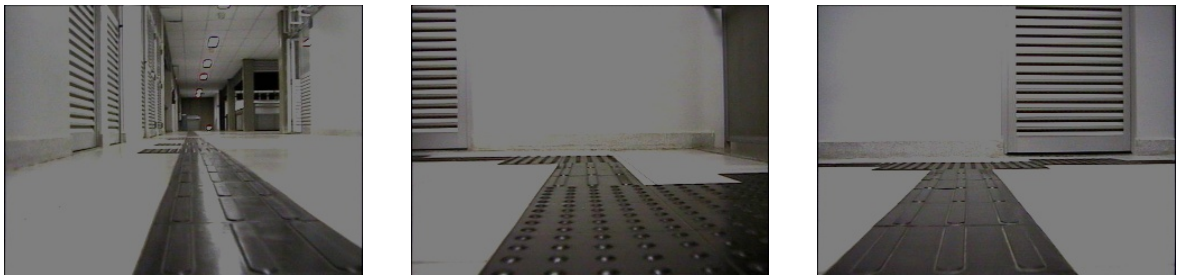


Figura 6.1: Posição da Câmera na estrutura do Pioneer 3AT

As imagens são obtidas de um ponto de vista em particular, que é localizado na parte frontal do robô (Figura 6.1), e um ajuste no ângulo de rotação da câmera é necessário pra obter uma imagem alinhada com o chão (horizontal). A localização da câmera não é a ideal, a melhor posição é sugerida na parte superior do robô, mas por questões técnicas não foi possível mudar a câmera de posição.



(a) Retas (b) Curvas Suaves (c) Curvas Suaves em T (Bifurcação)

Figura 6.2: Ponto de Vista da Câmera do Robô

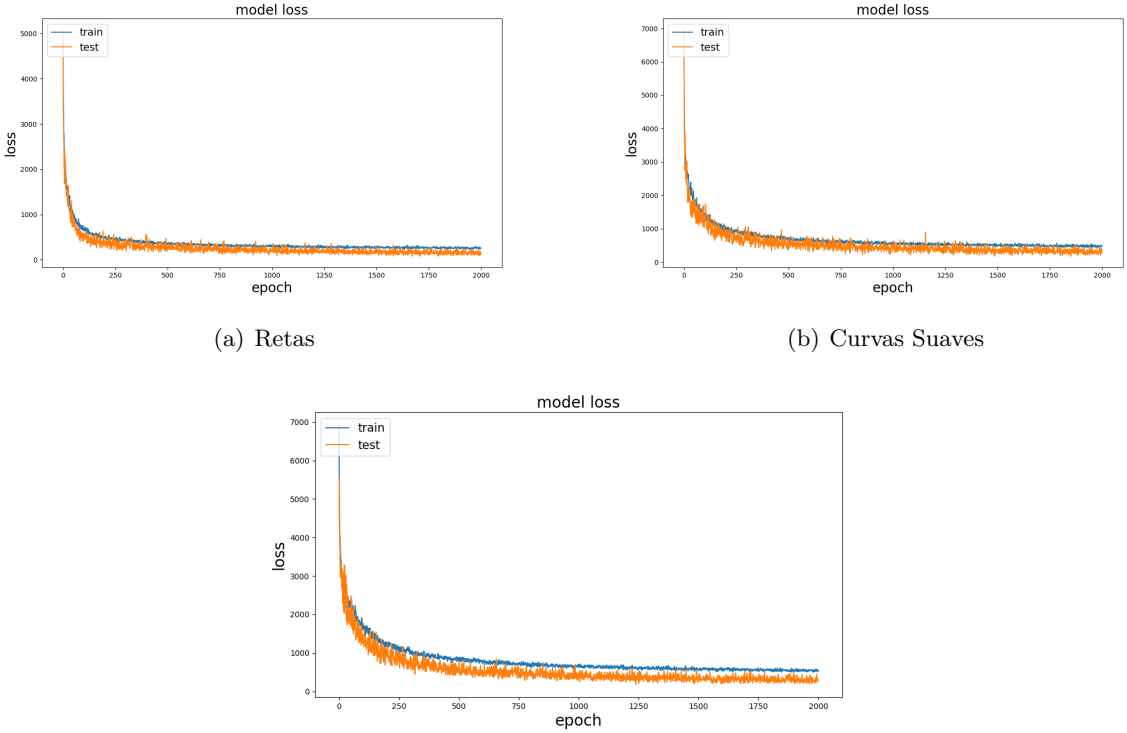
A figura 6.2 ilustra situações de cada comportamento reativo almejado, na ordem retas, curvas suaves e curvas em T; e também o ponto de vista dada a posição da câmera e os devidos ajustes na inclinação que foram implementados.

6.1.3 Treinamentos

Para cada proposta de comportamento, foi obtida uma rede que foi treinada com os devidos conjuntos de dados, dada a mesma estrutura sugerida e número máximo de 2000 *epochs*/ 64 de *batch*, o que totalizou um período de aproximadamente 9 horas para cada execução.

Antes da inicialização do servidores locais de dados, uma separação aleatória de todo o conjunto é feita, e divide os dados em 70% para treinamento, 15% para validação(aquela feita durante o treinamento e ao final de cada época) e 15% para teste (dados que são utilizados para obter o erro *NRMSE* ilustrado a seguir).

Os treinamento convergiram na dinâmica, mostrada na figura 6.3 :



(a) Retas

(b) Curvas Suaves

(c) Curvas Suaves em T

Figura 6.3: Representação da evolução de cada treinamento.

Durante a avaliação da rede que ocorre durante o treinamento, é observado que a partir da 1200ª época o treinamento não apresenta alteração relevante, contudo é necessário continuar até o 2000ª etapa para evitar que platôs interfiram na convergência do processo e pra composição da análise de sub-ajuste e sobre-ajuste comentada.

Apesar da arquitetura ser definida pelo código disponibilizado pelo *comma.ai*[37], algumas alterações foram feitas na estrutura, tais como utilizar um tamanho da imagem maior, modificar a camada lambda para que não fazer ajustes na imagem de entrada, aumentar/diminuir o tamanho dos filtros utilizados nas camadas de convolução e inserir mais camadas; de forma a encontrar algo que pudesse melhorar os resultados obtidos, mas essas tentativas não apresentaram mudanças que justificaram uma escolha diferente do padrão proposto inicialmente e portanto o modelo não foi alterado para nenhum dos resultados finais obtidos.

6.2 Resultados Obtidos

6.2.1 Retas

A figura 6.4, apresenta a matriz de confusão para casos de retas, e nessa imagem é possível ver como os dados se concentraram sobre a diagonal principal tal como uma matriz identidade.

Tabela 6.1: Erro *NRMSE* Retas

Comportamento	500 épocas	1000 épocas	2000 épocas
Retas	0.01683	0.01410	0.01043

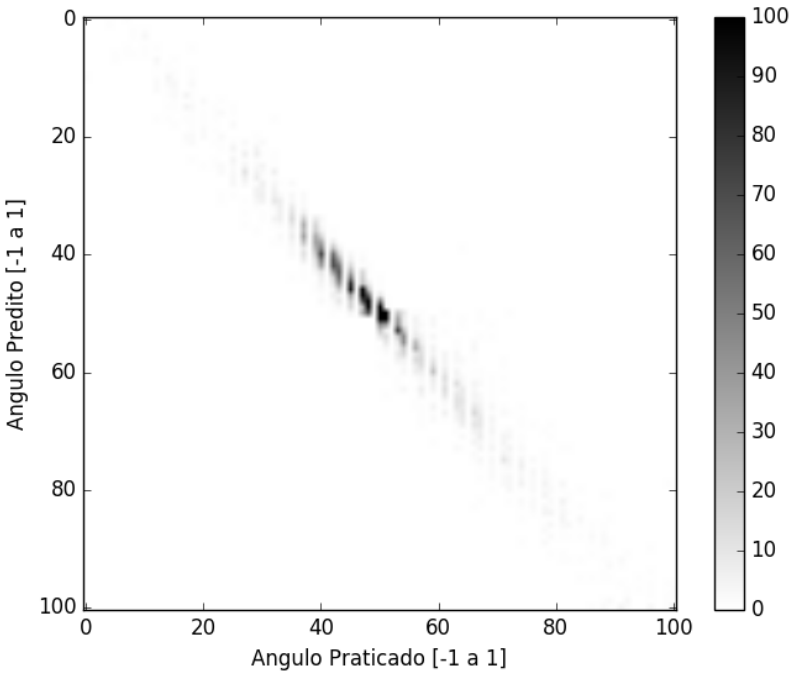


Figura 6.4: Matriz de confusão normalizada para validação em casos de retas

Outra informação que pode ser extraída é sobre como os dados estão distribuídos, e é possível observar que as amostras se concentram próximas ao centro da imagem, como consequência de uma variação mais cautelosa do controle quando em retas.

A figura 6.5, apresenta o gráfico resultante de um teste computacional que faz uso de um vídeo que não foi apresentado antes, para o comportamento específico de seguir faixas. Nessa imagem, estão dispostos o valor de esterçamento (ou velocidade angular) implementado pelo piloto e o sugerido pela rede, para a pequena amostra de vídeo em sequência temporal assim como capturada. É importante lembrar que os valores de esterçamento podem ir de -1 a 1 (-1 totalmente a direita e 1 totalmente a esquerda), mas como a situação envolve um caso de reta, então existe sentido a apresentação de pequenos valores, como ilustrados. A escala, na prática, corresponde a uma faixa de valores que vai de -90° até 90°, como na figura 4.3, para cada novo instante em $X^{'}$ e $Y^{'}$. Outra informação que também deve ser destacada é que a figura 6.5 não representa a posição do robô, e apenas ilustra uma comparação entre o que a rede faria caso estivesse no comando do robô e o que foi realizado pelo piloto, para um mesmo instante. As figuras a seguir, figura 6.7 e 6.8, que apresentam a relação de esterçamento seguem a mesma lógica da figura 6.5.

Da forma que foi apresentado, é possível ver que a rede sugere um valor muito próximo ao realizado pelo piloto, e em alguns momentos até antecipa a solução como visto nos ajustes feitos

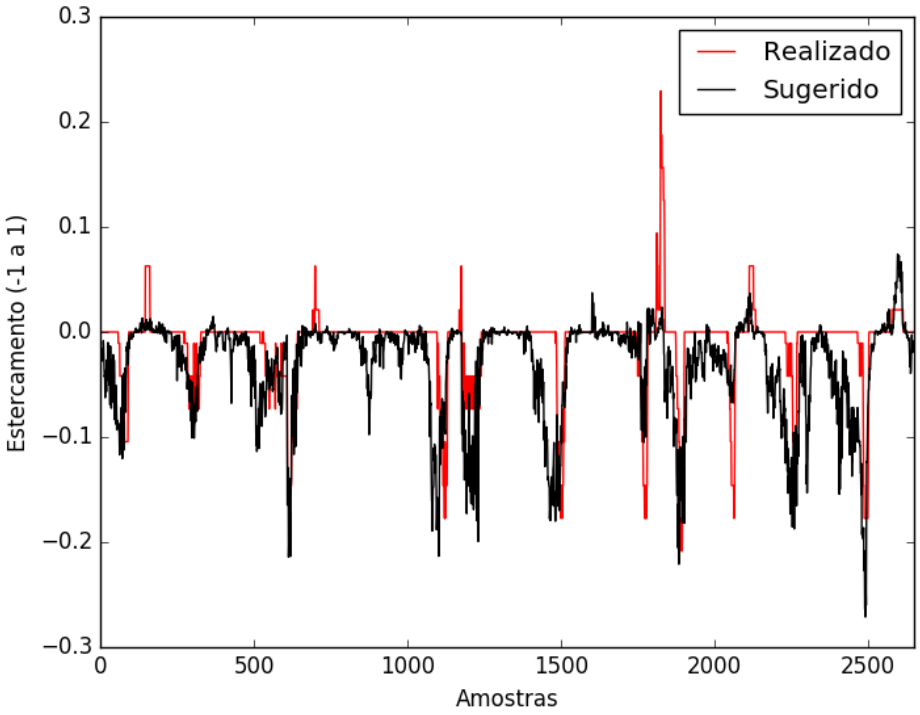


Figura 6.5: Esterçamento Realizado vs Sugerido (Retas)

para esterçamento negativos. Situações comuns na guiagem do robô, que foram anteriormente comentadas, são evidentes em toda a apresentação das amostras e em especial no pico de valor entre 1500 e 2000, quando ocorre um comando brusco realizado pelo piloto. Com isso, é visível que os problemas de desalinhamento das rodas do robô fazem com que o piloto seja forçado a movimentar-se com tendência contrária, uma vez que o desalinhamento puxa a máquina na direção esquerda (esterçamento positivo) e a tanto a rede como o piloto reajustam no sentido da direita (esterçamento negativo). O comando brusco apresentado foi fruto de um ajuste inadequado realizado durante a guiagem, o qual a rede viu como desnecessário e logo em seguida sugere que um esterçamento oposto seja feito para corrigir a rota.

6.2.2 Curvas Suaves

Tabela 6.2: Erro *NRMSE* Curvas Suaves

Comportamento	500 épocas	1000 épocas	2000 épocas
Retas	0.016532	0.014897	0.013232
Curvas Suaves	0.020964	0.018793	0.018236

A figura 6.6, apresenta a matriz de confusão para casos de curvas suaves, que da mesma forma que no caso anterior, apresenta a concentração de dados sobre ou nas proximidades da diagonal principal, tal como uma matriz identidade. Contudo é possível observar que nesse caso as amostras estão mais distribuídas ao longo da diagonal, como consequência da realização de curvas suaves;

e além disso continuam mantendo a relação de proximidade entre sugerido e realizado.

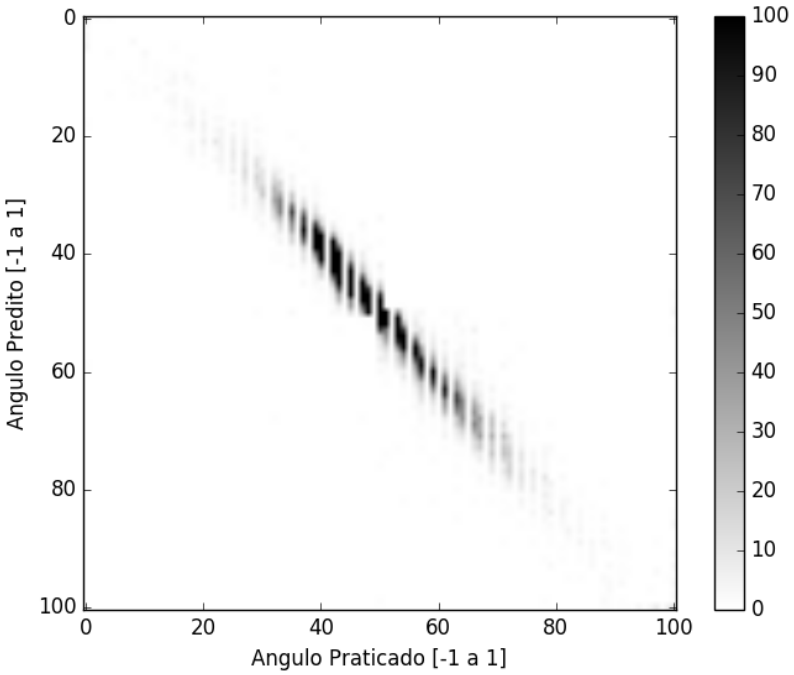


Figura 6.6: Matriz de confusão normalizada para validação de curvas suaves

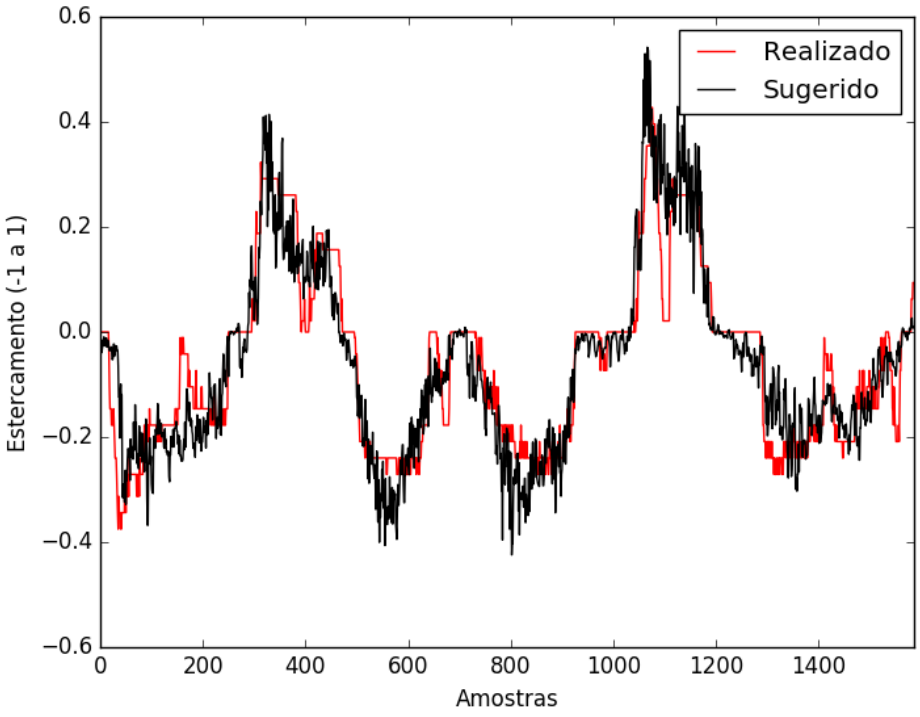


Figura 6.7: Esterçamento Realizado vs Sugerido (Curvas Suaves)

A figura 6.7, apresenta o gráfico resultante de um teste computacional tal como a figura 6.5, mas direcionado a ilustrar o comportamento específico de fazer curvas. É importante lembra que os valores de esterçamento podem ir de -1 a 1 (-1 totalmente a esquerda e 1 totalmente a direita), e para essa caso são apresentados valores maiores devido as curvas suaves.Como é apresentado na imagem, a rede sugere um valor muito próximo ao realizado pelo piloto, em ambas as direções, porém com uma variabilidade maior entre uma amostra e a imediatamente seguinte. Além disso, também foi possível observar que houveram situações onde a rede antecipou a solução, como por exemplo, entre as amostras 1200 e 1300.

O destaque nesse caso se deu pelo fato das sugestões feitas pela rede não acompanharem situações dadas como imprecisas na guiagem do robô, e refere-se a situação em que as curvas não foram realizadas com de maneira constante mas que não impediu a rede de apresentar uma solução mais coerente com a situação. Observe que entre no intervalo de 1000 a 1200 no gráfico, que representa uma curva para esquerda, o realizado pelo piloto (curva em vermelho) se dividiu duas ondas de comandos, contudo a rede apresenta persistência até a curva realmente ser finalizada.

6.2.3 Curvas Suaves em T

Tabela 6.3: Erro *NRMSE* Curvas em T

Comportamento	500 épocas	1000 épocas	2000 épocas
Retas	0.0189304	0.013593	0.012157
Curvas Suaves	0.024981	0.016758	0.015206
Curvas Suaves em T	0.026923	0.018191	0.016052

A figura 6.8, apresenta a matriz de confusão para casos de curvas suaves em T, e também apresenta uma concentração dos dados nas proximidades da diagonal principal, assim como os casos anteriores. Contudo é importante observar que houve mais ruído, com dados um pouco mais dispersos em relação a diagonal principal, e com uma leve acentuação da parte que se estende do centro até topo esquerdo do gráfico.

A figura 6.9, apresenta o gráfico resultante de um teste computacional para curvas suaves em T, e ilustra a tendência da rede em sugerir uma curva para direita quando diante de uma situação de bifurcação. Além disso, também foi possível observar a antecipação de comandos, como por exemplo, entre as amostras 0 e 100; e uma grande variabilidade entre as amostra, que pode ser fruto da apresentação de poucas amostras. De maneira geral a rede apresentou uma resposta razoável, se considerado que a mesma acerta o sentido que a curva deve ser feita, mas a alta variabilidade pode vir ser um problema quando implementado na prática, pois o modelo dinâmico de movimentação do robô e o sistema como um todo podem não acompanhar essas mudanças.

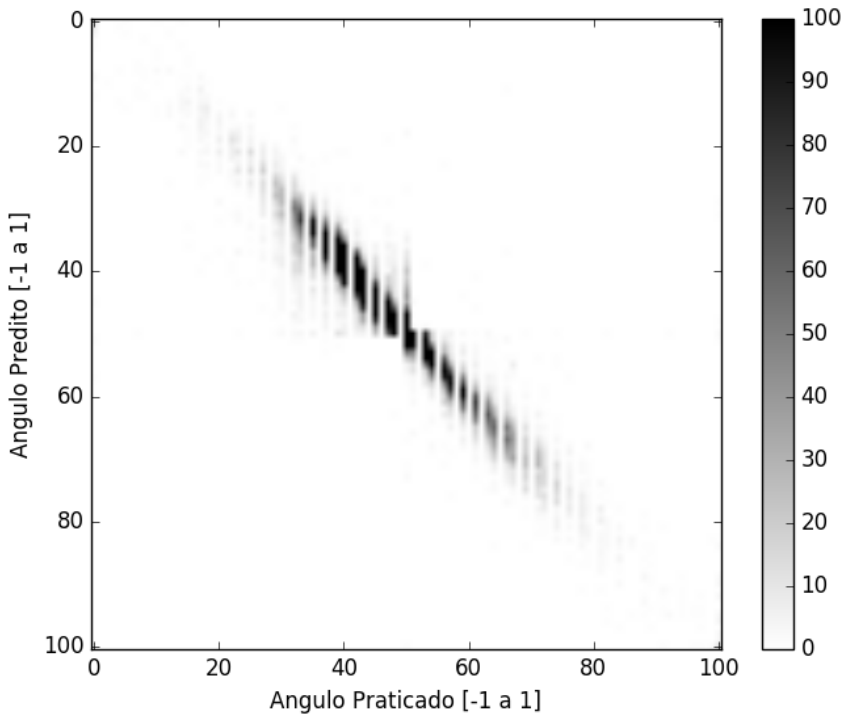


Figura 6.8: Matriz de confusão normalizada para validação de curvas suaves em T

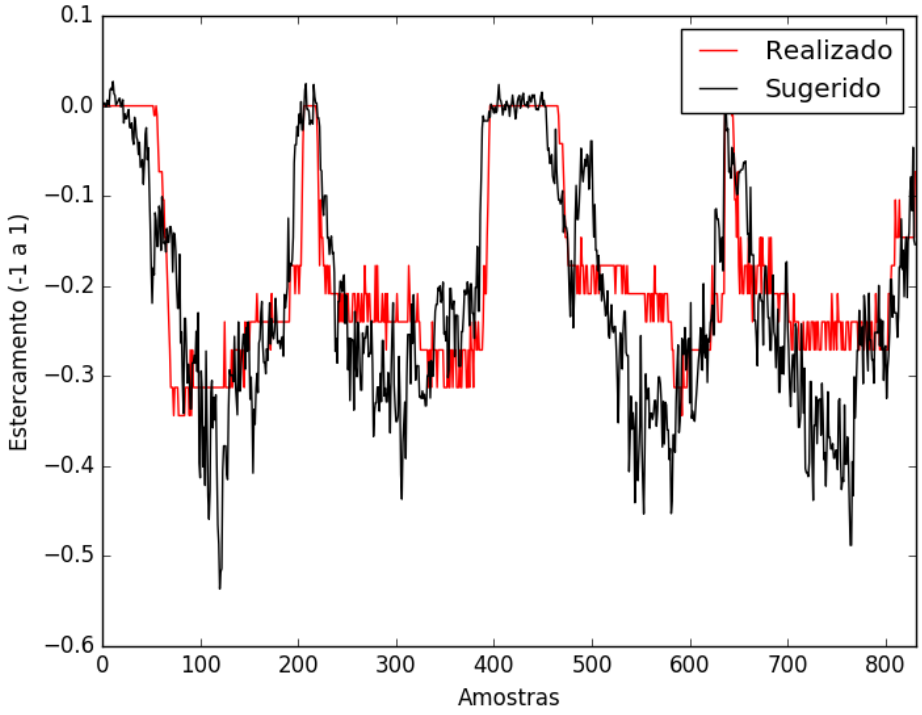


Figura 6.9: Esterçamento Realizado vs Sugerido (Curvas Suaves em T)

6.3 Testes Práticos

Após a série de avaliações numéricas que moldam o desempenho da rede, apresentadas pelas tabelas de erros normalizados(*NRMSE*) e pelos testes computacionais apresentados nos gráficos comparativos do esterçamento, uma abordagem prática que complementa a validação do trabalho foi implementada. A implementação prática consistiu da execução do controle de esterçamento diretamente realizado pela rede neural treinada, e assim foi possível avaliar como o modelo se apresenta na prática.

O robô foi inserido em um ambiente novo (térreo do CIC) e disposto sobre a orientação de cada rede respectiva ao seu comportamento, entretanto apenas os resultados de retas e curvas suaves foram promissores e apresentaram um bom desempenho. Em resumo, a rede que foi treinada para retas funcionou muito bem, inclusive em situações que não existiam paredes e nem portas ao redor, ambiente composto apenas de chão e guia de deficiente visual. A rede que foi treinada para curvas suaves, apresentou uma boa resposta para os teste realizados em retas, porém essa resposta oscilava bastante e impediu o uso da velocidade máxima; no caso dessa mesma rede apresentada para situações de curvas suaves, o robô conseguiu realizar as curvas mas nem todas as tentativas foram bem sucedidas, como por exemplo, curvas realizadas para esquerda.

Um vez que as implementações das redes treinadas para os dois primeiros comportamentos apresentaram uma resposta que se encaixou com a dinâmica de movimentação do robô, foram feitos gravações desses experimentos. Entretanto, dado a dificuldade de apresentar um resultado mais gráfico do que foi implementado na prática, considerando a linha temporal que permite dizer que o robô seguiu a reta ou mesmo fez uma curva, foi escolhido apresentar esse resultado na forma de um video, disponibilizado nos links:

- Retas: *https://vimeo.com/345595155*
- Retas e Curvas Suaves: *https://vimeo.com/345599445*

Para o caso da rede que engloba os 3 comportamentos(retas, curvas suaves e curvas em T), não foi possível obter um bom resultado prático que pudesse ser gravado, apesar de mostrar tendência em apontar um sentido correto para orientação. Como por exemplo, quando foram apresentadas situações de bifurcação a rede sugeriu que o robô fizesse uma curva a direita, porém a execução apresentava uma grande intensidade de curva a qual ocasionava imperfeições na imagem e posteriormente uma perda da referência da faixa.

6.3.1 Síntese

As tabela apresentadas ilustraram um panorama dos resultados obtidos para cada conjunto de comportamentos, e é evidente que os valores de erros apresentam valores baixos, não superando 5% de erro, embora em alguns testes práticos não tenha sido possível ver este excelente desempenho. Embora a composição de vários comportamentos ocasione uma variação maior do valor sugerido pela rede e possa ser mais instável na aplicação, como por exemplo a rede treinada com curvas

e retas em relação a rede treinada apenas com retas, o valor de erro normalizado(*NRMSE*) não apresentou muita alteração na comparação e ainda sim conseguiu generalizar a resposta ou mesmo indicar qual comportamento poderia ser mais adequado em cada situação. Uma outra informação relacionada as tabelas, envolvendo a dinâmica do treinamento que foi apresentada na figura 6.3, é a de que não houve sobre-ajuste, pois o valor do erro *NRMSE* relacionado a generalização (teste) da rede diminuiu com redes que foram treinadas com mais épocas.

Da análise dos gráficos que comparam o esterçamento da rede com o feito pelo piloto, é possível ver que o modelo começa a sugerir ajuste de esterçamento para realização de curva ou para se manter sobre a faixa, mesmo antes do piloto realizar a manobra. Isso foi observado visualmente na prática, quando a rede realiza o trajeto sem se distanciar muito da faixa apesar das situações em que o piloto falhou em manter o robô o mais próximo da linha.

Dada a dificuldade apresentada pelo modelo em fazer curvas intensas, apresentado por Arthur [32], uma hipótese inicial de curvas acentuadas até foi testada para a situação mas não apresentou resultados apreciáveis e optou-se por fazer as já explicadas curvas suaves no contexto desse trabalho, assim foi garantido melhor êxito no desempenho de curvas quando avaliado apenas o parâmetro *NRMSE*. Esse erro apresenta valores maiores quando em situações de curvas, onde a reprodução da mesma não é possível de ser realizada de forma igual em todas as amostras; e também é evidente nos momentos em que o piloto teve dificuldades em manter o robô o mais alinhado possível com a faixa de deficiente visual.

A implementação prática para retas e curvas foram um sucesso, entretanto não foi possível ter um resultado igualmente bom para a maioria das curvas feitas em bifurcações, o que pode ser explicado pelas variações bruscas nas imagens durante o esterçamento acentuado, que gera solavancos no movimento e deixa as imagens mais borradas, dificultando o trabalho da rede em sugerir valores mais corretos. Além disso é importante destacar que houveram dificuldades em termos do hardware do computador e na otimização do código de pilotagem prática, que fizeram a rede ser lenta na sua resposta e ocasionar um erro maior em curvas, mesmo quando a velocidade é reduzida ao mínimo. Alguns comportamentos emergentes que não eram esperados também foram observados, que foi o caso do robô conseguir desviar de bebedouros e voltar para a faixa, mesmo que estes não tenham sido apresentados com uma frequência muito grande nos dados coletados.

Por fim, para corroborar a hipótese de que a rede (treinada apenas com retas) fez uso da faixa para se orientar, foram apresentados situações em que não existiam portas e nem paredes (apenas as faixas e o chão) durante validação prática, e o comportamento foi executado corretamente.

Capítulo 7

Conclusões

Esse trabalho fez uso de uma mesma arquitetura de rede neural convolucional apresentada por *comma.ai*, que recebe uma imagem colorida e sugere um ângulo de esterçamento, mas nesse caso envolve uma aplicação robótica específica de paradigma reativo. Foram então apresentadas 3 situações diferentes, para as quais a rede treinada mimetiza o comportamento realizado por um piloto; resumidas em seguir uma faixa de deficiente, realizar curvas para o sentido disponível e realizar curvas para direita quando em uma bifurcação.

Todos os resultados, quando analisados computacionalmente, foram bastante animadores e não apresentaram muita discrepância, e dessa forma, a rede que só aprendeu o comportamento feito em retas, atuou com maior exatidão e precisão nos casos de retas; enquanto as demais redes apresentaram apenas uma boa exatidão. Entretanto, o modelo que apresentou melhor resposta prática consistiu na rede que só deveria seguir a faixas, dado a simplicidade dessa tarefa e pelo fato de possuir movimentos mais suaves. O baixo desempenho prático para as redes que trabalhavam com retas e com curvas deve-se a inserção de instabilidades que são provenientes da execução dos movimentos (coleta de dados) e da guiagem feita pelo controle *joystick* (amostragem reduzida do equipamento).

Apesar de não ser tão comum trabalhar com imagem para realização de comportamentos reativos, devido as dimensão dos dados, esse trabalho contempla com boas perspectivas a aplicação de uma inteligência artificial que tem o papel de sugerir uma resposta, além de compor/deliberar as possíveis saídas, para uma entrada de imagem; o que pode se encaixar como um paradigma robótica sofisticado. É importante destacar que esse trabalho não fez uso de representações de ambiente, memória ou qualquer realimentação de uma maneira direta; e mais importante ainda, que a mimetização é realizada apenas pelo que foi observado na atuação de um humano. Logo, se o objetivo de um comportamento reativo é mimetizar reações, e as reações que foram apresentadas pela rede treinada seguem a mesma tendência do que foi realizado por uma pessoa (*behaviorcloning*), então pode-se dizer que esse trabalho atingiu a meta de fazer um computador aprender a atuar reativamente para as situações que foram apresentadas.

7.1 Perspectivas Futuras

Dada que a composição de vários comportamentos inseriu uma imprecisão na saída da rede, uma boa abordagem de melhoria seria utilizar uma rede para cada comportamento em específico, tal como as regras reativas, e uma rede específica para a deliberação ou composição de comportamentos. Aquelas poderiam atuar adaptadas à cada caso e comportamento, enquanto estas, trabalhariam tal como um classificador.

Considerando as limitações impostas pelo uso de apenas uma câmera, uma alternativa poderia ser utilizar outras duas câmeras integrar a percepção do espaço, ou recortes da imagem que destacassem a faixa de orientação para deficiente visual, constituindo 2 ou 3 canais de uma mesma rede[32]. Caso contrário, ainda existiria a possibilidade de dispor redes que fizessem uso de outros sensores do robô, como por exemplo, sensor ultrassom e laser, desde que não proporcione um grande aumento no tempo de treinamento, o que só poderia ser visto com testes. Ambas soluções poderiam fornecer novos horizontes de implementação e abranger mais comportamentos reativos. Para as limitações que envolvem a coleta de dados, seria adequado o uso de um controle *joystick* com menor sensibilidade e maior resolução que o utilizado, dado a problemática que envolve a oscilação do comando de esterçamento adquirido (variações de até 5%).

Por fim, e não menos importante, uma implementação de mapas de atenção poderia evidenciar melhor quais partes da imagem foram utilizadas para motivar a reação apresentada.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] SMITH, C. *ml-linear-regression*. Disponível em: <https://github.com/crsmithdev/notebooks/blob/master/ml-linear-regression/ml-linear-regression.ipynb>.

[2] AGARWAL, A. *Polynomial Regression*. 2018. Disponível em: <https://towardsdatascience.com/polynomial-regression-bbe8b9d97491>.

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

[4] BAUCHSPIESS, A. *Apostila da disciplina de ICIN*. 2019. Disponível em: <http://www.ene.unb.br/adolfo/Lectures/ICIN/isi.pdf>.

[5] HAYKIN, S. *Neural networks and learning machines*. 3. ed. Prentice Hall, 2009. ISBN 0-13-147139-2. Disponível em: <http://dai.fmph.uniba.sk/courses/NN/haykin.neural-networks.3ed.2009.pdf>.

[6] LECUN, Y. et al. Gradient-Based Learning Applied to Document Recognition. *Proceedings of the IEEE*, v. 86, n. 11, p. 2278–2324, 1998.

[7] Dr. F. W. and J. Merlyn Clutterbuck. *Museu nacional da História americana*. Disponível em: <https://americanhistory.si.edu/collections/search/object/nmah_879329>.

[8] MATARIC, M. J. *The Robotics Primer*. MIT Press, 2007. 328 p. ISBN 9780262265508. Disponível em: <https://books.google.com.br/books?id=iZB9jwEACAAJ>.

[9] ROBOTS.ROS.ORG. *Pioneer 3-AT*. Disponível em: <https://robots.ros.org/pioneer-3-at/>.

[10] MobileRobots; Generation Robots. *Pioneer 3AT*. 2019. Disponível em: <https://www.generationrobots.com/en/402397-robot-mobile-pioneer-3-at.html>.

[11] POLAND/POZNAŃ, P. M. J. *Joystick movement directions;*. Disponível em: <https://en.wikipedia.org/wiki/File:JoystickDirections.svg>.

[12] CANON. *Câmera VC-C50i*. Disponível em: <https://www.bhphotovideo.com/c/product/313548-REG/Canon_0667B001_VC_C50i_1_4_Inch_CCD_Communication.html>.

[13] LAUGHSINTHESTOCKS. *Activation elu*. 2016. Disponível em: <https://upload.wikimedia.org/wikipedia/commons/b/bc/Activation_elu.svg>.

[14] DSA, E. *Deep Learning Book*. Disponível em: <http://deeplearningbook.com.br/deep-learning-a-tempestade-perfeita/>.

[15] Robin R. Murphy. *IntroductiontoAIRobotics*. MIT Press, 2000. 487 p. Disponível em: <http://www.profesaulosuna.com/data/files/ROBOTICA/ROBOTICS EBOOKS/Introduction to AI Robotics.pdf>.

[16] IFR. *Industrial robot sales increase worldwide by 31 percent*. Disponível em: <https://ifr.org/ifr-press-releases/news/industrial-robot-sales-increase-worldwide-by-29-percent>.

[17] William Grey Walter. A Machine That Learns. *Scientific American*, p. 60,61,62,63, 1951. Disponível em: <http://robotics.cse.tamu.edu/dshell/cs643/papers/walter51learns.pdf>.

[18] RobotCub Consortium. *an open source cognitive humanoid robotic platform*. 2009. Disponível em: <http://www.icub.org/>.

[19] Giorgio Metta, Giulio Sandini, David Vernon, Lorenzo Natale, F. N. The iCub humanoid robot: an open platform for research in embodied cognition. 2008. Disponível em: <http://www.robotcub.org/misc/papers/08_Metta_Sandini_Vernon_etal.pdf>.

[20] HONDA. *Asimo*. Disponível em: <https://asimo.honda.com/asimo-specs/>.

[21] MITCHELL, T. M. *Machine Learning*. McGraw-Hill, 1997. 432 p. ISBN 0070428077. Disponível em: <https://www.cs.ubbcluj.ro/ gabis/ml/ml-books/McGrawHill - Machine Learning -Tom Mitchell.pdf>.

[22] ACADEMY, D. S. *Deep Learning Book*. Disponível em: <deeplearningbook.com.br/>.

[23] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. [S.l.]: The MIT Press, 1969. ISBN 0-262-63022-2.

[24] McCulloch, W.S. & Pitts, W. Bulletin of Mathematical Biophysics. *Kluwer Academic Publishers*, v. 5, p. 115, 1943.

[25] ROSENBLATT, F. *The Perceptron, a Perceiving and Recognizing Automaton Project Para*. Cornell Aeronautical Laboratory, 1957. (Report: Cornell Aeronautical Laboratory). Disponível em: <https://books.google.com.br/books?id=P_XGPgAACAAJ>.

[26] ROSENBLATT, F. *THE PERCEPTRON: A PROBABILISTIC MODEL FOR INFORMATION STORAGE AND ORGANIZATION IN THE BRAIN*. 1958. 386–407 p. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.335.3398&rep=rep1&type=pdf>.

[27] RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. In: RUMELHART, D. E.; MCCLELLAND, J. L.; PDP Research Group, C. (Ed.). Cambridge, MA, USA: MIT Press, 1986. cap. Learning I, p. 318–362. ISBN 0-262-68053-X. Disponível em: <http://dl.acm.org/citation.cfm?id=104279.104293 https://web.stanford.edu/class/psych209a/ReadingsByDate/02_06/PDPVolIChapter8.pdf>.

[28] LIPPMANN, R. An introduction to computing with neural nets. *IEEE ASSP Magazine*, v. 4, p. 4–22, 1987.

[29] HASSABIS, D. *AlphaGo: using machine learning to master the ancient game of Go*. 2016. Disponível em: <https://blog.google/technology/ai/alphago-machine-learning-game-go/>.

[30] HUBEL, D. H. Exploration of the primary visual cortex, 1955–78. *Nature*, v. 299, n. 5883, p. 515–524, 1982. ISSN 1476-4687. Disponível em: <https://doi.org/10.1038/299515a0>.

[31] LECUN, Y. Generalization and Network Design Strategies. In: PFEIFER, R. et al. (Ed.). *Connectionism in Perspective*. Zurich, Switzerland: Elsevier, 1989.

[32] FERREIRA, A. E. T. *Estimação do ângulo de direção por Vídeo para Veículos Autônomos utilizando Redes Neurais Convolucionais Multicanais*. 67 p. Tese (Trabalho de Graduação) — Universidade de Brasília, 2017.

[33] Avron Barr; Edward A Feigenbaum. *The Handbook of Artificial Intelligence*. 1. ed. California, USA: William Kaufmann, Inc., 1981. Disponível em: <https://archive.org/details/handbookofartific01barr https://www.worldcat.org/title/handbook-of-artificial-intelligence-volume-1/oclc/1035139773>.

[34] ADEPTTECHNOLOGY. *Datasheet Pioneer3AT*. 2011. Disponível em: <https://www.generationrobots.com/media/Pioneer3AT-P3AT-RevA-datasheet.pdf>.

[35] Caracciolo, L.; de Luca, A.; Iannitti, S. Trajectory tracking control of a four-wheel differentially driven mobile robot. In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*. [S.l.: s.n.], 1999. v. 4, p. 2632–2638 vol.4. ISSN 1050-4729.

[36] MOBILEROBOTS. *Pioneer 3 Operations Manual*. 2006. 69 p. Disponível em: <https://www.inf.ufrgs.br/ prestes/Courses/Robotics/manual_pioneer.pdf>.

[37] COMMA.AI. *Exemplo do arquitetura de rede que foi usada na implementação do trabalho*. 2016. Disponível em: <https://github.com/commaai/research>.

[38] CLEVERT, D.-A.; UNTERTHINER, T.; HOCHREITER, S. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *CoRR*, abs/1511.0, 2016. Disponível em: <http://arxiv.org/abs/1511.07289>.

[39] BENGIO, Y.; LECUN, Y. (Ed.). *3rd International Conference on Learning Representations, {ICLR} 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. [s.n.], 2015. Disponível em: <https://iclr.cc/archive/www/doku.php%3Fid=iclr2015:accepted-main.html>.

[40] KOHAVI, R.; PROVOST, F. Confusion matrix. *Machine learning*, v. 30, n. 2-3, p. 271–274, 1998.

ANEXOS

Os códigos utilizados estão disponíveis em `https://github.com/TulioLima1502/TG` ou em `https://1drv.ms/u/s!AqilWUq_aRdSpa0ot3D6hFuErZ1-sA?e=dk7c4x`, também em anexo. O arquivo é uma cópia do repositório no github, e contém os códigos em Python que foram utilizados na implementação deste trabalho.