DISSERTAÇÃO DE MESTRADO

REDES NEURAIS ARTIFICIAIS APLICADAS EM APRENDIZAGEM DE TRAJETÓRIA EM ROBÓTICA MÓVEL

Jefferson Adiniz Borges Ferreira

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

REDES NEURAIS ARTIFICIAIS APLICADAS EM APRENDIZAGEM DE TRAJETÓRIA EM ROBÓTICA MÓVEL

Jefferson Adiniz Borges Ferreira

DISSERTAÇÃO SUBMETIDA AO DEPARTAMENTO DE ENGENHARIA MECÂNICA DA FACULDADE DE TECNOLOGIA DA UNIVERSIDADE DE BRASÍLIA COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM SISTEMAS MECATRÔNICOS

APROVADA POR:

_____

Prof. Dr. Daniel M. Muñoz Arboleda, PPMEC/UnB
Orientador

_____

Prof. Dr. Gerardo Idrobo Pizo, FGA/UnB
Membro Externo

_____

Prof. Dr. Rudi Henri van Els, PPMEC/UnB
Membro Interno

BRASÍLIA/DF, 18 de Dezembro de 2020

FERREIRA, J.A.B.

REDES NEURAIS ARTIFICIAIS APLICADAS EM APRENDIZAGEM DE TRAJETÓRIA EM ROBÓTICA MÓVEL / Jefferson Adiniz Borges Ferreira;

Orientador: Daniel M. Munõz Arboleda . -- Brasília, 2020.

Dissertação (Mestrado) – Universidade de Brasília – UnB
Faculdade de Tecnologia – FT
Programa de Pós-Graduação em Sistemas Mecatrônicos – PPMEC, 2020.

1. Robótica Móvel. 2. Aprendizado por demonstração. 3. Redes Neurais Artificiais. I. , Daniel M. Muñoz Arboleda, orientador. II. Universidade de Brasília. III. ENM/FT/UnB.

Dedicatória

Esse trabalho é dedicado às pessoas que pensam, trabalham e realizam.

Jefferson Adiniz Borges Ferreira

Agradecimentos

A Deus por ter criado toda a realidade e ter me feito capaz.

Aos meus pais, Adiniz e Mirian, por seu amor, apoio financeiro, mesmo em dificuldade, e principalmente pelo exemplo de vida.

Ao meu orientador, Daniel Mauricio Muñoz Arboleda, por ter me aceitado como aluno, ter sido sempre paciente, sereno e prestativo.

A Débora Janini e a Sulamita Ferreira, pelo apoio e as constantes revisões de português.

Ao meu amigo, Luiz Calixto pelas discussões e contribuições na construção do protótipo do robô.

A essa Universidade e seu corpo docente, que me instruíram durante minha trajetória. Aos meus amigos, colegas da faculdade e outras pessoas que direta ou indiretamente me auxiliaram nessa parte da minha caminhada, de coração, o meu muito obrigado.

Jefferson Adiniz Borges Ferreira

RESUMO

O estudo realizado neste trabalho tem como objetivo implementar técnicas de aprendizagem por demonstração usando redes neurais artificiais aplicadas em controle de trajetória de um robô com tração diferencial. Em particular, deseja-se imitar trajetórias que um usuário ensina ao robô móvel usando exclusivamente um controle reativo de desvio de obstácuos a partir da medição de distância com sensores ultrassônicos (entradas do modelo) e um controle de velocidade a partir das velocidades linear e angular que o usuário imprime ao robô (saídas desejadas).

Dois métodos de validação das técnicas de aprendizagem foram usadas neste trabalho, a primeira usando um robô virtual e a segunda um protótipo físico de baixo custo. Durante a validação por simulação, várias trajetórias foram realizadas no simulador EyeSim com auxílio da engine Unit3D. O simulador permite construir ambientes estruturados e compor uma base de dados com a informação dos sensores de medição de distâcia e as velocidades linear e angular do robô estimadas a partir do modelo cinemático e dos encoders das rodas. Já o protótipo físico foi construído com microcontroladores Arduino para um controle de baixo nível, assim como para realizar a comunicação via bluetooth para controle via aplicativo. Adicionalmente, foi utilizado para aquisição dos dados dos sensores ultrassônicos e dos encoders, assim como para armazenar os dados e enviar para nuvem. O robô físico foi treinado em ambientes estruturados estáticos e dinâmicos com obstáculo móvel.

Após a coleta dos dados, as redes neurais do tipo perceptron multicamadas foram treinadas no software Weka, usando o algoritmo de *backpropagatipon*. Depois das redes neurais serem treinadas, elas foram inseridas nos robôs (simulado e físico), observando o comportamento da trajetória realizada no processo de imitação e coleta de novos dados. Utilizando linguagem R foram feitas análises dos dados coletados nos experimentos simulados; foi utilizado também o Matlab, o qual realizou regressões lineares dos dados de treinamento, validação e teste das saídas entregues e esperadas pela rede. Esses resultados das regressões mostram o grau de eficiente na aprendizagem obtendo 99% de eficiência no robô simulado até 85% no robô físico.

Palavras-chave: Aprendizagem por Demonstração, Redes Neurais Artificiais, Perceptron Multicamadas, LfD e Robótica Móvel.

ABSTRACT

The study carried out in this work aims to implement demonstration learning techniques using artificial neural networks applied to control the trajectory of a robot with differential traction. In particular, the aim is to imitate the paths that a user teaches the mobile robot using exclusively a reactive control of deviation of obstacles from the distance medication with ultrasonic sensors (model inputs) and a speed control based on the linear and angular speeds that the user prints to the robot (desired outputs).

Two methods of validating the learning techniques were used in this work, the first using a virtual robot and the second a low-cost physical prototype. During the validation by simulation, several paths were performed in the EyeSim simulator with the aid of the Unit3D engine. The simulator allows you to build structured environments and compose a database with information from distance sensors and angular and linear velocities of the robot, estimated from the cinematic model and wheel encoders. The physical prototype was built with Arduino microcontrollers for low level control, as well as for communicating via bluetooth to a software application. Additionally, it was used to acquire data from ultrasonic sensors and encoders, as well as to store data and send it to the cloud. The physical robot was trained in static and dynamic structured environments with a mobile obstacle.

After data collection, multilayer perceptron neural networks were trained in the Weka software, using the backpropagatipon algorithm. After the neural networks were trained, the resulted neural models were programmed inserted in the robots (simulated and physical), observing the behavior of the path taken in the process of imitation and collection of new data. the R language as well as Matlab, the data collected in the simulated experiments were performed, which performed linear regressions of the training, validation and test data of the outputs delivered and expected by the network. These regression results show the degree of efficiency in learning, obtaining 99% efficiency in the simulated robot up to 85% in the physical robot.

Keywords: Demonstration Learning, Artificial Neural Networks, Multilayer Perceptron, LfD and Mobile Robotics

.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS E ABREVIAÇÕES

ρ Coeficiente de Correlação de postos de Spearman

τ Coeficiente de Correlação de postos de Kendall

FPGA: Field Programmable Gate Array

LfD: Learning from Demonstration

MLP: Neural Perceptron Multicamada

PbD: Programação de Robô por Demonstração

RBF: Função de Ativação de Base Radial

RNA: Rede Neural Artificial

SoC: System on Chip

Weka: Waikato Enverionment for Knowledge Analysis

Capítulo 1

Introdução

A aprendizagem de máquina é um ramo da inteligência artificial baseado na idéia de que sistemas podem aprender a identificar padrões e tomar decisões com o mínimo de intervenção humana. Neste estudo será utilizado aprendizado por demonstração, usando redes neurais em um protótipo de robô móvel construído para realizar trajetórias em ambientes controlados.

1.1 Definição do Problema

O aprendizado de máquina é um método de análise de dados que automatiza o desenvolvimento de modelos analíticos (HOSCH, 2019). É possível utilizar algoritmos para coletar dados de determinados comportamentos de uma máquina e apresentá-los para que a mesma possa imitar (aprender) o comportamento desejado. Adicionalmente, é possível ensinar a máquina a fazer uma estimação ou predição dos estados de funcionamento com base no histórico de dados. Os algoritmos usados em aprendizagem de máquina fazem o uso de dados de treinamento (armazenados numa base de dados) e, por meio de algoritmos de treinamento, conseguem "aprender" e prever situações com base na experiência adquirida.

Uma das técnicas de aprendizado de máquina é o LfD (do inglês *Learning from Demonstration*), em que um professor supervisiona todo o processo de aprendizado, dizendo o que é certo e o que é errado para determinada estimação ou predição. Neste método de aprendizado, a máquina é treinada através de um conjunto de dados onde, para cada entrada, a saída é conhecida (aprendizagem supervisionada). Neste caso, o algoritmo deve se ajustar para chegar aos resultados corretos com o máximo de acerto. Após o processo de treinamento, a máquina passa para o processo de imitação no qual recebe novos dados e imita o comportamento indicado pelo professor. Quando o aprendizado é constante, ou seja, o processo de treinamento é continuo, a máquina, à medida que o tempo passa, aumenta a experiência com aquele problema. Um dos desafios atuais da robótica móvel é fazer com que robôs de pequenas dimensões com restrições de desempenho computacionais consigam usar

técnicas de aprendizagem por demonstração devido à complexidade computacional dos algoritmos envolvidos no processo de treinamento. Existem 2 tipos de técnicas de LfD que podem realizar um processo de imitação de trajetória que são os de baixo nível e de alto nível. Nas técnicas LfD de baixo nível se ensinam padrões primitivos de movimento e o sistema aprende parâmetros desses padrões demonstrados usando inferência e análises estatísticas, modelos probabilísticos, técnicas de regressão não linear, processos Gaussinos e máquinas de vetores de suporte. Nas técnicas lfD de alto nível, além dos movimentos primitivos também são demonstradas tarefas complexas, constituídas de combinações de movimentos individuais.

O desafio aqui é extrair de tarefas complexas, padrões primitivos que possam facilitar o processo de treinamento. Enquanto na programação tradicional o robô tem que ser programado, exigindo do programador humano prever e codificar as possíveis situações e dividindo tudo em etapas e tarefas, o princípio de aprendizagem por demonstração é ensinar novas tarefas ao robô sem a necessidade de programação tradicional, dessa forma, simplificando o processo. No final do capítulo 2 será apresentada uma tabela de trabalho correlatos que serviu de base para essa pesquisa.

1.2 Justificativa

Uma das técnicas mais utilizadas de aprendizagem por demonstração são RNAs. RNAs são técnicas computacionais que apresentam um modelo matemático inspirado na estrutura neural de organismos inteligentes que adquirem conhecimento por meio da experiência, processo normalmente descrito como imitação da forma como os humanos aprendem (HAYKIN, 2001). As RNAs são amplamente usadas pela comunidade científica para resolver problemas de regressão de dados (aproximação universal de funções) e para problemas de classificação de padrões e clusterização.

No caso específico do LfD, as redes neurais artificiais podem ser usadas durante o processo de treinamento e aprendizagem, explorando a capacidade de aproximação de funções para modelar matematicamente o comportamento de um sistema. Essa flexibilidade da utilização de RNA’s para LfD, modificando os pesos sinápticos, tem a competência de gerar a saída desejada com uma maior capacidade de adaptação (Os pesos sinápticos são números, positivos ou negativos, dando a força, ou eficácia, do acoplamento sináptico entre um neurônio que envia o sinal e o neurônio que o recebe.).

As vantagens de se usar uma técnica de aprendizagem supervisionada em robótica móvel é a capacidade de utilizar as RNA's que permitem explorar comportamentos emergentes no processo de aprendizagem. Neste trabalho, será usada uma rede neural perceptron multicamada (MLP) para um problema de LfD no qual o robô móvel deve aprender a imitar trajetórias em ambientes diversos de forma que possa ser avaliada a capacidade de adaptação do sistema quando inserido em ambientes desconhecidos.

1.3 Objetivos

1.3.1 Objetivo Geral

Aplicar algoritmos de LfD baseados em redes neurais artificiais perceptron multicamadas para aprendizagem de trajetória em um robô móvel. Em particular será usado um robô móvel de pequenas dimensões com tração diferencial, construído para coletar dados de distância e de controle de velocidade dos motores, assim como validar o desempenho das RNA's no processo de aprendizagem de trajetórias.

1.3.2 Objetivos Específicos

Para atingir o objetivo geral deste trabalho foram estabelecidos os objetivos específicos a seguir:

- Construção de um ambiente de simulação do robô móvel de tração diferencial no software EyeSim usando ambientes estruturados para coleta de dados.
- Construção de um protótipo de robô móvel de tração diferencial com capacidade de aquisição de dados de distâncias, movimentação controlada e armazenamento dos dados dos sensores e velocidade de cada roda.
- Obter modelos baseados em redes neurais artificiais que permitam imitar trajetórias apresentadas por um usuário a partir de dados de distância e velocidade dos motores.

1.4 Aspectos Metodológicos

A metodologia usada no desenvolvimento do presente trabalho é composta das seguintes etapas:

- Primeira etapa: uma pesquisa e revisão bibliográfica do estado da arte sobre aprendizagem de máquina, *Learning from Demonstration*, e redes neurais artificiais;
- Utilização de softwares de simulação de trajetória do robô móvel de tração síncrona em ambiente fechado, coletando dados de distância e velocidade com EyeSim;
- Terceira etapa: Construção de um protótipo de robô móvel de tração síncrona utilizando a plataforma Arduino com capacidade de aquisição de dados de distâncias, movimentação controlada e armazenamento dos dados dos sensores e velocidade de cada roda para demonstração de trajetórias;
- Quarta etapa: realização do treinamento das redes, utilisando softwares como Weka e linguagem R, para obter um modelo baseado em redes neurais artificiais, que permita imitar trajetórias apresentadas por um usuário a partir de dados de distância e velocidade dos motores.
- Quinta etapa: Os pesos sinápticos resultantes de experimentos em diversos cenários são salvos em txt e carregados em algoritimo de imitação nas suas respectivas topologias.
- Sexta etapa: Após os robôs simulados e físicos imitarem a trajetória, o conjunto de dados dos robôs são carregados para testes não paramétricos e utilizando o Matlab, por meio da ferramenta nntool, é feito uma regressão linear da saída resultante, validação e teste das redes em suas respectivas topologias.

1.5 Contribuições do Trabalho

1- Desenvolvimento de uma plataforma de um robô móvel de baixo custo, com capacidade de imitação de trajetórias através de redes neurais artificiais perceptron multicamadas e comunicação sem fio com dispositivos móveis.

2- O aspecto metodológico de implementação de etapas e algoritmos simplificados de LfD em um robô físico de baixa desempenho computacional

3- Originou uma publicação no Congresso de Engenharias e Tecnologias do Centro Oeste – CET 2019 o artigo “Aprendizagem por demonstração usando redes neurais artificias em robótica móvel”.

1.6 Organização do Documento

O presente trabalho apresenta-se da seguinte maneira: O **capítulo 1**, como visto, é uma introdução geral a respeito do problema abordado, a justificativa deste, objetivos e aspectos metodológicos que foram desenvolvidos, além da contribuição do mesmo. O **capítulo 2** apresenta a fundamentação teórica a qual explicita os conceitos de robótica móvel, incluindo o robô de tração diferencial, o conceito de aprendizagem por demonstração (LfD), de RNA’s, de perceptron multicamadas, de *backpropagtion,* explicita também o desenvolvimento do controle e trabalhos correlatos. O **capítulo 3** descreve a metodologia abordada para o desenvolvimento do projeto, explicando o ambiente de simulação e as fases de demonstração, coleta de dados, treinamento e aprendizado, além de um protocolo experimental da pesquisa e o processo de construção do protótipo do robô. O **capítulo 4** apresenta os resultados dos sete experimentos desenvolvidos, sendo 3 simulados e 4 com o robô físico em ambientes diversos. Já o **capítulo 5** expõe a conclusão do trabalho.

Capítulo 2

Fundamentação Teórica

Nesse capítulo são apresentados conceitos sobre robótica móvel, aprendizagem de máquina, especificamente técnicas de aprendizagem por demonstração e sua implementação, em seguida são apresentados conceitos gerais sobre redes neurais artificiais, rede neural tipo perceptron múltiplas camadas e o algoritmo de treinamento backpropagation.

2.1 Robótica Móvel

A Robotic Industries Association (RIA) define um robô como sendo um manipulador programável multifuncional capaz de mover materiais, partes, ferramentas ou dispositivos específicos através de movimentos variáveis programados para realizar uma variedade de tarefas (DA SILVA, 2003) Entretanto, esta definição descreve toda uma categoria de máquinas, e sendo assim, qualquer equipamento capaz de ser programado é considerado um robô. Durante décadas a indústria vem utilizando esta tecnologia como substituta da mão-de-obra humana, em especial onde há riscos envolvidos. Os robôs utilizados pela indústria são, de um modo geral, grandes, fixos, capazes de efetuar tarefas como a manipulação de objetos em sua área de trabalho, desprovidos de inteligência de alto nível e não-autônomos (necessitam da interação com os seres humanos). Em resumo, poderiam ser classificados de máquinas-ferramentas.

Como o próprio nome indica, os robôs móveis são muito mais versáteis, pois não precisam estar fixados a uma célula de trabalho, podendo ser utilizados em tarefas onde não existam limites geográficos, movimentando-se por meio de pernas, lagartas ou rodas, dotados de autonomia ou não, capazes de serem programados em alto nível via software e de sentir o ambiente a sua volta através de seus sensores.

2.1.1 Robô de Tração Diferencial

Neste trabalho os robôs simulados e físicos usam a topologia de tração diferencial, com uma configuração típica de 3 rodas, uma boba na frente e duas traseiras posicionadas em triângulo; esta topologia tem como característica que a tração diferencial estabelece sua posição controlando a velocidade e a orientação das rodas traseiras. A figura abaixo mostra o diagrama de um robô de tração diferencial e suas variáveis para cálculo de posição e velocidade.

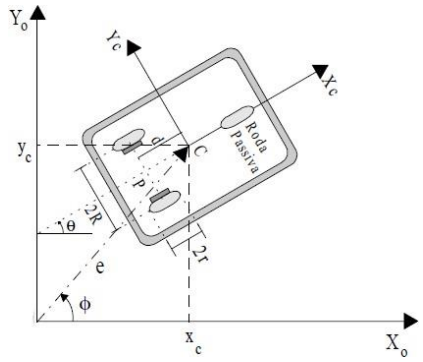

Figura 2.1: Robô móvel de tração diferencial em sistemas de coordenadas retangulares.
Fonte: Júnior & Hemerly (2003).

As variáveis a seguir são referentes aos cálculos de velocidade e trajetória do robô, considerando a figura 2.2:

- *r*: o raio da roda;
- *d*: distância entre as rodas motrizes;
- $ticks_{per.rev}$: número de marcadores do codificador (furos nos enconders) para uma rotação completa da roda;
- $ticks_L$: número de ticks durante a medição no encoder esquerdo;
- $ticks_R$: número de ticks durante a medição no encoder direito.

Os valores de S_L e S_R, que são as distâncias percorridas pela roda esquerda e direita, são determinadas em metros. Para o cálculo das distâncias em metros, os ticks medidos são divididos pelo número de ticks por rotação ($ticks_{per.rev}$) e multiplicados pela circunferência da roda.

$$S_L = \frac{2\pi r \times ticks_L}{ticks_{per.rev}} \quad (2.1)$$

$$S_R = \frac{2\pi r \times ticks_R}{ticks_{per.rev}} \quad (2.2)$$

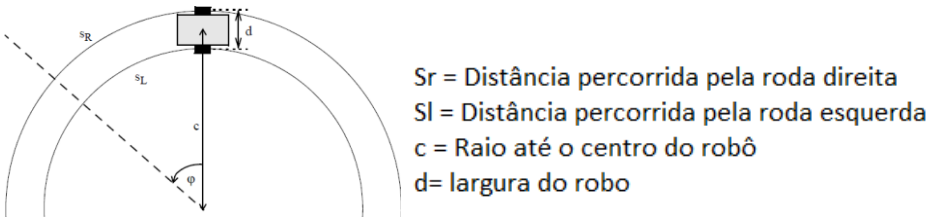

Figura 2.2: Cálculo para trajetória de um robô de tração diferencial. Fonte: Braünl (2006)

A distância que o veículo percorreu é dada por:

$$S = \frac{S_L + S_R}{2} \quad (2.3)$$

Essa equação 2.3 funciona para um robô avançando, retrocedendo ou fazendo uma curva em um local. Ainda se precisa saber a rotação do robô ϕ pela distância pecorrida.

Supondo que o robô siga um segmento circular pela esquerda, S_L e S_R podem ser definidos como a parte percorrida de um círculo completo (em radianos) multiplicada pelo giro de cada raio da roda. O raio de "viragem" do centro do veículo é "*c*", o raio de giro da roda

direita é: (c + d)/2, enquanto o raio de giro de a roda esquerda é: (c - d)/2. Ambos os círculos têm o mesmo centro.

$$S_R = \frac{\phi \times (c + d)}{2} \tag{2.4}$$

$$S_L = \frac{\phi \times (c - d)}{2} \tag{2.5}$$

Subtraindo ambas equações elimina-se o "c" e se tem:

$$S_R - S_L = \phi \times d \tag{2.6}$$

Isolando ϕ finalmente resulta em:

$$\phi = \frac{S_R - S_L}{d} \tag{2.7}$$

Se forem usadas as velocidades V_L e V_R em vez das distâncias S_L e S_R e usando θ'_L e θ'_R que são as rotações por segundo com o raio "r" das rodas direita e esquerda resulta em:

$$V_R = 2\pi \times \theta'_R \tag{2.8}$$

$$V_L = 2\pi \times \theta'_L \tag{2.9}$$

A fórmula que especifica as velocidades de um robô com tração diferencial agora pode ser expresso como uma matriz. Isso é chamado de matriz de equação cinemática direta:

$$\begin{pmatrix} v \\ w \end{pmatrix} = \begin{pmatrix} \frac{1}{2} & \frac{1}{2} \\ \frac{-1}{d} & \frac{1}{d} \end{pmatrix} \times \begin{pmatrix} \theta'_L \\ \theta'_R \end{pmatrix} \tag{2.10}$$

Sendo:

- *v*: velocidade linear do robô (igual a ds/dt);
- *w:* velocidade de rotação do robô (iguals a dϕ/dt ou ϕ);
- θ'_L e θ'_R: velocidades indivduais da rodas em rotações por segundo;
- *r:* raio da roda;
- *d:* distância entre as duas rodas

Outra forma de escrever é fazendo a equação cinemática inversa que pode ser escrita:

$$\begin{pmatrix} \theta'_L \\ \theta'_R \end{pmatrix} = \left(\frac{1}{2\pi r}\right) \begin{pmatrix} 1 & \frac{-d}{2} \\ 1 & \frac{d}{2} \end{pmatrix} \times \begin{pmatrix} v \\ w \end{pmatrix} \qquad (2.11)$$

Essas equações são usadas para calcular a velocidade angular e linear do robô com base nos dados adquiridos pelos sensores de distâncias.

2.2 Aprendizagem por Demonstração (LfD)

Aprendizagem por demonstração ou, do inglês, *Learning from demonstration* (LfD) surgiu aplicada à robótica pela primeira vez em 1980, usando a chamada Programação de Robô por Demonstração (PbD), também conhecido como "Aprendizagem por Imitação", a qual é um paradigma para permitir que robôs executem novas tarefas de forma autônoma. Ao invés de exigir que os usuários decomponham analiticamente e programem manualmente um comportamento desejado, o trabalho em LfD – PbD leva em consideração que um controlador de robô apropriado pode se derivado de observações do próprio desempenho humano.

No contexto de robótica móvel, a principal ideia do LfD é que, em um cenário de programação tradicional, um programador humano teria que raciocinar antecipadamente e codificar um controlador que seja capaz de responder a qualquer situação que o robô possa enfrentar. Esse processo pode envolver a divisão de uma tarefa em centenas de etapas diferentes e testar minuciosamente cada uma delas. Sendo assim, o princípio de aprendizagem por demonstração é ensinar novas tarefas a robôs sem a necessidade de programação, dessa forma, simplificando o processo. Dentre as técnicas usadas em LfD encontram-se as técnicas de aprendizado supervisionado e não supervisionadas.

As abordagens atuais para codificar habilidades por meio de LfD podem ser amplamente divididas entre duas tendências: uma representação de baixo nível da habilidade, assumindo a forma de um mapeamento não linear entre informações sensoriais e motoras e uma representação de alto nível da habilidade que decompõe a habilidade em uma sequência de unidades de ação-percepção.

Na aprendizagem de baixo nível movimentos ou ações individuais podem ser ensinados separadamente, em vez de todos de uma vez. O professor humano então forneceria um ou mais exemplos de cada sub-movimento além dos outros (guiar com a mão o robô por exemplo). Se o aprendizado procede da observação de uma única instância do movimento ou ação, chama-se esse aprendizado de one-shot learning (WU & DEMIRIS, 2010). Para aprender padrões de locomoção diferente do simples registro e reprodução, o controlador recebe conhecimento prévio na forma de padrões de movimento primitivos e aprende parâmetros para esses padrões na demonstração. (NAKANISHI et al, 2004). A aprendizagem mutli-shot pode ser realizada em lote após a gravação de várias demonstrações ou incrementalmente à medida que novas demonstrações são realizadas (LEE & OTT, 2011). O aprendizado geralmente realiza inferência da análise estatística dos dados nas demonstrações, onde os sinais são modelados por meio de uma função de densidade de probabilidade e analisados com várias técnicas de regressão não linear decorrentes do aprendizado de máquina. Atualmente, os métodos populares incluem Processo Gaussiano, Modelos de Mistura Gaussiana, Máquinas de Vetor de Suporte.

Na aprendizagem de alto nível, aprender tarefas complexas, compostas de uma combinação e justaposição de movimentos individuais é o objetivo final desse tipo de LfD. Uma abordagem comum é primeiro aprender modelos de todos os movimentos individuais, usando demonstrações de cada uma dessas ações individualmente (DANIEL et al, 2012) e, em seguida, aprender a sequência ou combinação correta em um segundo estágio, observando um humano executando toda a tarefa (SKOGLUND et al, 2007) ou por meio do aprendizado por reforço (MÜLING et al, 2013). No entanto, essa abordagem assume que existe um conjunto

conhecido de todas as ações primitivas necessárias. Para tarefas específicas, isso pode ser verdade, mas até o momento não existe um banco de dados de ações primitivas de uso geral, e não está claro se a variabilidade do movimento humano pode realmente ser reduzida a uma lista finita.

Uma alternativa é assistir o ser humano executar a tarefa completa e segmentar automaticamente a tarefa para extrair as ações primitivas (que podem então se tornar dependentes da tarefa) (KULIC et al, 2012). A principal vantagem é que as ações primitivas e a maneira como elas devem ser combinadas são aprendidas de uma só vez. Uma questão que surge é que o número de tarefas primitivas é frequentemente desconhecido, e pode haver várias segmentações possíveis que devem ser consideradas (GROLLMAN & JENKINS, 2010).

2.2.1 Aprendizado não Supervisionado

A aprendizagem não supervisionada ou aprendizagem sem um professor se caracteriza por não possuir exemplos ou demonstrações da função a ser aprendida pelo sistema. Esse tipo de aprendizagem é dividida em aprendizagem por reforço e aprendizagem auto organizada (HAYKIN, 2001).

Para MITCHEL (1997), a aprendizagem por reforço é o estudo de como um agente autônomo, que percebe e atua no seu ambiente, pode aprender a escolher ações adequadas a fim de atingir os seus objetivos. KAELBLING et al (1996) cita que aprendizagem por reforço preocupa-se com agentes que aprendem um determinado comportamento por meio de interações de tentativa e erro em um ambiente dinâmico. Em termos gerais, aprendizagem por reforço baseia-se na ideia de que a tendência, por parte de um agente, a produzir uma determinada ação deve ser reforçada se ela produz resultados favoráveis e enfraquecida se ela produz resultados desfavoráveis.

Em um sistema de aprendizagem por reforço, o estado do ambiente é representado por um conjunto de variáveis (espaço de estados) percebidas pelos sensores do agente. Uma ação escolhida pelo agente muda o estado do ambiente e o valor desta transição de estados é passado ao mesmo por meio de um sinal escalar de reforço, denominado recompensa. O objetivo do método de levar o agente a escolher a sequência de ações que tendem a aumentar a soma dos valores de recompensa. O agente move-se autonomamente no espaço de estados interagindo com o ambiente e aprendendo sobre ele por meio de ações por experimentação. Cada vez que o agente realiza uma ação, uma entidade externa treinadora (alguns autores referem-se a ela

como um professor (SUTTON & BARTO, 1998) ou ainda como crítico (KAELBLING et al, 1996)) ou mesmo o ambiente pode lhe dar uma recompensa ou uma penalidade, indicando o quão desejável é alcançar o estado resultante. Desta forma, o reforço nem sempre significa avanço como pode também inibir o agente em relação à ação executada. A figura 2.3 abaixo representa um esquema genérico da ideia de aprendizagem por reforço.

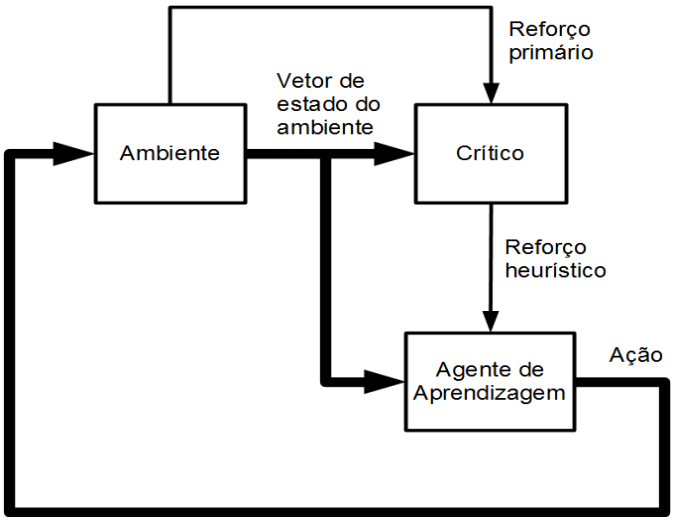

Figura 2.3: Representação genérica da ideia de aprendizagem por reforço. Fonte: SUTTON & BARTO (1998).

A aprendizagem por reforço é uma técnica que pode ser feita em tempo real ou on-line, ou seja, em constante adaptação, não necessitando de um treinamento prévio ou off-line do sistema. Essa vantagem torna-se importante no intuito de se aplicar esse método a sistemas robóticos que navegam em um ambiente incerto. Conforme KAELBLING (1996), em aprendizagem por reforço, não existe apresentação de conjuntos de exemplos.

A aprendizagem por reforço é diferente de aprendizagem supervisionada, que é o tipo de aprendizagem estudado na maioria das pesquisas atuais em aprendizagem de máquina e reconhecimento de padrão estatístico, como será apresentado na seguinte seção.

2.2.2 Aprendizado Supervisionado

Neste tipo de aprendizagem existe um “professor” que avalia a resposta da rede ao padrão atual de entradas. Este trabalho foi desenvolvido com aprendizado supervisionado; é, então, importante não confundir com o aprendizado por reforço em que não existe apresentação de conjuntos de exemplos e não necessita de treinamento prévio. No aprendizado supervisionado as alterações dos pesos são calculadas de forma a minimizar o erro entre a resposta da rede e a resposta desejada. Este tipo de aprendizagem normalmente é utilizado para treinar rede feedforward, um tipo de rede neural sem realimentação com neurônios agrupados em camadas, devido à facilidade de avaliar o desempenho da mesma para um determinado estado do sistema. O processo de aprendizagem se divide nas seguintes etapas.

A primeira etapa é chamada de demonstração. Assume-se que o professor possui conhecimento sobre o ambiente, que é representado por um conjunto de exemplos de entrada-saída, sendo que esse mesmo ambiente é desconhecido pelo robô e consequentemente pela rede neural. (HAYKIN, 2001). O professor, baseado em seu conhecimento sobre o ambiente, possui a capacidade de fornecer uma resposta à rede neural, que representa a ação ótima a ser realizada pela RNA de acordo com o vetor de treinamento.

Na próxima etapa é feito a aprendizagem supervisionada. Os parâmetros da rede neural artificial são ajustados baseados no vetor de treinamento e no sinal de erro, que é definido pela diferença entre a resposta desejada e a resposta real da rede.

A última etapa é a imitação. Quando a condição de aprendizagem completa é estabelecida, significa que a rede neural terminou seu processo de treinamento. Dessa forma, o auxílio do professor não é mais necessário, e a rede neural é capaz de reconhecer e lidar com o ambiente sem a necessidade de ajuda externa. A esse processo é dado o nome de Imitação, neural artificial é capaz de responder ao ambiente de forma onde a rede completamente independente, de acordo com o que foi ensinado e esperado na fase de aprendizagem supervisionada.O diagrama de blocos da figura 2.4 representa as etapas da aprendizagem supervisionada.

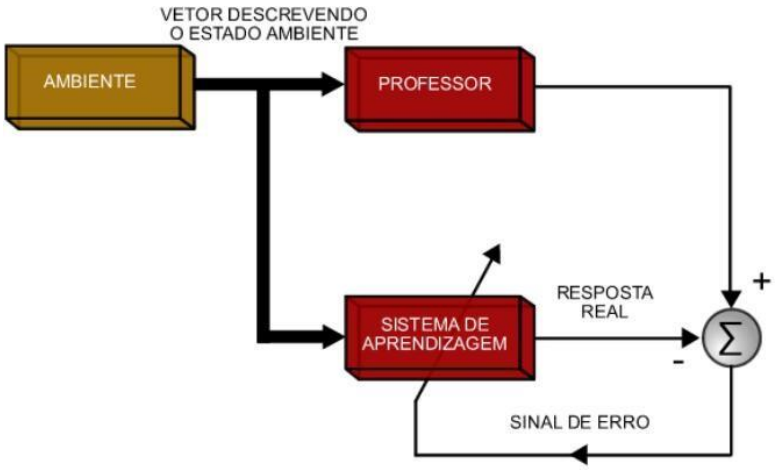

Figura 2.4: Diagrama de blocos da aprendizagem supervisionada. Fonte: HAYKIN (2001), adaptado.

2.3 Redes Neurais Artificiais (RNA's)

Uma rede neural artificial pode ser definida como "uma coleção de processadores paralelos conectados em forma de um grafo dirigido e organizado de modo que esta estrutura se adapte ao problema considerado". (FREEMAN & SKAPURA, 1991)

Segundo BITTENCOURT (1998), algumas das características que tornam a metodologia de redes neurais uma abordagem interessante com vistas à solução de problemas são as seguintes:

- Capacidade de aprender por meio de exemplos e de generalizar esta aprendizagem de maneira a reconhecer instâncias similares que nunca haviam sido apresentadas como exemplo;
- Elevada imunidade ao ruído, isto é, o desempenho de uma rede neural não entra em colapso em presença de informações falsas ou ausentes, embora sofra uma piora gradativa;
- Não requer conhecimento a respeito de eventuais modelos matemáticos dos domínios de aplicação;

- Apresentam bom desempenho em tarefas mal definidas, onde falta o conhecimento explícito sobre como encontrar uma solução. Do ponto de vista de se trabalhar com agentes autônomos, explorando um ambiente desconhecido, esta característica torna-se de extrema importância e justifica uma das razões para o uso de redes neurais nesse problema específico.

A origem da teoria das redes neurais remota aos modelos matemáticos de neurônios biológicos. Santiago Ramón y Cajal postulou sobre a comunicação das células nervosas pela sinapse e que a interconexão entre os neurônios não seria feita ao acaso e sim seria altamente específica e estruturada. A descrição de Cajal das estruturas cerebrais voltou as pesquisas para o desconhecido campo das estruturas formadas por grupos de neurônios (BARRETO, 2001). Esses estudos, deram início, de certa forma, à visão das redes neurais como um conjunto de estruturas, o que se traduz na primeira noção básica pressuposta para a utilização da inspiração biológica na computação de redes de células paralelas.

Muito simplificadamente, um neurônio biológico é formado por um corpo celular (soma), o qual contém o núcleo da célula, por diversos dendritos, pelos quais os impulsos elétricos são recebidos e por um axônio, aonde os impulsos elétricos são enviados. Os neurônios são interligados por meio de regiões eletroquimicamente ativas denominadas sinapses; são esses pontos de contato que permitem a propagação dos impulsos nervosos de uma célula a outra. As sinapses podem ser excitatórias ou inibitórias. "As sinapses excitatórias, cujos neuro-excitadores são os íons sódio, permitem a passagem da informação entre os neurônios e as sinapses inibitórias, cujos neuro-bloqueadores são íons potássio, bloqueiam a atividade da célula, impedindo ou dificultando a passagem da informação" (ROISENBERG, 2001).

Embora as redes neurais artificiais tenham inspiração em neurônios biológicos, é importante lembrar que suas semelhanças são mínimas. A intenção, originalmente, era imitar a realidade biológica. Mesmo que muitos pesquisadores atuais ainda discordem disso, a pesquisa atual é motivada por dois fatores citados a seguir por BARRETO (2001):

- Modelar o sistema nervoso com precisão suficiente de modo a se poder observar um comportamento emergente, que, sendo este, semelhante ao comportamento de um ser vivo modelado, possa servir de apoio às hipóteses usadas na modelagem;

- Construir computadores com alto grau de paralelismo.

2.3.1 Características Gerais de uma Rede Neural

Uma rede neural artificial é composta por várias unidades de processamento. Essas unidades, os neurônios, estão associadas a determinado peso. As unidades fazem operações apenas sobre seus dados locais, que são entradas recebidas pelas suas conexões. As interações entre os neurônios geram o comportamento inteligente da rede.

O neurônio artificial é uma estrutura lógico matemática que procura simular a forma, o comportamento e as funções de um neurônio biológico. Assim sendo, os dendritos foram substituídos por entradas, cujas ligações com o corpo celular artificial são realizadas através de elementos chamados de peso (simulando as sinapses). Os estímulos captados pelas entradas são processados pela função de soma e o limiar de disparo do neurônio biológico foi substituído pela função de transferência. (TAFNER, 1999). O esquema representativo de um neurônio artificial pode ser visto na figura 2.5 a seguir:

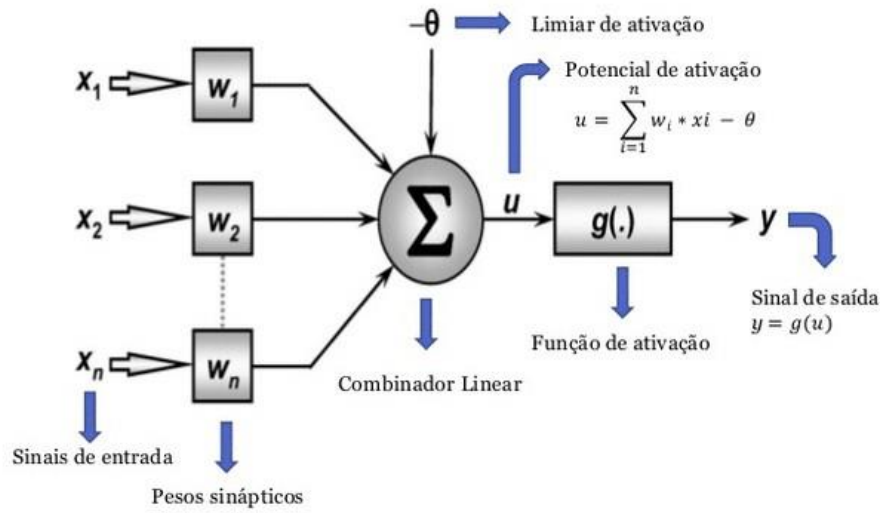

Figura 2.5: Esquema representativo de um neurônio artificial. Fonte: McCulloch & Pitts (1943), adaptado.

A operação de uma unidade de processamento, proposta por McCullock e Pitts em 1943, pode ser resumida da seguinte maneira: cada sinal de entrada é multiplicado por um peso, que indica a sua influência na saída da unidade. Se a soma ponderada dos sinais produzidos exceder a um certo limite (threshold), a unidade produz uma determinada resposta de saída.

Uma arquitetura neural é tipicamente organizada em camadas, estas, interligadas. Por definição, uma camada é um conjunto de neurônios recebendo as entradas em um mesmo local e tendo as saídas em um mesmo local. (BARRETO,2001).

Neste trabalho, foi usado a topologia de redes recorrentes, que são as redes cíclicas, com realimentação. Ou seja, a saída do processamento atual é novamente aplicada no processamento seguinte, funcionando como uma nova entrada. Um exemplo de rede recorrente está explicitado na figura 2.6.

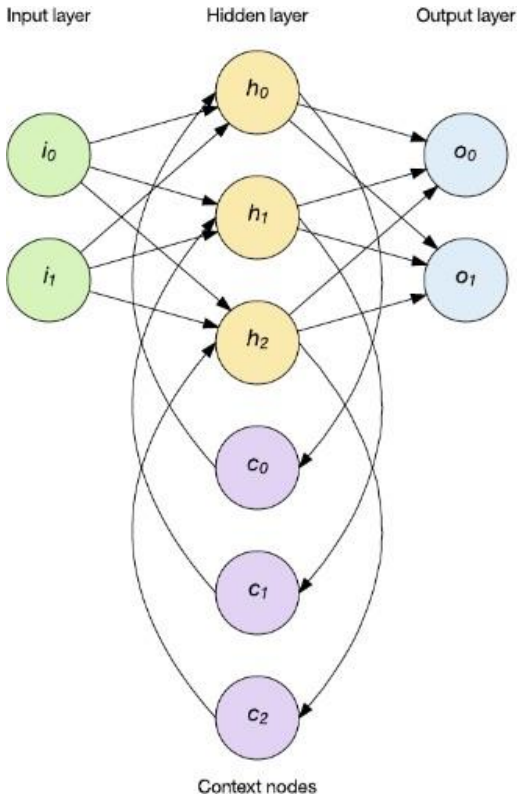

Figura 2.6: Exemplificação de uma arquitetura de rede neural recorrente de Jordan 1997. Fonte: JONES, (2017).

2.3.2 Perceptron Multicamadas (MLP)

As redes recorrentes citadas anteriormente geralmente são Perceptron Multicamadas (MLP), que é uma rede neural semelhante à Perceptron, mas com mais de uma camada de neurônios e alimentação direta (STUART & LING, 2004). Tal tipo de rede é composta por camadas de neurônios ligadas entre si por sinapses com pesos. O aprendizado nesse tipo de rede é geralmente feito por meio do algoritmo de retro propagação do erro (HAYKIN, 2001), mas

existem outros algoritmos para este fim, como a Rprop. O Perceptron é um algoritmo de aprendizado de classificadores binários; ele é capaz de resolver apenas problemas linearmente separáveis. Por meio do processo de treinamento, o algoritmo aprende a classificar as entradas em dois grupos diferentes exemplificado na figura 2.7 abaixo.

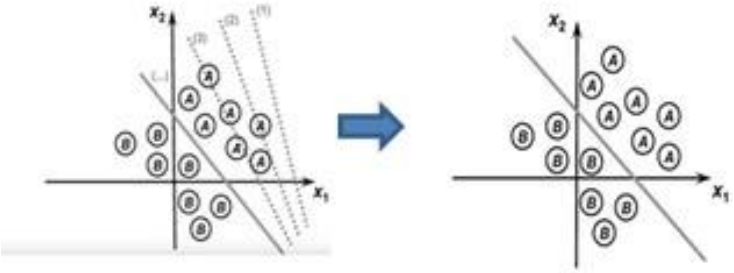

Figura 2.7: Processo de treinamnto perceptron. Fonte: Del Lama (2016).

O Perceptron Multicamadas (MLP) é uma rede neural semelhante ao Perceptron simples, porém possui mais de uma camada de neurônios. Em casos em que não há a possibilidade de uma única reta separar os elementos, há o uso da MLP que gera mais de uma reta classificadora, como exemplificado na figura 2.8. O aprendizado nesse tipo de rede é geralmente feito por meio do algoritmo de retro propagação do erro, mas existem outros algoritmos para este fim. Cada camada da rede tem uma função específica. A camada de saída recebe os estímulos da camada intermediária e constrói a resposta.

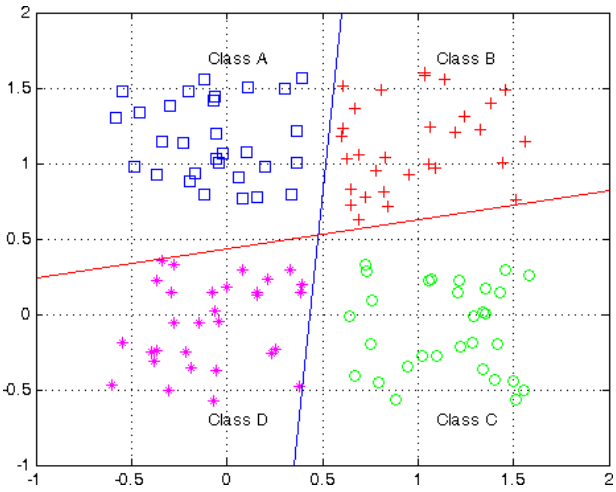

Figura 2.8: Treinamento perceptron multicamadas. Fonte: Potocnik (2012).

Uma rede neural com Função de Ativação de Base Radial (RBF) consiste em um modelo neural multicamadas, capaz de aprender padrões complexos e resolver problemas não-linearmente separáveis.

A arquitetura de uma rede RBF tem três camadas: camada de entrada, na qual os padrões são apresentados à rede: a camada intermediária (única), que aplica uma transformação não linear do espaço de entrada para o espaço escondido (alta dimensionalidade) e camada de saída, que fornece a resposta da rede ao padrão apresentado.

Cada neurônio da camada oculta calcula uma função base radial, alguns exemplos de funções radiais de base podem ser vistos na figura . As equações das camadas ocultas e de saída estão listadas abaixo:

- Camada oculta:

$$u = ||x_i - t_i|| \qquad (2.12)$$

- Camada de saída:

$$u = \sum wi\phi i||xi\ -\ ti|| \qquad (2.13)$$

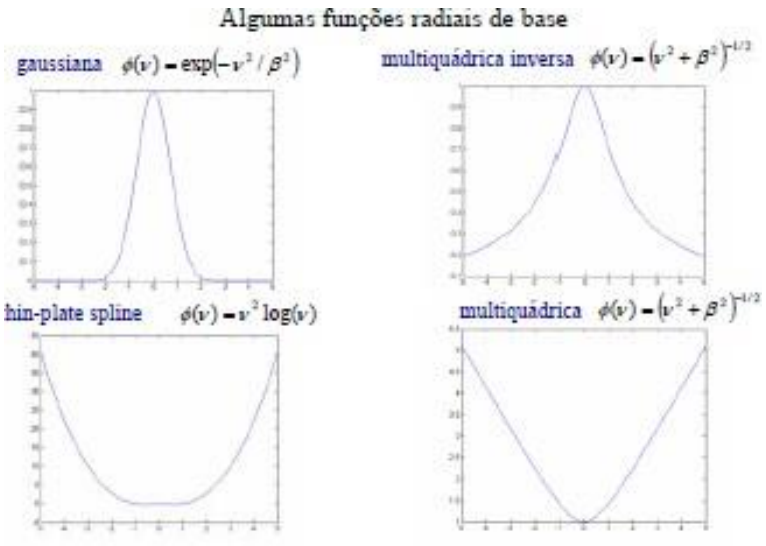

Figura 2.9: RBF-Funções de Base Radial. Fonte: Exemplo MatLab.

A diferença principal entre MLP e RBF é que a primeira utiliza hiperplanos para particionar espaço de entradas enquanto a segunda utiliza hiperelipsóides para particionar o espaço de entradas (na camada escondida).

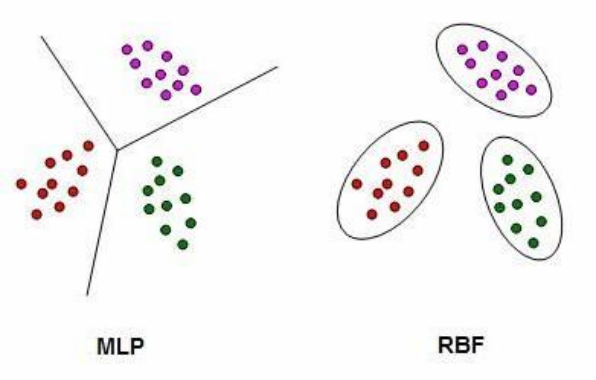

Figura 2.10: MLP X RBF. Fonte: Bishop (1995), adaptado.

Uma rede RBF (na maioria dos casos) tem uma única camada escondida. Os nós da MLP, localizados nas camadas escondidas e de saída, compartilham um modelo neuronal comum. Já na rede RBF, os nós da camada escondida são calculados diferentemente e têm um propósito diferente dos de saída. A camada escondida de uma rede RBF é não linear e a saída é linear.

O argumento da função de ativação de cada unidade escondida em uma rede RBF calcula a distância (euclidiana ou não) entre o vetor de entrada e o centro da unidade. Na MLP é calculado produto interno do vetor de entrada e do vetor de pesos sinápticos da unidade. Redes RBF normalmente usam não-linearidades localizadas exponencialmente decrescentes (ex.: funções gaussianas) gerando aproximações locais.

Neste trabalho, usa-se MLP, as camadas ocultas e de saída fazem uso da logística função sigmoide. Considerando:

$$u_i(t) = W_i^T(t)x(t) - \theta_i(t) \tag{2.14}$$

$$y_i(t) = \Phi\big(u_i(t)\big) = \frac{1}{1 + e^{-u_1(t)}} \tag{2.15}$$

Assim, a saída do i-ésimo neurônio $y_i(t)$, com i = 1,...,c , é dado por:

$$y_i(t) = \Phi[\sum_{k=1}^{q} m_i(k(t)\,\Phi(W_k^T(t)x(t) - \theta_k(t)) - \theta_i(t)] \qquad (2.16)$$

Em que $x(t) \in R_p$ é o vetor de entrada atual, $W_k = [W_{k1}, W_{k2} \ldots W_{kp}]^T$ é o vetor de pesos da k-ésima oculta neurônio, m_{ik} é o peso que liga o k-ésimo escondido neurônio ao i-ésimo neurônio de saída. Os parâmetros $\theta_k(t)$ e $\theta_i(t)$ representam, respectivamente, os limiares de ativação do k-ésimo neurônio oculto e o i-ésimo neurônio de saída.

Os pesos e limiares dos neurônios de saída, $m_i=[m_{i0}, m_{i1} \ldots m_{iq}]^T$, são ajustados. Os pesos e limiares da camada oculta são ajustado pelo procedimento clássico de retro propagação de gradientes de erros de saída para a camada oculta. Com base nisso, uma rede neural aprende por meio de um processo iterativo de ajuste de seus pesos sinápticos. O processo de aprendizagem segue a seguinte sequência:

1 - A rede neural é estimulada pelo ambiente de informação;

2 - A estrutura interna da rede é alterada como resultado do estimulo;

3 - Devido às alterações que ocorreram em sua estrutura interna, a rede tem modificada sua resposta aos estímulos do ambiente.

2.4 Algoritmo de Backpropagation

O algoritmo de Backpropagation procura encontrar iterativamente a mínima diferença entre as saídas desejadas e as saídas obtidas pela rede neural com o mínimo de erro. Dessa forma, ajustando os pesos entre as camadas por meio da retropropagação do erro encontrado em cada inteiração, obtem-se as saídas desejadas.

Esse algoritmo é um dos tipos de treinamento supervisionado, em que a rede é analisada em dois casos: na sua propagação (camada por camada) e na sua retropropagação (análise contrária à propagação), Backpropagation. No primeiro, os pesos sinápticos w_{ij} (peso sináptico conectando a saída do neurônio *i* à entrada do neurônio *j*) da rede são todos fixos. No segundo os pesos são todos ajustados.

Basicamente, um padrão de entrada é aplicado como um estímulo aos elementos da primeira camada da rede, que é propagado por cada uma das outras camadas até que a saída *a* seja

gerada. Ela é então comparada com a saída desejada a_d, gerando um sinal de erro para cada elemento de saída. O sinal de erro é então retropropagado da camada de saída para cada elemento da camada intermediária anterior que contribui diretamente para a formação da saída.

Entretanto, cada elemento da camada intermediária recebe apenas uma porção do sinal de erro total, proporcional apenas à contribuição relativa de cada elemento na formação da saída original. Este processo se repete, camada por camada, até que cada elemento da rede receba um sinal de erro que descreva sua contribuição relativa para o erro total.

Com base no sinal de erro recebido, os pesos sinápticos são então atualizados (de acordo com uma regra de correção de erro) para cada elemento de modo a fazer a rede convergir para o valor de saída desejada a_d. A ilustração do algoritmo Backpropagation pode ser verificada na Figura 2.11.

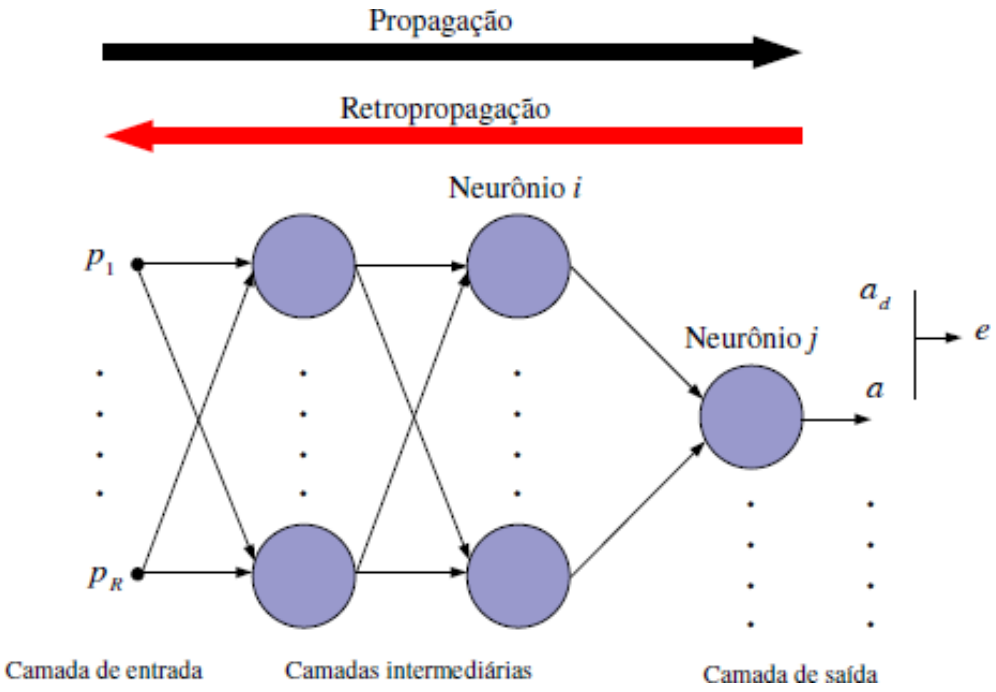

Figura 2.11: Ilustração do algoritmo Backpropagation. Fonte:Haykin (2001), adaptado.

O erro total apresentado pela rede é definido como a média da soma dos erros quadráticos:

$$E = \frac{1}{2N}\sum_{i=1}^{N}(d_i - y_i)^2 \qquad (2.17)$$

E é o erro médio total cometido pela rede, *N* é a quantidade de ádrôes que forma o conjunto de treinamento, d_i é a saída desejada para o i-ésima entrada e y_i é o valor produzido pela rede para a i-ésima saída.

O algoritmo de *backpropagation* é um algoritmo supervisionado que trabalha com pares de valores, entrada e saída desejadas. Formado por duas etapas bem definidas: *foward*, que calcula as saídas de cada nodo para cada entrada e *backward*, que atualiza os pesos da rede por meio da retropropagação do erro.

Algoritmo 1 – Pseudo-código Backpropagation

1. Inicializar pesos e bias;

2. Apresentar o padrão de entrada juntamente com sua respectiva saída desejada;

3. Propagar esse padrão de camada em camada de forma que seja calculada a saída para cada nodo da rede;

4. Comparar a saída gerada pela rede com a saída desejada e calcular o erro cometido pela rede para os nodos da camada de saída;

5. Atualizar os pesos dos nodos da camada de saída com base no erro competido por tais nodos 6. Até chegar à camada de entrada: a. Calcular o erro dos nodos da camada intermediária baseado no erro cometido pelos nodos imediatamente seguintes ponderado pelos pesos entre os nodos da camada atual e os nodos imediatamente seguintes;

7. Repetir os passos 2, 3, 4, 5 e 6 até obter um erro mínimo ou até atingir um dado número de iterações.

O erro de um nodo *j* da camada de saída é:

$$\delta = (d_j - y_j) \times f'(net_j) \qquad (2.18)$$

Onde d_j é a saída desejada para o nodo, y_j a saída que foi produzida pela rede, *f'* é a derivada da função de ativação para o valor de net_j, que é a ativação do nodo *j* dada pela equação 2.19.

$$net_j = \sum_{i=1}^{n} x_i w_{ji} \tag{2.19}$$

Na equação 2.19, *n* é o número de nodos da camada imediatamente anterior conectados ao nodo *j*; x_i é a saída de cada nodo da camda anterior e que serve como entrada j e *Wji* é o peso associado à conexão entre cada noda da camada imediatamente anterior e o nodo j. Para um nodo j da camada intermediária seu erro é dado por:

$$\delta = f'(net_j) \sum_{n} \delta_n W_{nj} \tag{2.20}$$

O valor δ_n de epresenta os erros cometidos pelos nodos imediatamente à direita de *j* e W_{nj} representa os pesos entre *j* e os nodos imediatamente a sua direita.

$$W_{ji}(k+1) = W_{ji}(k) + \Delta W_{ji} \tag{2.21}$$

$$\Delta W_{ji} = \eta \delta_j x_i \tag{2.22}$$

Na equação 2,22, η representa a taxa de aprendizagem, em um valor no intervalo (0,1), δ_j é o erro produzido por um nodo da camda de saída e x_i é o valor de entrada do nodo *j*.

2.5 Controle de Movimento em Robótica Móvel

Para usar um método de controle por RNAs é necessário que o modelo de controle seja tão próximo quanto possível do real, de modo que os resultados previstos expressem resultados melhores. O método utilizado neste trabalho foi o direto, mostrado a seguir:

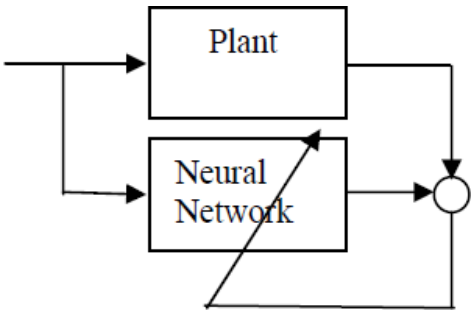

Figura 2.12: Modelo do controlador direto de trajetória por RNA. Fonte: Haykin (2001).

Na figura acima é possível ver que a mesma entrada é enviada para a planta e para a rede. A rede neural usa um modelo de aprendizagem supervisionada, em que a saída da planta atua como um “professor” fornecendo os valores desejados.

A rede compara a saída da planta com seus resultados e corrige os parâmetros dentro de uma época. Após a correção, os novos parâmetros são informados ao controlador preditivo, de forma que novas saídas de controle são previstas até que a trajetória seja finalizada. Como a rede neural tem uma estrutura estática, foi usada uma rede recorrente a fim de captuar a dinâmica do processo.

Além disso, existem métodos de controle indireto, onde não é necessário fazer a identificação de parâmetros de sistema, deixando para a rede neural apenas a função de controlador; neste caso, a rede compara os valores de referência com os obtidos pela saída do processo e gera os sinais de controle apropriados.

No caso desse projeto, o processo de controle de trajetória acontece pela RNA, que simplifica o processo de contole diretamente. Os dados dos sensores ultrassônicos fornecem a posição aproximada ao robô no arduino pelo processo de odometria; os valores dos pesos sinápticos da RNA são reultados do treinamento utilizando os dados de velocidades captados pelos encoders e de posição são captados pelos sensores de dostância. No processo de imitação são comparados as entradas e saída na RNA que move os atuadores (motores) aproximando a posição obtida pelo algoritmo de backpropagation. Um esquemático simplificado pode ser visto na imagem a seguir:

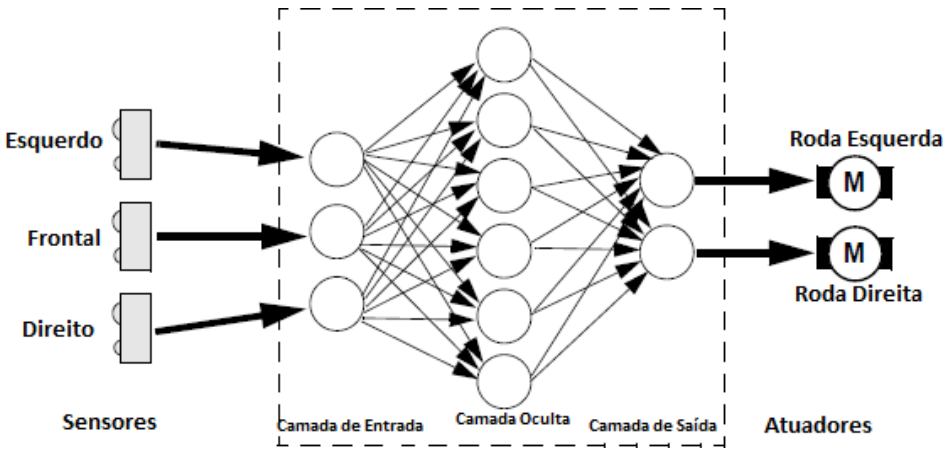

Figura 2.13: Modelo do controle por RNA do robô. Fonte: T. Braünl (2003), adaptado.

2.6 Trabalhos Correlatos

Durante a escrita desta dissertação foram pesquisados artigos importantes citados atualmente para Learning from Demonstration; analisando suas técnicas, a plataforma de desenvolvimento, os resultados obtidos e processamento de treinamento em geral off line, alguns artigos estão na tabela 2.1 abaixo. Dentre os artigos se destacam o "A Robot that Walks; Emergent Behaviors from a Carefully Evolved Network", escrito no MIT em 1989, foi a primeira tentativa de implementar um robô móvel que evoluía aprendendo com uma rede neural rudimentar em máquina de estado, ainda estava distante do LfD conhecido de hoje.

Em 2002, na IEEE International Conference on Robotics and Automation em Washington, o PhD William D. Smart e a professora do MIT Leslie Pack Kaelbling apresentaram o artigo "Effective Reinforcement Learning for Mobile Robots"; a principal ideia do artigo era demonstrar que é mais fácil mostrar ao robô o que você quer e deixar ele aprender por si, do que a programação tradicional, que leva em conta inúmeros cenários. A plataforma usada foi um robô móvel de tração síncrona B21r, as técnicas para implementação da LfD foram o Q – Learning, que usa diversas redes neurais, mais especificamente uma rede neural convolucional (CNN do inglês Convolutional Neural network ou ConvNet) é uma classe de rede neural artificial do tipo feed-forward.

Outro trabalho é o do Dr. Hans Jörg Andreas Schneebeli professor da Universidade Federal do Espírito Santo e do PhD José Alberto Santos Victory do Instituto Técnico Superior de Lisboa, juntos em 2006, escreveram o artigo "Aprendizagem por Imitação através de mapeamento visuomotor baseado em imagens omnidirecionais" em que propõe uma metodologia robô aprender uma tarefa a adaptando e a representando de acordo com a sua capacidade motora e sensorial. Por fim o último artigo da tabela foi escrito numa parceria entre Universidade de Atlanta e a Universidade de Kyoto em uma pesquisa sobre braços robóticos, pelos PhDs: Stefan Schaal e Hikaridai Seikocho; o artigo trata de comparar modelos matemáticos para otimizar o aprendizado de trajetórias utilizando LfD, usando técnicas de RNAs e Q-learning, aplicado a um braço robô utilizado em processos industriais.

Tabela 2.1: Artigos analisados que se destacaram de Learning from Demonstration.

Autor	**Ano**	**Técnica**	**Plataforma**	**Aplicação**	**Resultados**
Brooks, A Rodney	1989	FSM - Máquina de estados	PC- Robô Móvel	Robô móvel usando FSM para aprender por demonstração, a primeira tentativas de implementação de LfD em FSM no MIT com rede neural simples.	Demonstrou vários resultados: capacidade de aprendizado da rede neural, sistema de controle mais simplificado e a não necessidade de fusão de sensores em nesse caso específico
Smart,William D. ; Kaelbling,Leslie Pack	2002	Q-Learning	B21r mobile robot	Robótica Móvel, novos trajetos e desvios de obstáculos sem especificar como deixando o robô aprender sozinho.	Treinado a 1m distância da parede teve 46 % acertos, 2m de distância da parede 25% e a 3m 18.7% no caso mais complexo
Abbeel, Pieter ; Ng Y., Andrew	2004	Markov	PC	Utilizar cadeias de Markov para otimizar LfD, especificamente a função de recompensa.	Obteve melhora de Desempenho em alguns casos específicos como aprendizado por reforço inverso
Schneebeli, Hans; Vassalo, Raquel; Victor, Jose	2006	Processamento de Imagens	PC- Robô Móvel	Um robô mapear e aprender com base no proc. de imagens a realizar tarefas, mapear navegação em LfD	O robô conseguiu mapear e aprender em hierarquia etapas de auto exploração e aprendizagem por imitação

Brenna D. Argall; Sonia, Chernovab; Manuela, Veloso; Brett, Browninga	2009	LfD algoritmos	PC - Robô Móvel	Robô móvel autônomo com capacidade de sobrevivência em ambientes hostis com capacidade de aprendizagem LfD	O artigo conseguiu demonstrar a eficácia do uso de LfD no controle de robô em ambientes hostis mas demonstra necessidade de melhoras em sempre mapear, modelar o sistema e planejar as tarefas.
Perico, Danilo; Bianchi , Reinaldo	2011	Heurísticas para aceleração de aprendizado em LfD	PC - Robô Móvel	Sair de um labirinto, sendo que alguns caminhos são demonstrados a ele. Usando heurísticas para acelerar o Aprendizado por Reforço.	A criação da nova heurística permitiu uma melhora no tempo de aprendizagem quando comparado com o AR sem aceleração
Brys, Tim; Harutyunyan, Anna ; Taylor, Matthew; Suay, Halit; Chernova, Sonia; Nowé, Ann	2015	RL, RLfD (shaping), RLfD (HAT)	PC	Comparação das principais técnicas de LfD em situações comuns da indústria e a comparação dos seus resultados tentativa de criar um algoritmo que otimiza o aprendizado	O algoritmo RLfD de Shapping foi um pouco melhor que os outros na maioria dos casos superando o HAT que era o objetivo

Schaal, Stefan	2018	Q-Learning	SARCOS anthropomorphic robot arm	Aprimorar o aprendizado de trajetórias para processos industriais, comparando os algoritmos de Q - Learning mais conhecidos usando aprendizado por reforço	Conseguiu aprimorar o algoritmo de Q-learning e teve um problema de pior desempenho com o LQR em aprendizagem por reforço
Silvera, Rodrigo;	2018	CNAPap	Máquina com processo de movimentação de peças eletropneumáticas , composta por atuadores lineares, ventosa e sensores analógicos de posicionamento linear	Alternativa para novas implementações de	Verificou-se que o Sistema Paraconsistente de Aprendizagem
Filho, João;		(Célula Neural Artificial Paraconsistente de aprendizagem)		controles em processos industriais com Aprendizagem Por Demonstração. Com o objetivo de proporcionar maior facilidade na atribuição de movimentos em máquinas industriais, foi construído uma Célula Neural Artificial Paraconsistente de aprendizagem CNAPap	Por Demonstração - SPAPD-LPA2v na forma do bloco funcional FB_CNAP pode realmente ser implementado em aplicações industriais que envolvam as características da LPA2v e Inteligência Artificial.
Garcia Dorotea					
XIE ZongWu, ZHANG Qi, JIANG ZaiNan & LIU Hong	2020	Revisão de todas as principais técnias para	PC- Simulação	Conhecer as principais metodologias e técnicas e suas	Conseguiu demonstrar as principais técnicas de LfD e comprovar

		planejamento de LfD		vantagens para cada tipo de aplicação.	vantagens e desvantagens de várias metodologias.
XIE ZongWu, ZHANG Qi, JIANG ZaiNan & LIU Hong	2020	Revisão de todas as principais técnias para planejamento de LfD	PC- Simulação	Conhecer as principais metodologias e técnicas e suas vantagens para cada tipo de aplicação.	Conseguiu demonstrar as principais técnicas de LfD e comprovar vantagens e desvantagens de várias metodologias.
Aran Sena and Matthew Howard	2019	LfD	Robô manipulador	Robô manipulador para colheita em agricultura	O robô conseguiu aprender a detectar quais plantas deviam ou não ser colhidas, porém apresentou oportunidades de melhorias na parte de detecção.
Harish Ravichandar, Athanasios S. Polydoros,Soni a Chernova, e Aude Billard	2019	LfD	PC Simulação e Robô	Comparativo de técnicas de LfD para teleoperação e movimentação cinética	Comparação entre duas formas de ensinar via teleoperações e por demonstração cinética em situações específicas cada uma das técnicas apresentou vantagens e desvantagens.
Joseph DelPreto , Jeffrey I. Lipton, Lindsay Sanneman , Aidan J. Fay,Christopher Fourie , Changhyun Choi, and Daniela Rus.	2020	Revisão de todas as principais técnias para planejamento de LfD	PC- Simulação	Conhecer as principais metodologias e técnicas e suas vantagens para cada tipo de aplicação.	Conseguiu demonstrar as principais técnicas de LfD e comprovar vantagens e desvantagens de várias metodologias.

Capítulo 3

METODOLOGIA

Neste trabalho foram realizadas implementações e treinamentos de RNAs para aprendizado por demonstração aplicado em aprendizagem e controle de trajetórias em robótica móvel utilizando dados de simulações no *EyeSim*. Além de construir um protótipo robô físico para coleta de dados e para aprender uma trajetória, assim como para realizar testes mais realistas validando o uso das RNA's na solução do problema de LfD.

A metodologia e ferramentas adotadas para a solução do problema proposto está abaixo e foi resumida na figura 3.1:

- Estudo teórico e revisão bibliográfica: Na primeira parte foi realizado um estudo teórico sobre robótica móvel, inteligência artificial e aprendizagem de máquina (especificamente aprendizagem por demonstração utilizando redes neurais artificias); conforme apresentado no capítulo 2;
- Simulação: Foi utilizado o software de projeto de robôs, *EyeSim*, em combinação com o *Unity 3D* no Windows para simular o robô em diversas trajetórias em cenários distintos, enquanto se coletava os dados de distância e velocidade;
- Treinamento das redes neurais: Usando linguagem R combinada com o software de inteligência artificial Weka, além da ferramenta do Matlab nntools, foram treinadas as redes neurais e os resultados foram conferidos por Cross-validation e salvos em txt;
- Imitação: Os pesos sinápticos das RNAs gerados pelo Weka foram salvos em txt depois codificadas em C++ e Arduino, de forma a imitar o comportamento aprendido no *EyeSim* e no robô físico. Por último, foi usado o método estatístico de comparação (coeficiente de correlação de postos *Spearman*), que indica que as saídas esperadas estão correlacionadas às saídas obtidas pela rede.

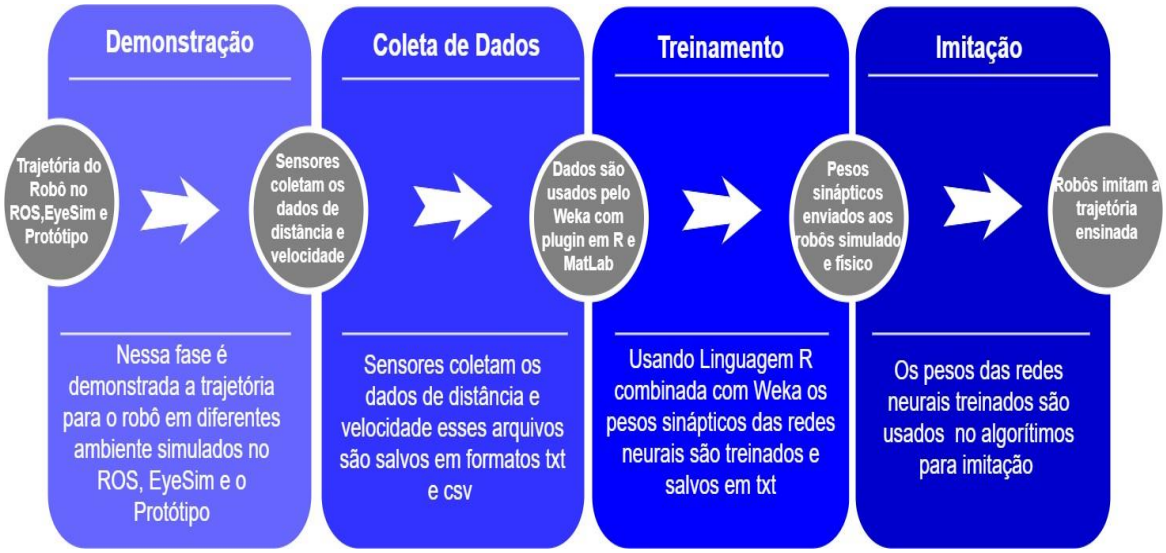

Figura 3.1: Fluxograma da metodologia desenvolvida. Fonte: Autor.

Os algoritmos desenvolvidos em C, os pesos sinápticos das redes neurais, o modelo da topologia da RNA utilizado no Weka, as regressões lineares dos resultados no Matlab e os vídeos dos experimentos realizados podem ser encontrados no repositório do laboratório LEIA.

3.1 Ambiente de Simulação *EyeSim*

O *EyeSim* é um simulador que possibilita uma visualização 3D de robôs móveis; ele foi feito usando como base a engine de jogos *Unity 3D*. O software ainda permite utilizar e criar diferentes cenários (worlds), topologias de robôs, sensores de posição, câmera e comunicação a rádio. Foi utilizado um robô, o *Eve.graco*, com 7 sensores (frontal, esquerda, direita e 4 diagonais); sendo, o robô, de tração diferencial.

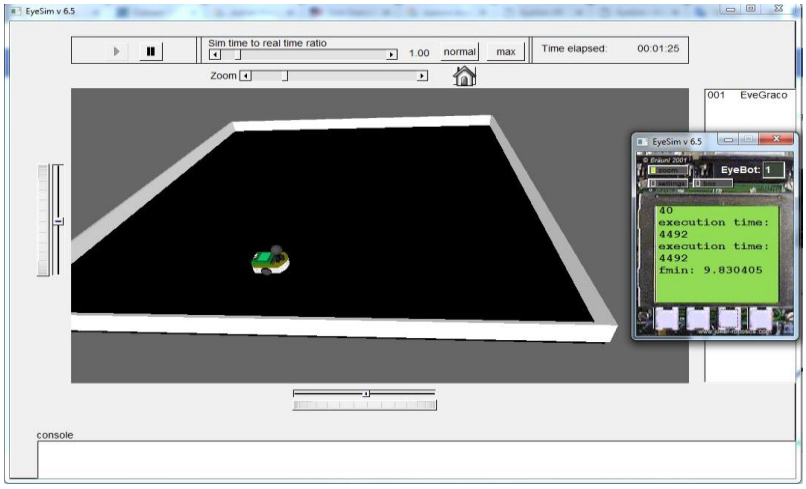

Figura 3.2: Ambiente de Coleta de dados EyeSim. Fonte: Próprio autor.

Na primeira etapa de simulação, os dados dos sensores de distância foram coletados, em milímetros, junto com a velocidade linear e angular estimada do robô e foram armazenados compondo, respectivamente, os dados de entrada e saída desejada da rede.

Na segunda etapa, teve-se a fase de aprendizagem, que basicamente é o processo de adaptação dos pesos sinápticos da rede neural depois do treinamento usando os dados coletados. Cada trajetória gerou diferentes pesos sinápticos. Após o processo de geração desses pesos, aplicou- se de volta os pesos sinápticos no algoritmo de demonstração, de forma que o robô "imitasse" a trajetória ensinada com certa precisão. Espera-se que quanto maior e mais sólida for a coleta de dados na fase de demostração, maior será a capacidade e precisão para o movimento do robô.

3.2 Fase de Demonstração e Coleta de Dados

Nessa fase foram demonstradas trajetórias com diversos comportamentos para o robô em diferentes ambientes, três cenários simulados e três cenários físicos.

Para os três cenários simulados, o simulador *EyeSim* foi usado para a coleta de dados e demonstração de trajetória ao robô, usando os valores dos sensores de distância e velocidades das rodas do robô. Após realizar diferentes trajetórias e percursos com o robô, foi iniciado o treinamento da rede neural artificial perceptron multicamadas, usando o software Weka, em paralelo com o plugin da linguagem R, para testar os resultados a rede.

Dessa forma, foram realizados seis experimentos, sendo eles:

- Experimento 1:

Neste cenário o objetivo era ensinar várias curvas e retas sucessivas, andando próximo dos objetos e não encostando nas paredes, para isso foi feito um cenário em formato de cruz a fim de treinar esta habilidade.

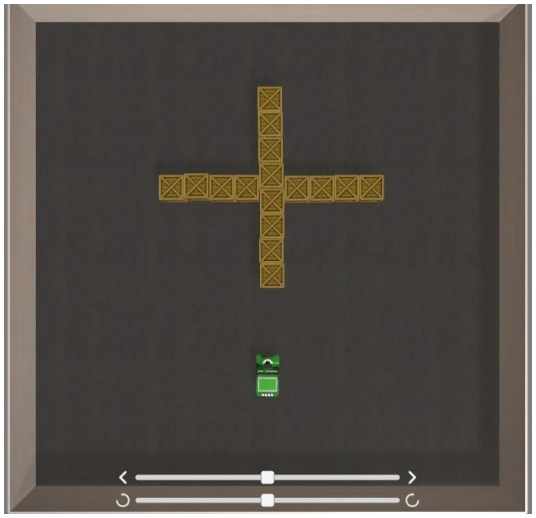

Figura 3.3: Cenário do Experimento 1. Fonte: Próprio autor.

- Experimento 2:

Os cenários cruz e um outro cenário não estruturado foram modificados para semi-estruturados, a fim de testar a capacidade de imitação em um cenário com objetivos e curvas que eles desconheciam utilizando apenas a experiência prévia aprendida nos experimentos anteriores.

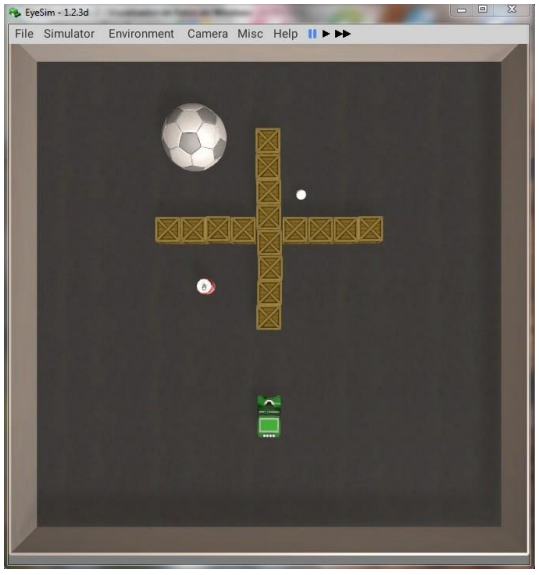

Figura 3.4: Cenário do Experimento 2. Fonte: Próprio autor.

- Experimento com robô físico 1 – Contornar a caixa:

Nestes experimentos foi utilizado o protótipo do robô móvel construído e feito um cenário utilizando cartolinas em que um lado ficava a trajetória ensinada e no outro a aprendida, como pode ser visto na figura 3.5. Colocando uma caneta de quadro branco na estrutura do robô para assim avaliar a trajetória realizada. O objetivo desse primeiro experimento é ensinar ao robô a capacidade de contornar um objeto simples.

Figura 3.5: Cenário do Experimento 1 com robô físico. Fonte: Próprio autor.

- Experimento com robô físico 2 – Cruz:

O objetivo neste experimento foi comparar a capacidade de imitação do robô com um cenário parecido ao simulado, no caso, o cenário em cruz. O cenário fechado possui uma dimensão aproximada de 3m x 1,5m; o robô é controlado a fim de realizar o trajeto semelhante ao ensinado na simulação enquanto coleta os dados de velocidade e distância.

Figura 3.6: Ensino de trajetória em cruz com robô físico. Fonte: Próprio autor.

- Experimento com robô físico 3 – Contornar parede:

Neste experimento o objetivo foi ensinar ao robô como desviar da parede em um cenário vazio, andando em linha reta o mais próximo das quinas possível, sem colidir. O cenário fechado possuia medidas de aproximadamente 3m x 1,5m com paredes feitas por isopor.

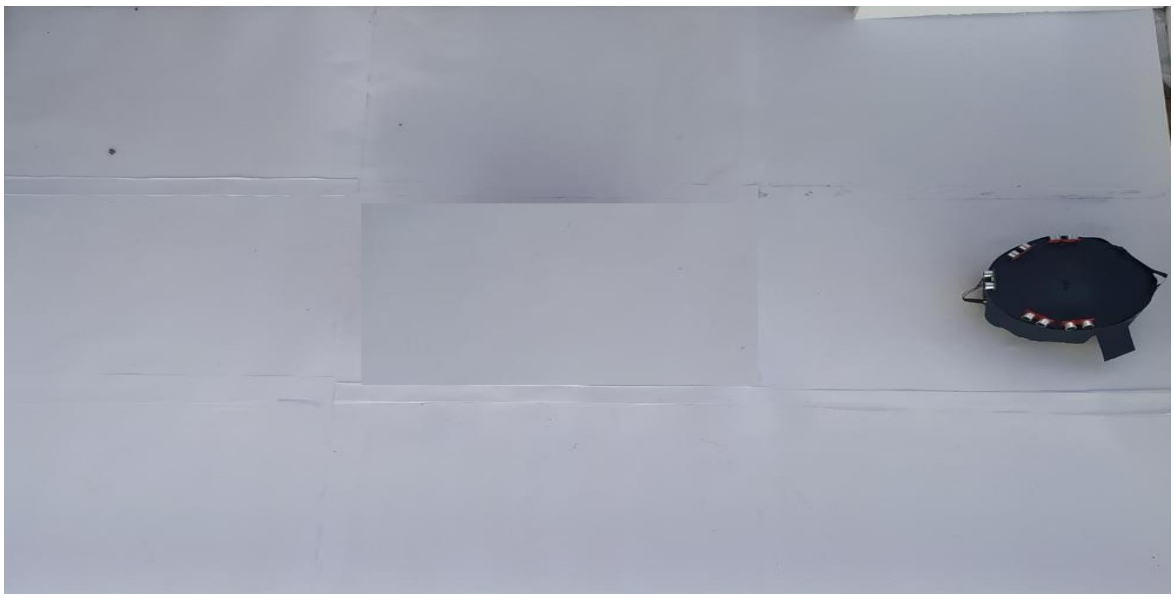

Figura 3.7: Ensino de desvio de obstáculos estáticos e cinéticos. Fonte: Próprio autor.

3.3 Fase de Aprendizagem

A fase de aprendizagem é o resultado da adaptação dos pesos sinápticos de uma RNA, utilizando os dados coletados anteriormente pelos robôs simulados e físico. A precisão dos pesos sinápticos gerados determina a eficiência do aprendizado e da execução das tarefas pelo robô. É importante esclarecer que, para um mesmo cenário, diferentes pesos sinápticos são obtidos para diferentes trajetórias ou comportamentos ensinados. Dessa maneira, após a realização do treinamento de determinada tarefa e a geração de seus pesos sinápticos, os novos pesos podem ser aplicados ao robô no intuito de imitar a trajetória ou comportamento ensinado.

Um dos fatores mais importantes na influência desse aprendizado é a quantidade de dados coletados na fase de demonstração; quanto maior o número de dados em determinada tarefa ou situação, mais experiência e informações o robô terá sobre que decisão tomar nessa ocasião. Para treinar os pesos sinápticos e validar as diferents topologias de redes foram utilizados o software de aprendizado de máquina Weka e a ferramenta do Matlab “nntools”.

3.3.1 Weka

O software Weka (*Waikato Environment for Knowledge Analysis*) começou a ser desenvolvido em 1993, na Universidade de Wikato, Nova Zeilândia, sendo adquirido posteriormente por uma empresa no final do ano de 2006. O Weka encontra-se licenciado pela *General Public License* sendo portanto possível estudar e alterar o respectivo código fonte.

O Weka tem como objetivo agregar algoritmos provenientes de diferentes abordagens/paradigmas na sub-área da inteligência artificial dedicada ao estudo de aprendizagem de máquina. Esta sub-área pretende desenvolver algoritmos e técnicas que permitam a um sistema “aprender”, no sentido de obter novo conhecimento, quer de maneira indutiva quer dedutiva.

O Weka aplica uma análise computacional e estatística dos dados fornecidos recorrendo a técnicas de mineração de dados tentando, indutivamente, a partir dos padrões encontrados, gerar hipóteses para soluções e teorias sobre os dados em questão. Os modelos Weka podem ser usados, contruídos e avaliados em liguagem R usando o pacote RWeka. Os algoritmos

das RNAs em R e ferramentas de visualização do Weka foram feitas usando o pacote RPlugin para Weka.

Pela versatilidade e poder de análise foi escolhido o Weka, em conjunto com o Plugin de linguagem R, a fim de treinar as redes neurais; este processo foi realizado em três etapas:

- Primeira: escolha dos dados do experimento feito no robô e salvo no Excel em formato .csv;
- Segunda: escolha do classificador entre os modelos disponíveis pelos Plugins instalados e uso do *Mult Layer Perceptron* em liguagem R para treinar as RNAs.
- Terceira: Resultados dos pesos sinápticos das redes neurais salvos em .txt e rodados nos algoritmos dos robôs simulados e físicos; além da avaliação de validação cruzada, feita 10 vezes em partes distintas dos dados, obtendo-se uma estimativa do desempenho preditivo.

A imagem abaixo mostra um dos treinamentos do robô e a tela do Weka:

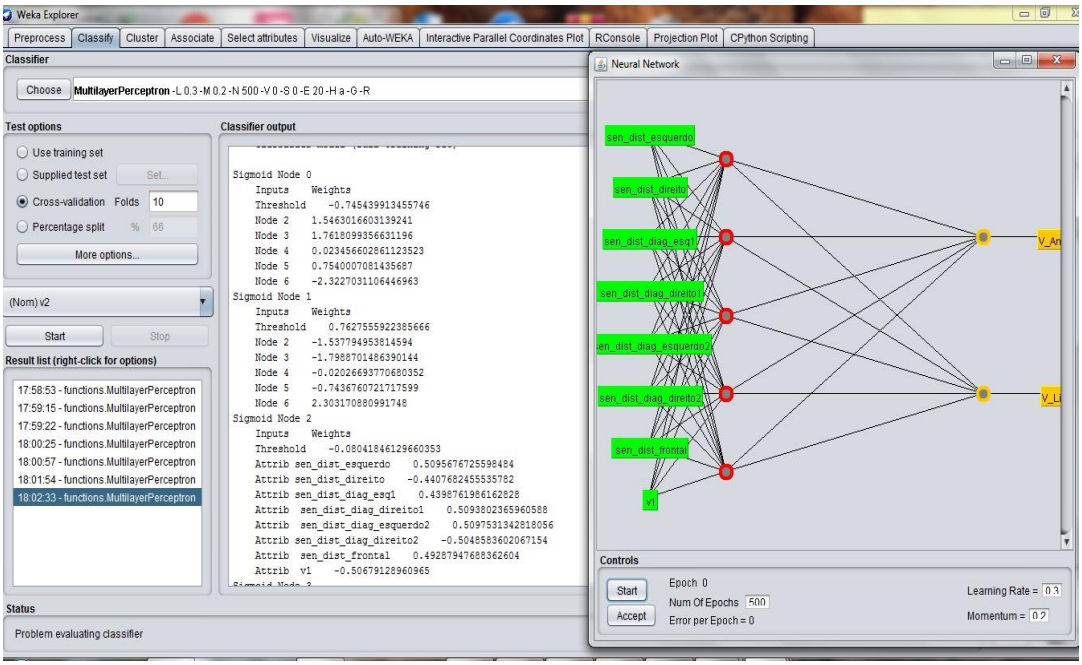

Figura 3.8: Tela de resultados de treinamento do robô no Weka. Fonte: Próprio autor.

A validação cruzada é uma técnica de avaliação padrão e uma maneira sistemática de executar uma segmentação percentual repitida dos dados de treinamento. O conjunto de dados carregados é dividido em 10 pedaços (‘dobras’), sendo nove separados para treinos e um para teste, repetindo-se para cada um dos 10 pedaços, que fornece des resultados de avaliação, que são calculados em média. Na validação cruzada, “estratificada”, ao fazer a segmentação inicial, garante-se que cada dobra contenha aproximadamente a proporção correta dos valores da classe. Após a validação cruzada ser executada 10 vezes, o Weka invoca o algoritmo de aprendizado uma última (11ª vez) em todo o conjunto de dados para obter o modelo que se imprime na tela de resultado, vide figura 3.9.

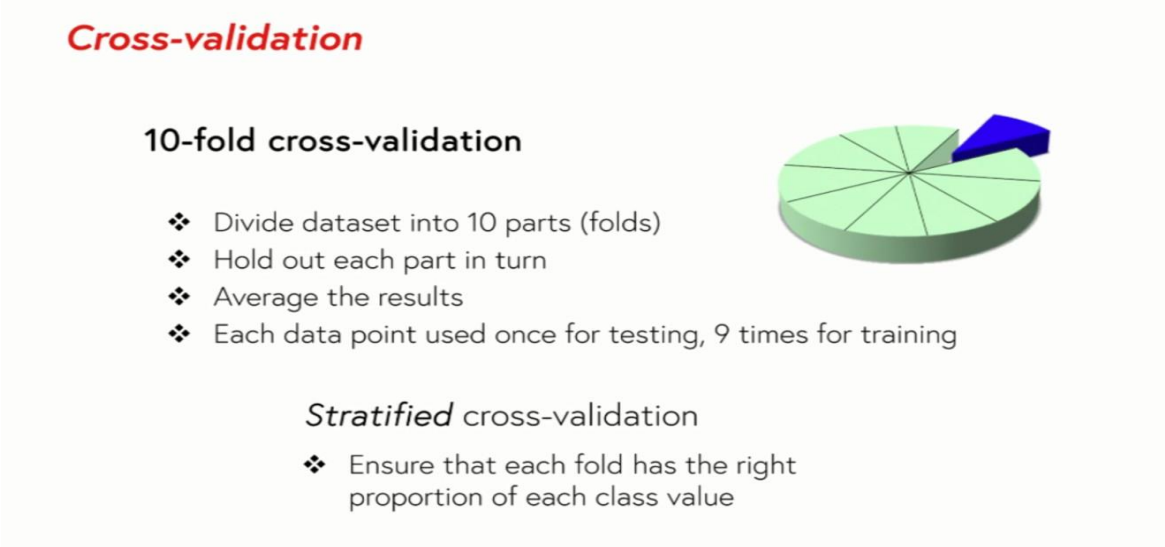

Figura 3.9: Processo de Cross Validation. Fonte: WITTEN (2016).

O Weka foi utilizado para calcular o rô de Spearman. Em estatística, o coeficiente de correlação de postos de Spearman que recebe este nome em homenagem ao psicólogo e estatistico Charles Spearman frequentemente denotado pela letra grega (ρ) ou r_s, é uma medida não paramétrica de correlação de postos (depedência estatística entre a classificação de duas variáveis). O coeficiente avalia com que intensidade a relação entre duas variáveis pode ser descrita pelo uso de ma função monótona.

A correlação de Spearman entre duas variáveis é igual à correlação de Pearson entre os valores de postos das mesmas variáveis. Enquanto a corrrelação de Pearson avalia relações lineares, a correlação de Spearman avalia relações monótonas, seja elas lineares ou não. Na estatística, o coeficiente de correlação de postos de Kendall, comumente chamado de correlação

tau de Kendall (devido a letra grega τ), é uma estatística usada para medir a correlação de postos entre duas quantidades medidas. Um teste tau é um teste de hipóteses não paramétrico referente à dependência estatística baseada no coeficiente tau. Intuitivamente, a correlação de Kendall entre duas variáveis será elevada se as observações tiverem uma classificação semelhante (ou idêntica, no caso de correlação igual 1) comparadas as duas variáveis.

3.3.2 Matlab nntool

Uma das principais vantagens do software Matlab é a facilidade de escrever e depurar um programa, se comparado a outras linguagens de programação (tais como C, Basic, Pascal ou Fortran). Além disso, possui diversas funções matemáticas, matriciais e gráficas que simplificam e minimizam a estrutura do programa.

O Matlab também dispõe de diversas bibliotecas ou ferramentas (toolboxes) para aplicações específicas, como por exemplo: redes neurais, lógica fuzzy, otimização de sistemas, wavelets, cálculo simbólico, processamento de sinais e outros.

O toolbox abordado neste trabalho foi o de redes neurais, que pode ser executado por meio de uma interface gráfica (nntool), linhas de comando ou script em formato ".m". O ambiente de trabalho do nntool pode ser visto abaixo:

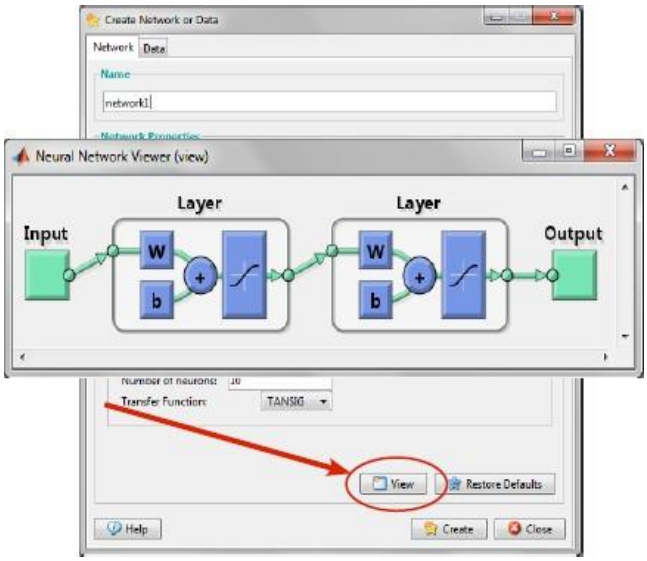

Figura 3.10: Ambiente de trabalho do nntool. Fonte: Demuth & Beale (2016).

O Matlab foi utilizado para verificar a melhor validação de performance, tomando como base o erro quadrático médio. A mesma topologia treinada no Weka era usada no MatLab; os dados foram carregados e, por regressão linear, foi analisado o desempenho dos pesos sinápticos, treino, validação e teste.

Em estatística, regressão linear é uma equação para se estimar a condicional (valor esperado) de uma variável y, dados os valores de algumas outras variáveis x. A regressão, em geral, tem como objetivo tratar de um valor que não se consegue estimar inicialmente.

A regressão é chamada "linear" porque se considera que a relação da resposta às variáveis é uma função linear de alguns parâmetros. Os modelos de regressão que não são uma função linear dos parâmetros se chamam modelos de regressão não-linear.

Modelos de regressão linear são frequentemente ajustados usando a abordagem dos mínimos quadrados. Mas também podem ser montados de outras maneiras, tal como minimizando a "falta de ajuste" em alguma outra norma (com menos desvios absolutos de regressão), ou através da minimização de uma penalização da versão dos mínimos quadrados. Por outro lado, a abordagem de mínimos quadrados pode ser utilizado para ajustar modelos que não são modelos lineares. Assim, embora os termos "mínimos quadrados" e "modelo linear" estejam intimamente ligados, eles não são sinônimos.

3.4 Protocolo Experimental

Para os resultados de simulação foi realizado o seguinte procedimento:

1) *EyeSim* - simulação dos robôs com sensores de velocidade e de distância e com cenários estruturados e não estruturados para coleta de dados, salvos em txt e convertidos em csv na fase de demonstração.

2) Weka - Treino da rede neural artificial com topologia 7-4-2 com 7 entradas dos sensores, 4 neurônios na camada oculta e 2 neurônios de saídas correspondentes as velocidades; os pesos sinápticos resultantes do treinamento foram salvos em txt nomeados "td data".

3) Imitação no *EyeSim* - Esta etapa foi realizada utilizando o algoritmo de imitação em linguagem C, tomando como base os valores dos pesos sinápticos carregados no arquivo em txt.

4) Matlab - Os dados do experimento são carregados com a mesma topologia de RNA usada no Weka por meio da ferramenta nntool para achar a melhor validação de

performance, tomando como base o erro quadrático médio e a época (ou momento) que se consegue o menor erro.

5) Regressão Linear no Matlab - Os dados de treinamento, de validação (conforme o método explicado no 4º parágrafo da seção 3.3.1) e de teste da rede neural sofreram regressão linear.

Já para o robô físico construído, o procedimento adotado foi o seguinte:

1) Ambientes experimentais estruturados e não estruturados, ambientes estáticos e construídos com caixas de papelão e dinâmicos usando uma bola em movimentação em direção ao robô.

2) Weka - Treinamento a rede neural artificial com topologia 5-3-2 com 5 entradas dos sensores, 3 neurônios na camada oculta e 2 neurônios de saídas correspondentes as velocidades. Os pesos sinápticos resultantes do treinamento são salvos em txt.

3) Imitação no ambiente físico utilizando o algoritmo de imitação adaptado para o robô arduino tomando com base os valores dos pesos sinápticos carregados no arquivo em txt.

4) Matlab - Os dados do experimento são carregados com a mesma topologia de RNA utilizada no Weka (5-3-2) por meio da ferramenta nntool, para achar a melhor validação da performance tomando como base o erro quadrático médio e a época (ou momento) que se consegue o menor erro.

5) Regressão linear o Matlab dos dados de treinamento, saída da RNA, validação conforme o método explicado no 4º parágrafo da seção 3.3.1 e teste da rede neural.

O fluxograma abaixo explica o funciomento a arquitetura geral do sistema do robô físico:

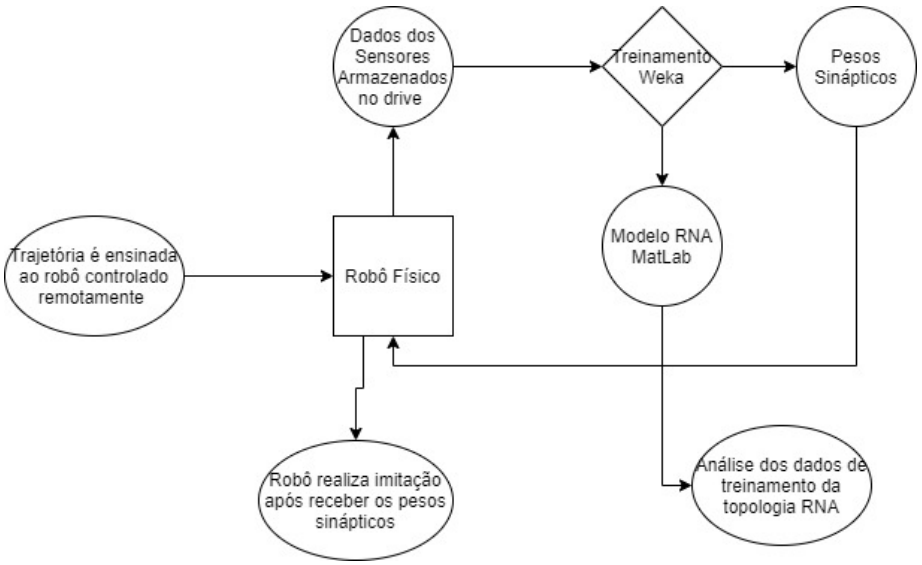

Figura 3.11: Fluxograma da arquitetura geral do sistema. Fonte: Fonte: Próprio autor.

Em resumo o protocolo experimental tanto no robô simulado quanto no físico segue as etapas de demonstração, treinamento, imitação e análise.

- **1) Demonstração**: ambientes simulados EyeSim e construídos para o robô físico
- **2) Treinamento** da rede neural utilizando Weka
    - Topologia (7,4,2) para ambientes simulados
    - Topologia (5,3,2) para ambientes reais
- **3) Imitação:**
    - Pesos sinápticos carregados no código embarcado de imitação para ambientes simulados e reais
    - Avaliar o comportamento emergente com testes em ambientes desconhecidos
- **4) Análise** de dados utilizando MatLab
    - Gradiente e momento do erro
    - Erro quadrático médio
    - Regressões lineares do treinamento, teste, validação e saídas.

3.5 Construção do Protótipo Robô

Foi construído um protótipo de robô móvel utilizando arduino para coleta de dados e treinamento das RNA's. Para a construção desse robô foram utilizados os seguintes materiais:

- 1 Arduino UNO;
- 1 Arduino MEGA;
- 1 Ponte H - Com L298N;
- 1 Alimentação de 12 V com no mínimo 3A (utilizadas 8 Pilhas em série);
- 1 Bateria 9 V;

- 1 Módulo SD Card MH-SD;
- 1 Módulo BlueTooth HC-05;
- 2 Motores com caixa de redução de rodas;
- 2 Rodas furadas para encoder;
- 2 Sensores de velocidade LM393;
- 5 Sensores HC-SR04;
- 1 Roda boba que deu a altura correta para o robô, entre a base e onde a roda toca o chão, uma distância de 4,5 cm;
- 2 Parafusos de diâmetro 3mm e comprimento de 10 mm e duas porcas;
- 1 Parafuso de diâmetro de 5 mm e comprimento de 50 mm com uma ruela e uma porca;
- 1 Presilha plástica;
- 1 Prancheta para anotações em MDF ou um pedaço de MDF de 19 cm x 12 cm.

Figura 3.12: Materiais protótipo 1 - estrutura em MDF. Fonte: Próprio autor.

3.5.1 Sensores Ultrassônicos

O funcionamento do HC-SR04 (datasheet) se baseia no envio de sinais ultrassônicos pelo sensor, que aguarda o retorno (echo) do sinal, e com base no tempo entre envio e retorno, calcula a distância entre o sensor e o objeto detectado.

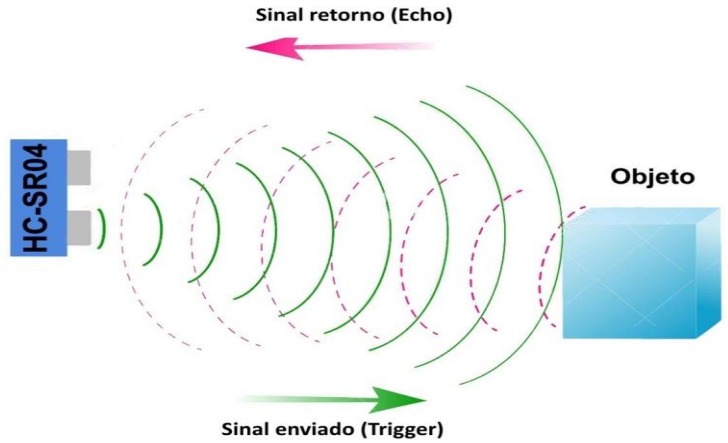

Figura 3.13: Funcionamento do HC-SR04. Fonte: Site Flipflop.

Primeiramente é enviado um pulso de 10μs, indicando o início da transmissão de dados. Depois disso são enviados 8 pulsos de 40 KHz e o sensor então aguarda o retorno (em nível alto/high) como mostrado na figura 3.14, para determinar a distância entre o sensor e o objeto, utilizando a equação 3.1 :

$$Distância = \frac{Tempo\ echo\ em\ nível\ alto\ \times Velocidade\ do\ som}{2} \qquad (3.1)$$

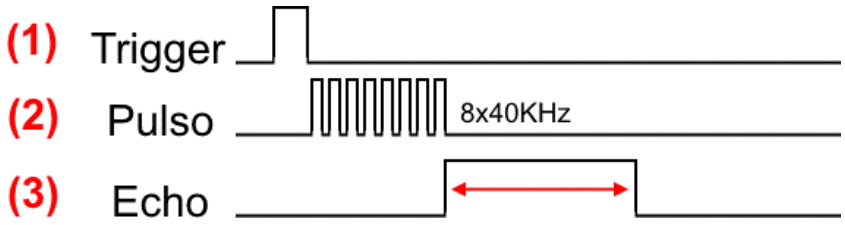

Figura 3.14: Diagrama de tempo do HC-SR04. Fonte: Datasheet do HC-SRO4.

Para calibrar o sensor ultrassônico foi utilizado um objeto como obstáculo e uma fita métrica para conferir se a medição dos sensores estavam coerentes. Os resultados foram acompanhados pelo serial monitor na IDE do arduino pela tela do notebook.

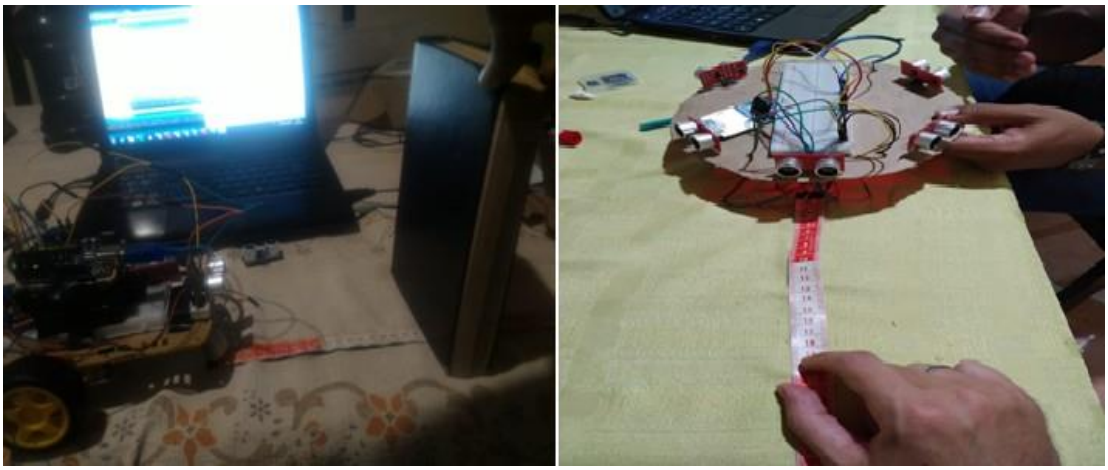

Figura 3.15: Calibração do sensor de distância. Fonte: Próprio autor.

3.5.2 Ponte H

Uma Ponte H é um circuito especial que permite realizar a inversão da direção (polaridade) de corrente que flui através de uma carga. É muito utilizada, por exemplo, para controlar a direção de rotação de um motor DC.

O circuito necessita de um caminho que carregue a corrente ao motor em uma direção e outro caminho que leve a corrente no sentido oposto. Além disso, o circuito deve ser capaz de ligar e desligar a corrente que alimenta o motor.

Uma ponte H possui quatro interruptores eletrônicos, que podem ser controlados de forma independente. A figura a seguir mostra a disposição dessas quatro chaves (switches) (S1 a S4) em relação a um motor DC controlado.

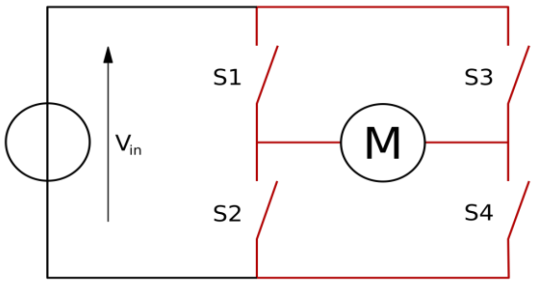

Figura 3.16: Esquemático de uma Ponte H. Fonte: BUTTAY (2006).

O circuito de ponte H é usado para determinar o sentido da corrente e valor de tensão no controle de um motor D. A imagem acima pode ser usada para ilustrar de modo genérico o funcionamento do circuito. Acionando-se em conjunto, as chaves S1 e S4, o terminal direito do motor fica com uma tensão mais positiva que o esquerdo, fazendo a corrente fluir da direita para a esquerda (no desenho, corrente convencional). Deste modo, o motor adquire sentido de giro que denotaremos por Sentido 1.

Acionando-se em conjunto as chaves S3 e S2, o terminal esquerdo do motor fica com uma tensão mais positiva que o direito, fazendo a corrente fluir da esquerda para a direita. Deste modo, o motor adquire sentido de giro que denotaremos por Sentido 2, que é inverso ao Sentido 1. Neste trabalho foi utilizado o módulo de Ponte H LN298N conforme a figura 3.17 abaixo:

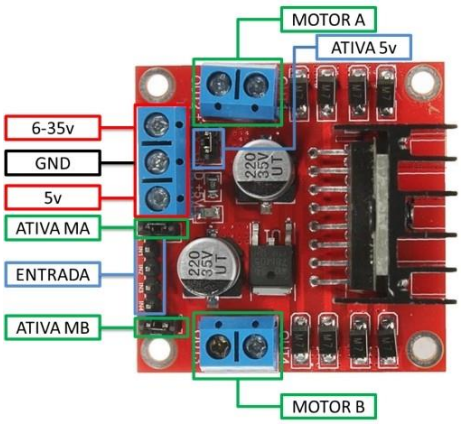

Figura 3.17: Ponte H LN298N. Fonte: Datasheet Ln298N adaptado.

- (Motor A) e (Motor B) se referem aos conectores para ligação de 2 motores DC ou 1 motor de passo;
- (Ativa MA) e (Ativa MB) – são os pinos responsáveis pelo controle PWM dos motores A e B. Se estiver com jumper, não haverá controle de velocidade, pois os pinos estarão ligados aos 5v. Esses pinos podem ser utilizados em conjunto com os pinos PWM do Arduino;
- (Ativa 5v) e (5v) – Este Driver Ponte H L298N possui um regulador de tensão integrado. Quando o driver está operando entre 6-35V, este regulador disponibiliza uma saída regulada de +5v no pino (5v) para um uso externo (com jumper), podendo alimentar por exemplo outro componente eletrônico. Portanto não alimente este pino (5v) com +5v do Arduino se estiver controlando um motor de 6-35v e jumper conectado, isto danificará a placa. O pino (5v) somente se tornará uma entrada caso esteja controlando um motor de 4-5,5v (sem jumper), assim poderá usar a saída +5v do Arduino;
- (6-35v) e (GND) – Aqui será conectado a fonte de alimentação externa quando o driver estiver controlando um motor que opere entre 6-35v. Por exemplo se estiver usando um motor DC 12v, basta conectar a fonte externa de 12v neste pino e (GND);
- (Entrada) – Este barramento é composto por IN1, IN2, IN3 e IN4. Sendo estes pinos responsáveis pela rotação do Motor A (IN1 e IN2) e Motor B (IN3 e IN4).

A tabela 3.1 abaixo mostra a ordem de ativação do Motor A através dos pinos IN1 e IN2. O mesmo esquema pode ser aplicado aos pinos IN3 e IN4, que controlam o Motor B.

Tabela 3.1: Ativação dos motores.

Motor	IN1	IN2
HORÁRIO	12V	GND
ANTI-HORÁRIO	GND	12V
PONTO MORTO	GND	GND
FREIO	12V	12V

3.5.3 Encoder e sensor de velocidade

Encoder é um dispositivo/sensor eletro-mecânico cuja funcionalidade é transformar posição em sinal elétrico digital. Com a utilização de encoders é possível quantizar distâncias, controlar velocidades, medir ângulos, número de rotações, realizar posicionamentos, rotacionar braços robóticos, etc.

O encoder é composto basicamente por um disco com marcações, um componente emissor e um receptor, como mostrado na figura 3.18. Os encoders ópticos utilizam led como o componente emissor e um sensor fotodetector como o receptor.

As marcações no disco possuem a funcionalidade de bloqueio e desbloqueio do feixe de luz do led para o fotodetector, desse modo, a medida que o disco vai girando, o fotodetector, juntamente com um circuito eletrônico, repassa para as saídas um sinal em forma de uma onda quadrada, proporcional ao número de marcações do encoder, de acordo com a resolução do mesmo. Logo, a resolução do encoder é o número de marcações presentes no disco do dispositivo, que equivale a quantidade de ondas quadradas, ou clock, gerado em uma volta do encoder. Como pode ser visto pela imagem figura 3.18:

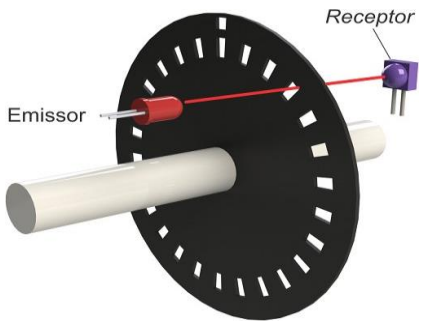

Figura 3.18: Exemplo de funcionamento de Enconder. Fonte: Almeida (2017).

Para ler o encoder é necessário um sensor optointerruptor, para ajudar no cálculo da velocidade das rodas. Tal como observado na figura 3.19.

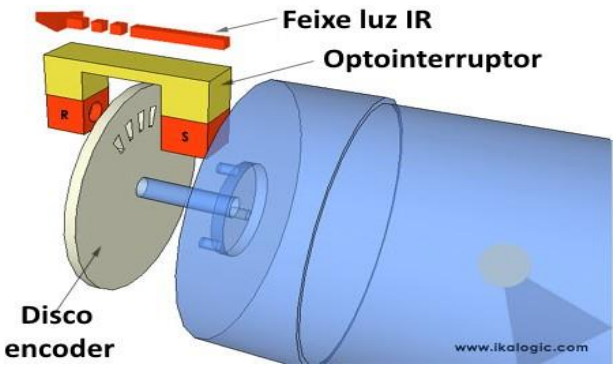

Figura 3.19: Exemplo de funcionamenteo de Enconder. Fonte: Almeida (2017).

Nesse projeto foi utilizado o sensor LM393 que possui uma tensão de operação: 3,3V – 5V DC, saída digital e analógica. Sua montagem no robô pode ser conferida na figura 3.20 abaixo; ele se conecta com as portas 38 e 39 do Arduino Mega que fica na tampa superior do robô conforme a figura.

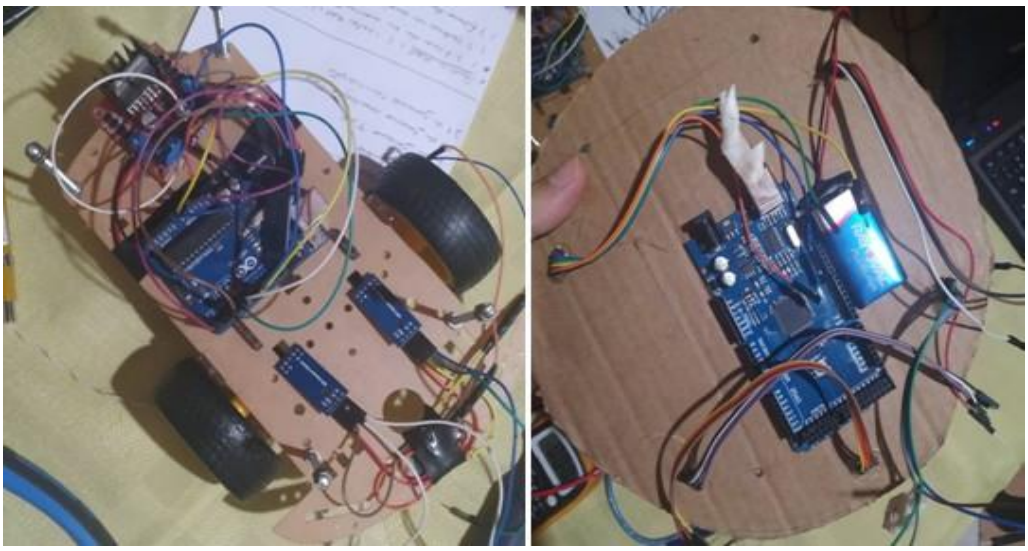

Figura 3.20: Sensor de velocidade LM393 encaixado nos Enconders durante a montagem de uma versão teste do robô. Fonte: Próprio autor.

3.5.4 Módulo Bluetooth e SD Card

Para comunicação foi utilizado o módulo Bluetooth HC-05, que possibilita transmitir e receber dados por meio de comunicação sem fio. Este módulo pode ser utilizado para criação de comunicação wireless para troca de informações entre dispositivos. No caso desta dissertação foi usado para se comunicar com o app e controlar o Robô físico. As principais características do módulo são:

- Tensão de operação: 3,6V – 6VDC;
- Frequência de operação: 2,4GHz;
- Nível de sinal lógico: 3,3V;
- Protocolo bluetooth: v2.0+EDR;
- Banda: ISM;
- Modulação: GFSK;
- Segurança: autenticação e criptografia;
- Modo de funcionamento: master / slave;
- Temperatura de operação: -40° a 105° celsius;
- Alcance do sinal: 10m;
- Senha padrão (PIN): 1234.

Figura 3.21: Módulo Bluetooth HC-05. Fonte: Site Baú da Eletrônica.

Para armazenar os dados dos sensores de distância e as velocidades dos motores calculadas pelos sensores encoder foi utilizado um módulo SDcard com cartão de memória de 8 GB. O módulo aceita cartões formatados em FAT16 ou FAT32 e utiliza a comunicação via interface SPI por meio dos pinos MOSI, SCK, MISO e CS, mostrados na figura 3.22 abaixo:

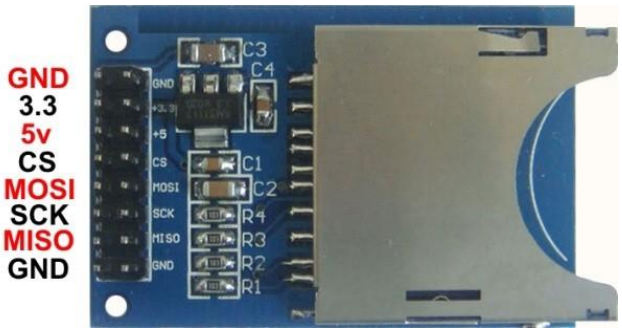

Figura 3.22: Módulo SD.Fonte: Site Baú da Eletônica.

- CS:

Chip Select. Dados são transferidos usando uma SPI (*Serial Peripheral Interface*) padrão. Isto significa que dispositivos podem transmitir dados usando a mesma linha de transmissão. O dispositivo irá receber todos os dados, mas só irá interpretá-los quando selecionado. Isto é o que este pino faz, quando em Zero qualquer dado é ignorado. Se estiver usando um dos exemplos da IDE Arduino este canal está conectado ao pino 4.

- MOSI:

Master Out Slave In. Esta é uma das formas de comunicação de dispositivos conforme comentado acima na SPI. Na configuração o microcontrolador é o Mestre, enquanto que o módulo é o escravo. Se estiver usando um dos exemplos da IDE Arduino este canal está conectado ao pino 11.

- SCK:

Serial Clock. Este é a saída do que for considerado o mestre que sincroniza os dados. Se estiver usando um dos exemplos da IDE Arduino este canal está conectado ao pino 13.

- MISO:

Master In Slave Out. Diferentemente do MOSI o Mestre é a entrada e o Escravo a saída. Conectado ao pino 12.

- GND:

  O primeiro estando conectado, este não precisa conectar.

O esquema de conexão nos arduinos com base nas portas está presente na tabela 3.2.

Tabela 3.2: Conexão nos arduinos com base nas portas.

	SCK	**MISO**	**MOSI**	**CS**
UNO	13	12	11	10
MEGA	52	50	51	53

3.5.5 Esquemático de Montagem do Protótipo Robô

Depois de construída a estrutura, foi elaborada a montagem do esquemático, seguindo o modelo projetado no ambiente Frietzing, como mostrado na figura 3.23. Os códigos de programação estão no repositório do laboratório LEIA/GRACO da UnB.

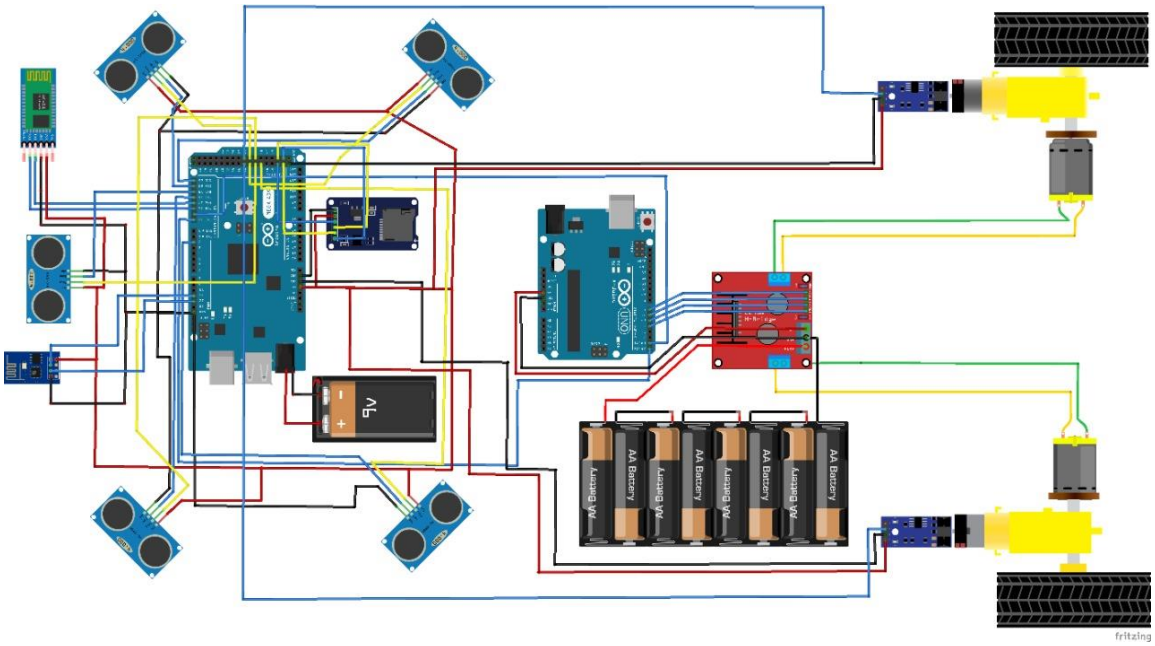

Figura 3.23: Esquemático do robô desenhado no programa Frietzing. Fonte: Próprio Autor.

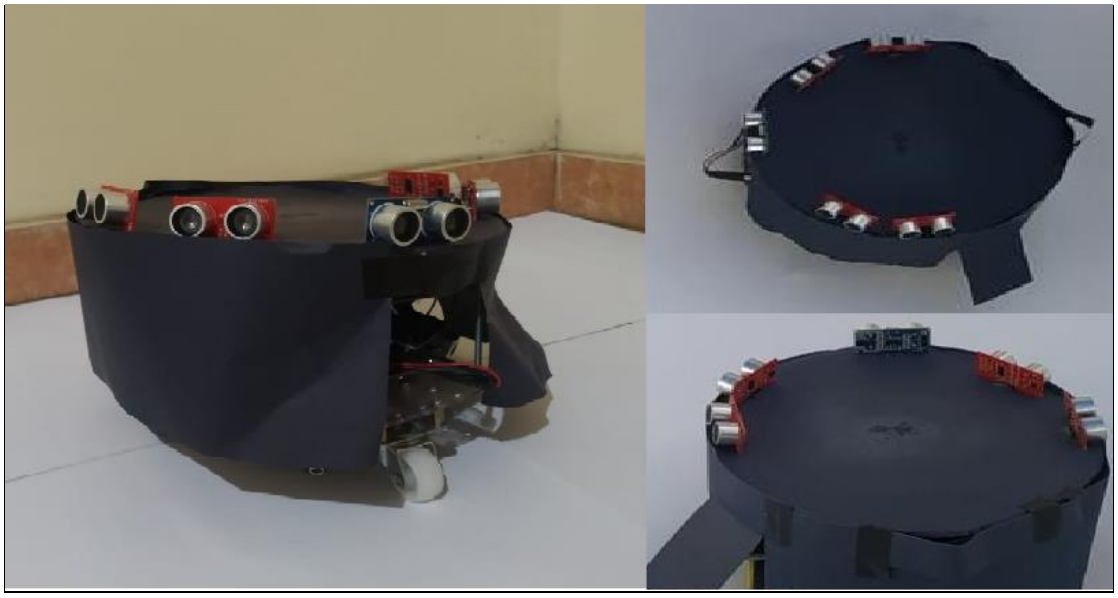

Figura 3.24: Protótipo Robô finalizado - vista frontal superior e traseira. Fonte: Próprio Autor.

Capítulo 4

Resultados e Discussão

Neste capítulo são apresentados os resultados obtidos no processo de verificação e validação do trabalho. Em uma primeira etapa foram realizadas diversas simulações e treinamentos das redes neurais artificiais e do robô de tração diferencial no simulador EyeSim. Após essa etapa de simulações foram realizados demonstrações e treinamentos das redes neurais a partir dos dados coletados no protótipo robô físico. Por último, este capítulo apresenta uma análise dos pesos sinápticos gerados, seus erros associados e uma comparação de trajetórias ensinadas e aprendidas em diversos ambientes estruturados e não estruturados.

4.1 Resultados das Simulações

O simulador EyeSim foi utilizado para realizar diferentes trajetórias com o robô por meio do teclado do computador. Dados em diferentes trajetórias percorridas pelo robô foram coletados e armazenadas em um arquivo .txt.

Cada trajetória de cada experimento tem uma quantidade diferente de dados relacionados com a trajetória demonstrada. Dessa forma, quanto mais dados coletados na fase de demonstração, melhor será o resultado do treinamento dos pesos sinápticos das redes utilizadas na fase de aprendizagem.

Conforme descrito na seção 3.2, uma rede neural artificial com camadas ocultas foi treinada no Weka, que otimiza a topologia de rede gerando como saídas um valor de velocidade angular e outro de velocidade linear.

4.1.1 Experimento 1- Contorno de obstáculos minimizando a distância

O principal objetivo do experimento 1 é ensinar o robô a contornar obstáculos minimizando a distância; com isso os pesos sinápticos correspondentes aos sensores frontais e diagonais do lado esquerdo foram um pouco maiores do que os pesos do lado direito.

Os dados das velocidades das rodas e sensores de distâncias coletadas durante a trajetória do experimento 1, no cenário em cruz, foram coletadas e posteriormente as redes foram treinadas no Weka. É possível visualizar, na figura 4.1, o resultado das trajetórias ensinadas e aprendidas no EyeSim.

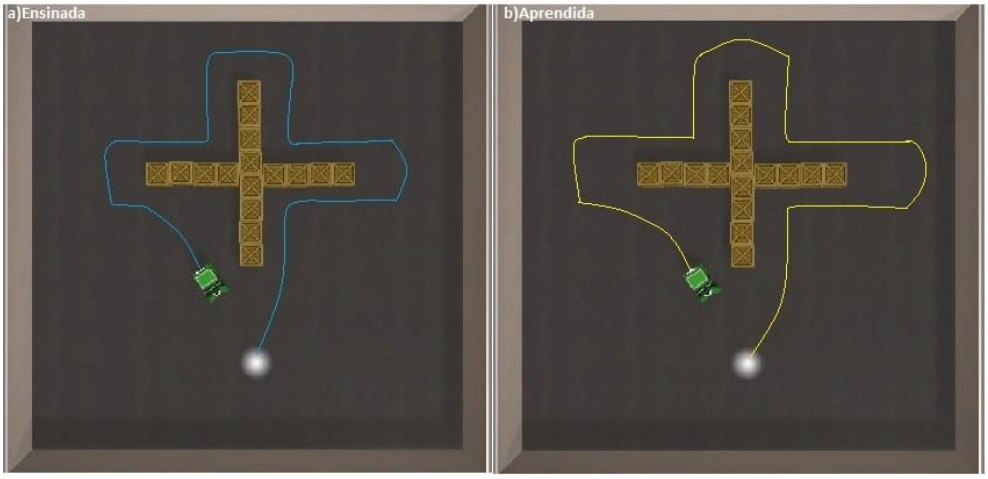

Figura 4.1: Resultado do experimento 1: trajetória ensinada (a) e trajetória aprendida (b). Fonte: Próprio autor.

Em seguida, utilizando a ferramenta “nntool” do MatLab, a fim de encontrar a melhor validação da performance e usando a mesma topologia da rede treinada no Weka (7-4-2), foi possível desenvolver o gráfico abaixo. Nele destacam-se o erro quadrático médio e o momento em que se conseguiu o menor erro com o conjunto de dados de validação em relação aos de treinamento e teste.

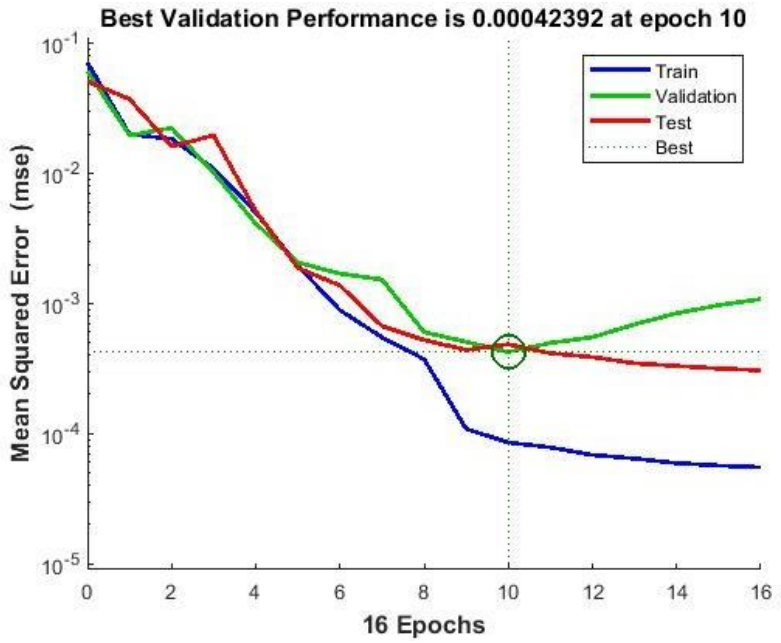

Figura 4.2: Resultado do experimento 1: melhor validação da performance com base no erro quadrático médio. Fonte: Próprio autor.

Foi utilizada regressão linear nos valores estimados de saída da rede neural com os dados reais de treinamento, validação e teste com o objetivo de visualizar a quantidade de acertos no conjunto de dados. A regressão linear obteve erro abaixo de 1% nos resultados, ou seja, 99% de proximidade do conjunto de dados.

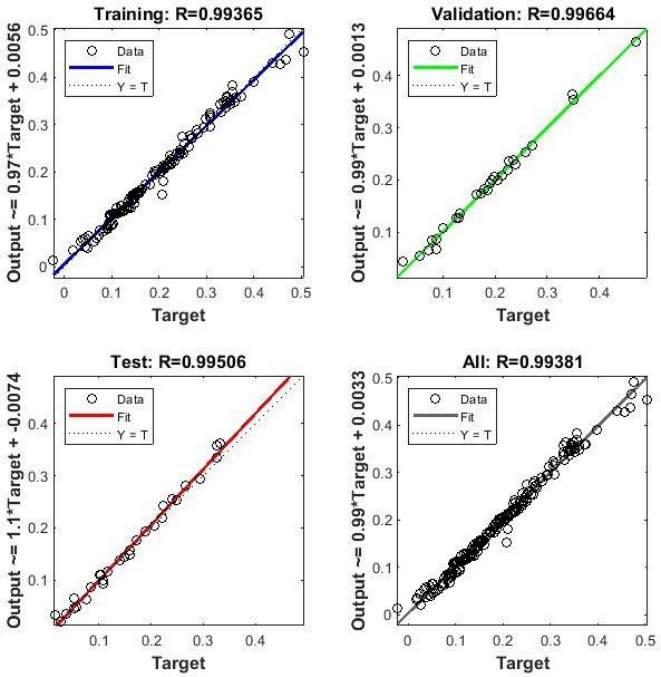

Figura 4.3: Resultado do experimento 1: Regressão linear da rede neural no conjunto de dados. Fonte: Próprio autor.

A análise estatística da rede treinada mostra o quadro resumo apresentado na tabela 4.1, demostrando que o processo de treinamento foi bem sucedido.

Tabela 4.1: Resumo dos Resultados do Experimento 1.

Variável	Valor
ρ (Rô de Spearman)	0.9960
T	0.9904
Erro Absoluto Médio	0.0012
Raiz Quadrada do Erro Médio	0.0016
Erro Relativo Absoluto	2.0429%
Erro Quadrático Relativo da Raiz	2.3144%

4.1.2 Experimento 2- Contorno de obstáculo em ambiente desconhecido

Neste experimento foi realizada uma simulação usando os pesos sinápticos calculados nas experiências, porém usando ambientes modificados para averiguar o comportamento em um ambiente desconhecido.

Primeiramente, o cenário do experimento 1 foi modificado, colocando objetos em formatos diferentes e construindo um cenário sem saída. O objetivo era observar se o robô escaparia e faria o caminho inverso ou se ele ficaria preso. O robô foi capaz de evitar a colisão com uma caixa frontal, contudo ficou preso em uma caixa lateral.

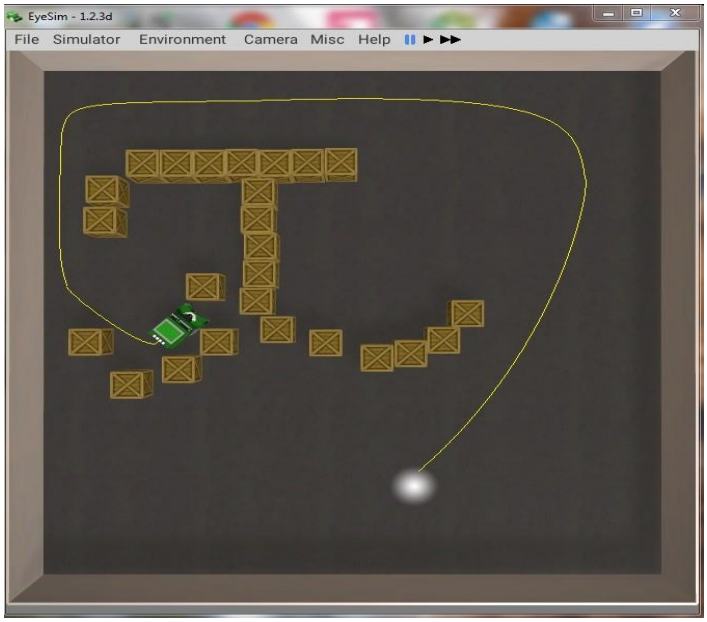

Figura 4.4: Resultado do experimento 2: trajetória em um ambiente não estruturado. Fonte: Próprio autor.

Após isso, o cenário do experimento foi modificado acrescentando objetos diferentes como bola de futebol, lata de refrigerante, etc. Esses objetos foram colocados onde o robô foi treinado para passar. O objetivo era observar se ele colidiria ou se desviaria dos objetos, ficando preso ou se chocando com a parede. O robô evitou a parede e tentou realizar o percurso original (ensinado). Contudo, como o sensor frontal possui uma influência de peso sináptico alto, o robô conseguiu desviar dos obstáculos.

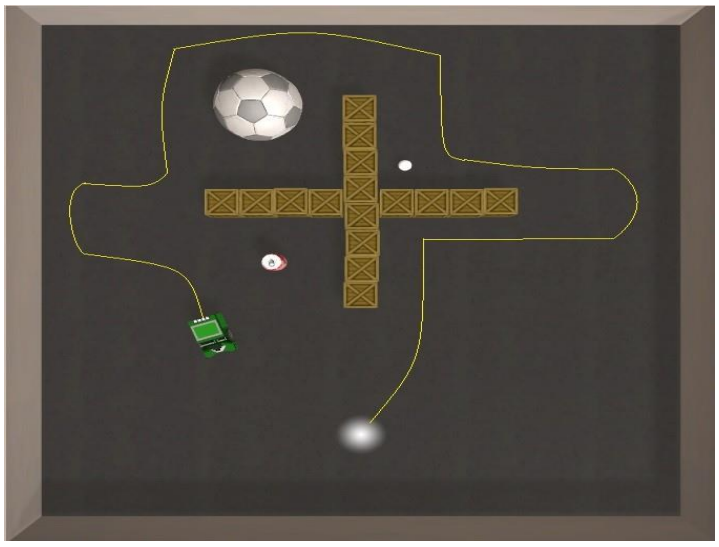

Figura 4.5: Resultado do experimento 2: trajetória “cruz” modificada. Fonte: Próprio autor.

4.2 Resultados com o Robô Físico

Nesses experimentos simulados o robô funcionou bem, com base nos resultados obtidos nas regressões lineares dos dados de simulação, treinamento, validação, teste e comparação das trajetórias aprendidas e simuladas. Além desses resultados, podemos ver o comportamento do robô em ambientes desconhecidos. Nos experimentos a seguir, considerando o caso mais complexo (experimento em cruz), foi montado um cenário físico próximo ao simulado para o robô físico e outros cenários em todos eles foram feitas diversas vezes a mesma trajetória para coleta de dados além de modificações de sentido, posição e mudança de objetos no cenário para análise do comportamento do robô físico.

4.2.1 Experimento 1 do Robô físico – Contornar Caixa

O objetivo deste experimento foi utilizar um protótipo de robô real adaptando o algoritmo de imitação para o arduino, apesar da limitada capacidade de processamento. Assim como nos experimentos anteriores, um ambiente foi preparado, nesse caso com apenas uma caixa ao centro. No simulador existia a função de visualizar o trajeto e sensores. Entretanto, no robô físico, a fim de ver a trajetória percorrida, foi colocada uma caneta preta de quadro (no furo traseiro da estrutura) para a trajetória ensinada e uma caneta azul para a trajetória aprendida, conforme a figura 4.6.

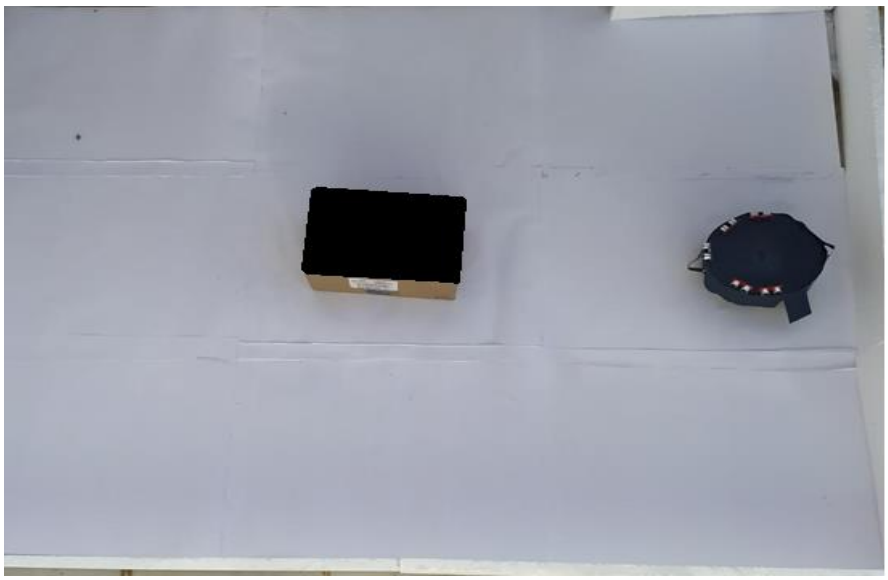

Figura 4.6: Robô físico com caneta presa em furo traseiro. Fonte: Próprio autor.

O robô foi controlado por um aplicativo Bluetooth RC controller, o qual permite enviar os comandos de direção conforme atribuído. Os dados de distâncias (coletados pelos sensores ultrassônicos) e das velocidades (pelos encoders das rodas) foram salvos no cartão de memória e enviados para nuvem através do sensor ESP 01.

Figura 4.7: Realização da trajetória ensinada. Fonte: Próprio autor.

Os pontos desenhados pelo robô foram manualmente interligados na cartolina. Foi retirada uma fotografia (que foi editada no *Paint)* resultando na trajetória. O arquivo em formato .csv que contém os dados coletados durante a trajetória, das velocidades e das distâncias, foi utilizado pelo Weka para treinar os pesos sinápticos da rede, resultando em uma topologia 5,3,2. Ou seja, 5 entradas, 3 neurônios na camada oculta e 2 saídas. Os pesos sinápticos foram transferidos pela esp 01 via wi-fi de volta no protótipo do robô, com o algoritmo para fazer a trajetória demonstrada de maneira autônoma. O robô fez o trajeto, com a caneta azul na parte traseira, tracejando o caminho percorrido.

A figura 4.8 mostra uma comparação das trajetórias ensinada e imitada. O robô foi capaz de fazer quase uma circunferência, porém não de forma perfeita e ainda colidiu levemente com a caixa ao imitar a trajetória ensinada.

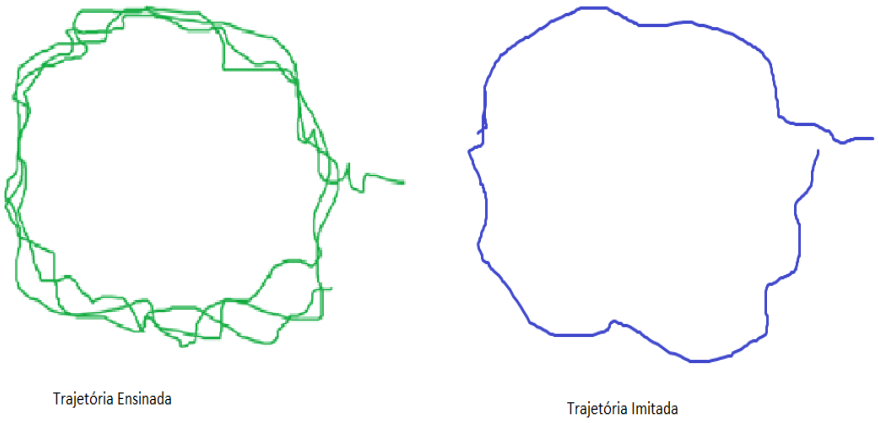

Figura 4.8: Resultado do experimento 1 do robô físico: trajetória ensinada X trajetória imitada. Fonte: Próprio autor.

A ferramenta “nntool” do MatLab foi usada para encontrar a melhor validação da performance. Foi utilizada a mesma topologia da rede treinada no Weka (5,3,2). O gráfico abaixo mostra o erro médio quadrático e o momento em que se conseguiu o menor erro.

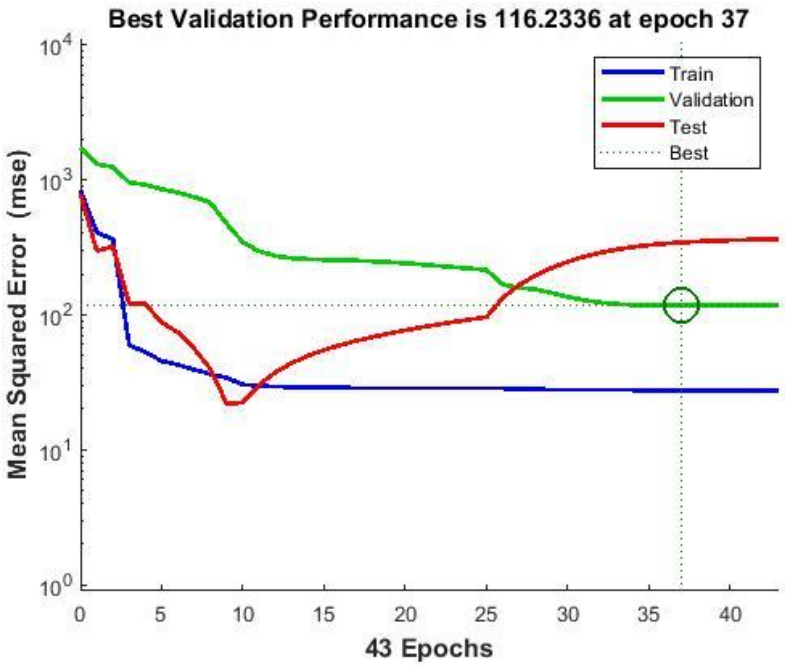

Figura 4.9: Resultado do experimento 1 do robô físico: Erro quadrático médio. Fonte: Próprio autor.

A fim de avaliar os resultados da rede neural treinada e dos dados coletados, foi realizada uma regressão linear com os dados de treinamento e de teste de saída da rede. Os resultados são vistos nos gráficos a seguir:

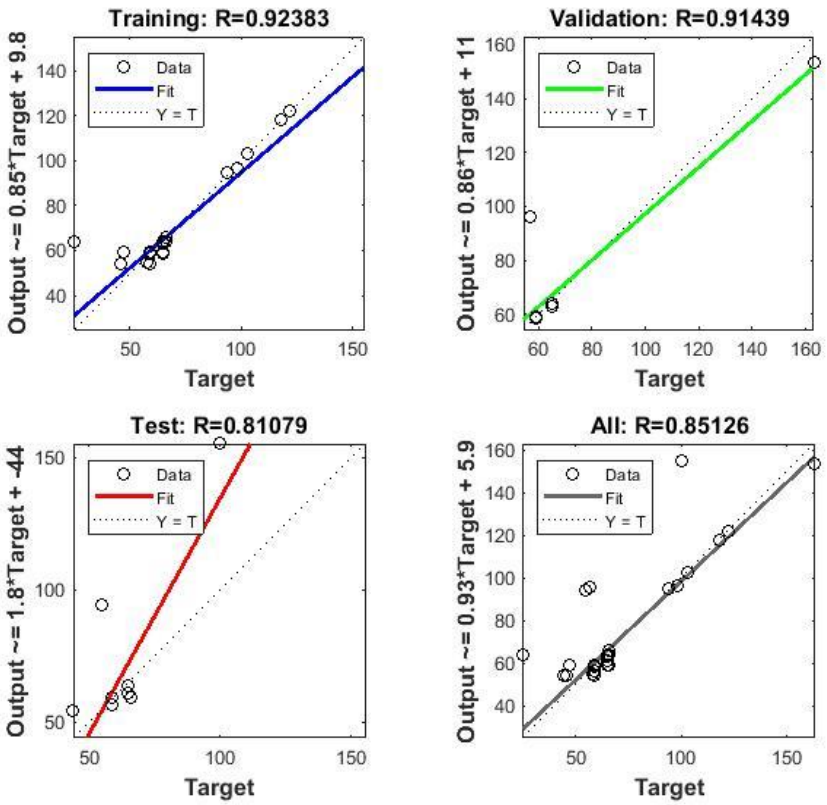

Figura 4.10: Resultado do experimento 1 do robô físico: Regressões Lineares do treinamento, validação saída e de teste. Fonte: Próprio autor.

Enquanto o robô simulado, no pior dos casos, teve precisão de 96%, o robô físico, usando o algoritmo de imitação no arduino, conseguiu 85% de precisão; um motivo para isso é a diferença na coleta de dados (quanto mais dados, mais preciso é a rede neural).

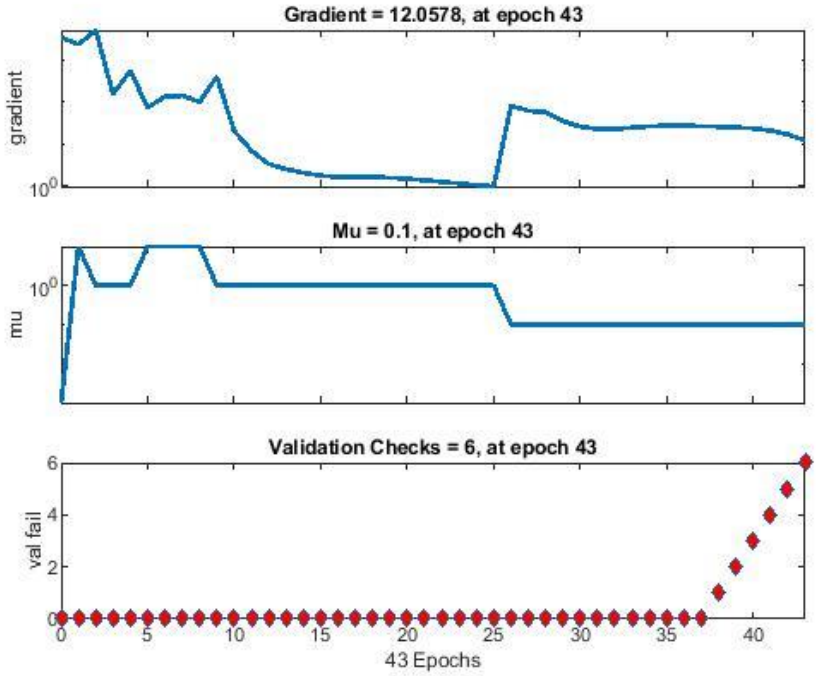

Figura 4.11. Desempenho da Rede Neural robô físico experimento caixa. Fonte: Pro´prio autor

O gradiente mostra o grau de inclinação da rede na superfície do erro quando busca alcançar o erro mínimo global. Portanto, em cada etapa do treinamento, o vetor de peso utilizado anteriormente é alterado com objetivo de provocar uma brusca queda na superfície do erro. Observa-se que quando a inclinação do gradiente diminui, a distância do erro aumenta (e vice-versa).

O Mu é um incremento adicionado para se chegar ao resultado final, enquanto que o gradiente procura obter o menor valor de erro. Portanto, à medida que o gradiente diminui, o Mu aumenta. Como o Mu é um incremento que acontece para se chegar ao mínimo valor, o MSE busca se aproximar do valor mínimo do erro. Durante a verificação de validação, indica que foram realizadas 6 iterações consecutivas e o resultado não sofreu mais alteração, com isso, o teste é finalizado, conforme destacado com um círculo na epoch 37.

4.2.2 Experimento 2 do Robô físico - Cenário em cruz físico

Este experimento ensinou o robô a percorrer o caminho de um cenário fechado em formato de cruz, semelhante ao experimento 1 dos resultados de simulação; enquanto ele se desviava das paredes e obstáculos, os dados foram coletados.

Figura 4.12: Resultado do experimento 2 do robô físico: Ensino do cenário em cruz físico ao robô. Fonte: Próprio autor.

O arquivo em formato .csv que contém os dados coletados, durante a trajetória, das velocidades e das distâncias, foi utilizado pelo Weka para treinar os pesos sinápticos da rede, resultando em uma topologia 5,3,2. Ou seja, 5 entradas, 3 neurônios na camada oculta e 2 saídas.

Os pesos sinápticos foram colocados de volta no protótipo para executar a trajetória de maneira autônoma. O robô fez o trajeto, com a caneta azul, tracejando o caminho percorrido.Adotada a topologia da rede neural 5,3,2, os resultados obtidos comparando as duas trajetórias (a ensinada e a imitada), podes ser vistos na figura 4.13. Dessa forma, nota-se que o robô foi capaz de realizar aproximadamente a trajetória, no entanto não nas mesmas proporções.

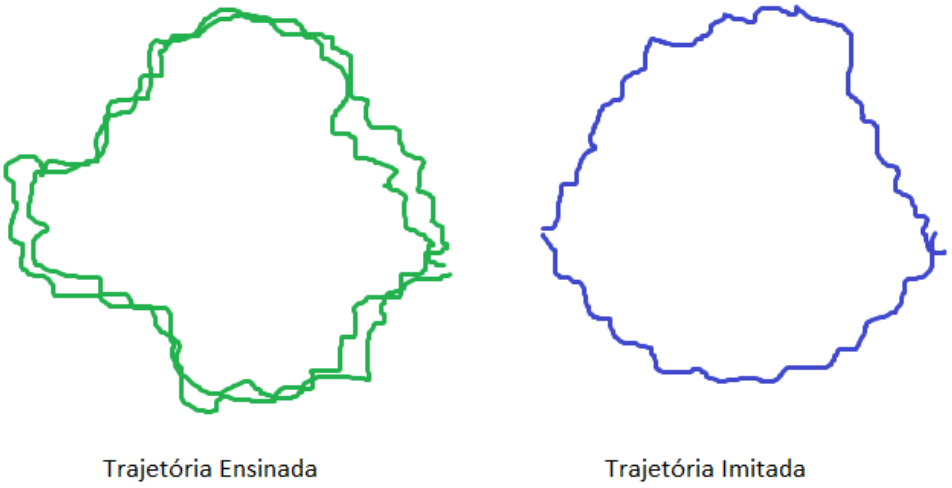

Figura 4.13: Resultado do experimento 2 do robô físico: trajetória ensinada X trajetória imitada. Fonte: Próprio autor.

O gráfico abaixo mostra o erro médio quadrático e o momento em que se conseguiu o menor erro.

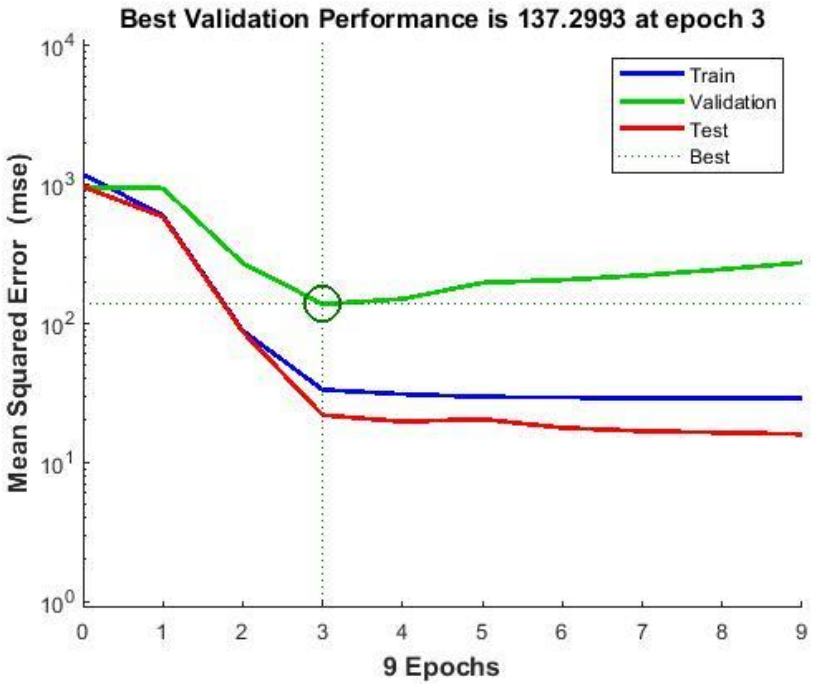

Figura 4.14: Resultado do experimento 2 do robô físico: Erro quadrático médio. Fonte: Próprio autor.

Os resultados da rede neural treinada e dos dados coletados, foi realizada uma regressão linear com os dados de treinamento e de teste de saída da rede. Os resultados são vistos nos gráficos a seguir:

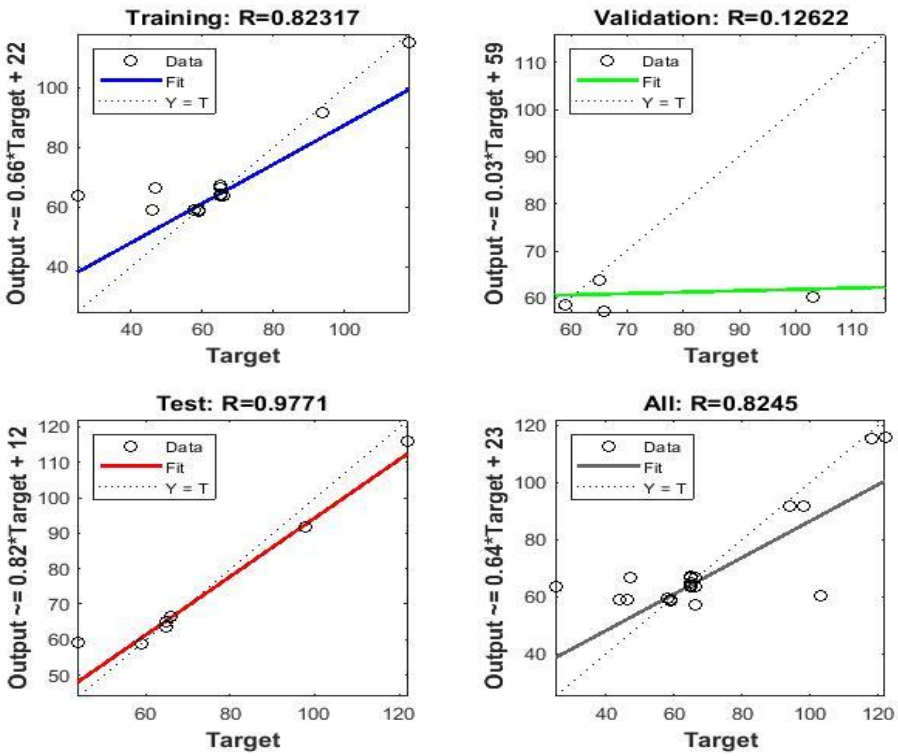

Figura 4.15: Resultado do experimento 2 do robô físico: Regressões Lineares do treinamento, validação saída e de teste. Fonte: Próprio autor.

O gráfico em azul representa os resultados de saída encontrados no treinamento da RNA de 82,3%; o segundo gráfico, em verde, representa a validação da rede com os dados separados conforme explicado na subseção 3.3.1 com 12,6% de acerto; o gráfico em cinza é a regressão linear da saída da rede neural, a qual obteve precisão de 82,4% e por último, o gráfico em vermelho é o teste de saída com o robô imitando e que obteve 97,2% de precisão.

O robô simulado em um cenário em cruz obteve 99,5% de precisão em teste, enquanto o real 82,4%.

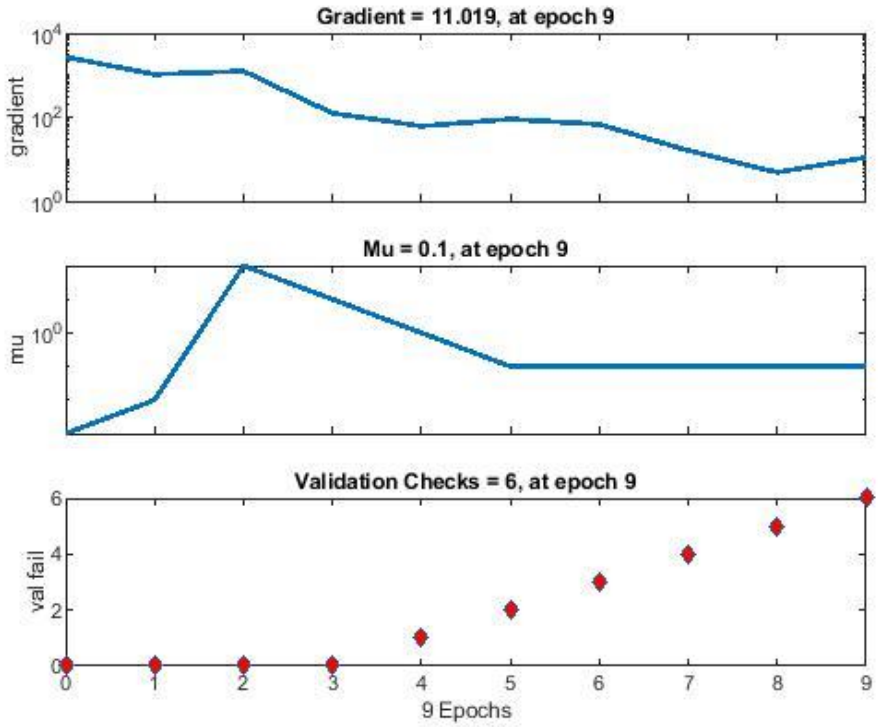

Figura 4.16. Desempenho da Rede Neural robô físico experimento cruz Fonte: Próprio autor

O processo de validação, indica que foram realizadas 6 iterações consecutivas e o resultado não sofreu mais alteração, com isso, o teste é finalizado, conforme destacado com um círculo na epoch 3.

4.2.3 Experimento 3 do Robô físico – Contornar parede

O objetivo desse experimento foi ensinar o robô a percorrer a parede de todo cenário sem colidir. O cenário foi montado utilizando cartolinas ao chão para marcação do trajeto.

Figura 4.17: Robô na posição inicial no cenário ensinado do experimento 3. Fonte: Próprio autor.

O arquivo em formato .csv que contém os dados coletados, durante a trajetória, das velocidades e das distâncias foi utilizado pelo Weka para treinar os pesos sinápticos da rede, resultando em uma topologia 5,3,2, ou seja 5 entradas, 3 neurônios na camada oculta e 2 saídas.

Os pesos sinápticos foram enviados de volta no protótipo para executar a trajetória de maneira autônoma. O robô fez o trajeto, com a caneta azul, tracejando o caminho percorrido. O resultado da comparação entre as duas trajetórias - ensinada e a imitada, pode ser visto na figura 4.18. Com isso, nota-se que o robô foi capaz de realizar as curvas sem colidir com a parede, no entanto não nas mesmas proporções.

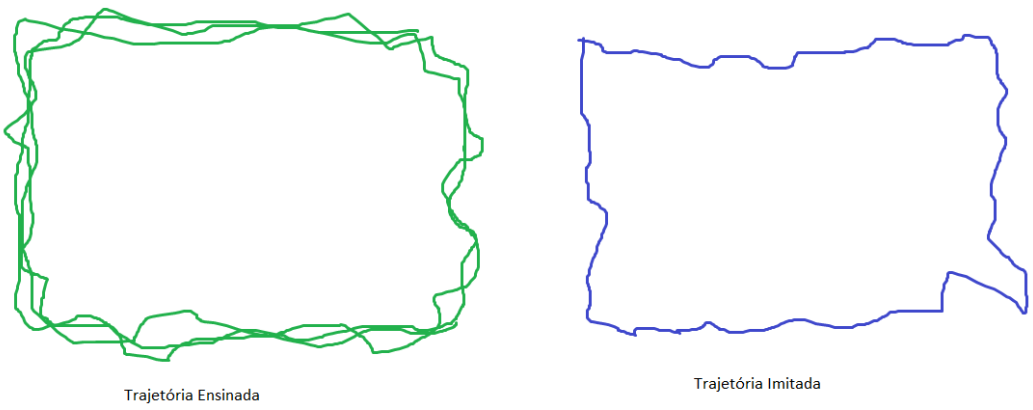

Figura 4.18: Resultado do experimento 3 do robô físico: trajetória ensinada X trajetória imitada. Fonte: Próprio autor.

O gráfico abaixo mostra o erro médio quadrático e o momento em que se conseguiu o menor erro.

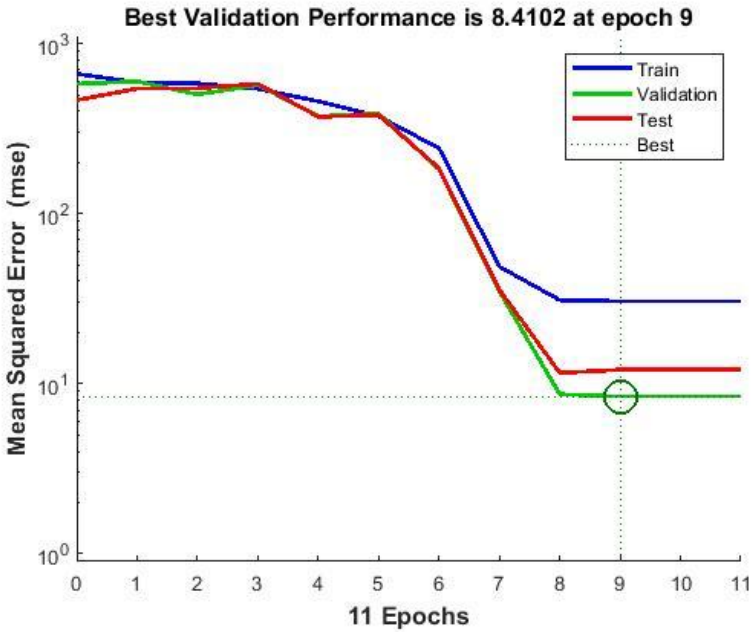

Figura 4.19: Resultado do experimento 3 do robô físico: Erro quadrático médio. Fonte: Próprio autor.

A fim de avaliar os resultados da rede neural treinada e dos dados coletados, foi realizada uma regressão linear com os dados de treinamento e de teste de saída da rede. Os resultados são vistos nos gráficos a seguir:

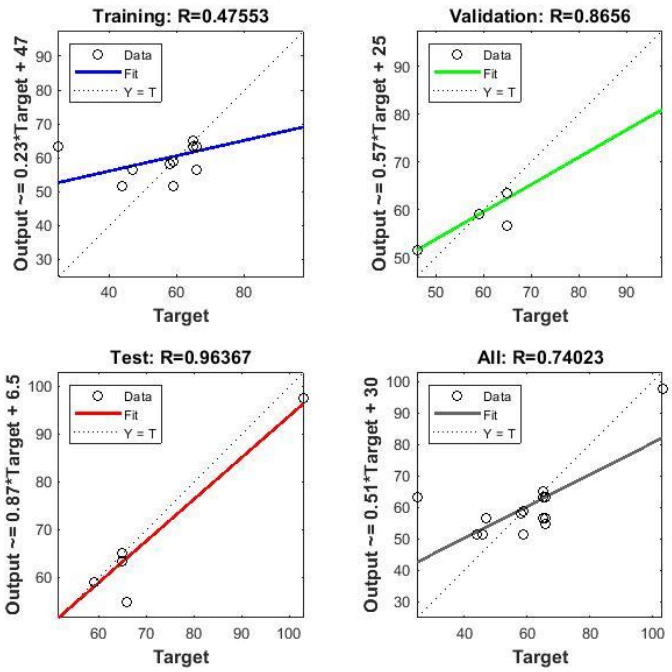

Figura 4.20: Resultado do experimento 3 do robô físico: Regressões Lineares do treinamento, validação saída e de teste. Fonte: Próprio autor.

O robô físico, no arduino, conseguiu 74% de precisão no treinamento da RNA; similar ao experimento anterior.

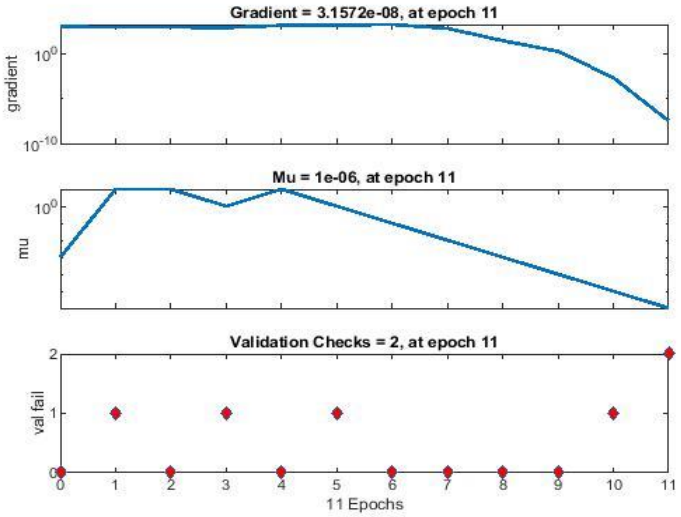

Figura 4.21. Desempenho da Rede Neural robô físico experimento parede. Fonte: Próprio autor.

O gradiente mostra o grau de inclinação da rede na superfície do erro quando busca alcançar o erro mínimo global. O Mu = 1e-06 é um incremento adicionado para se chegar ao resultado final, enquanto que o gradiente procura obter o menor valor de erro. Portanto, à medida que o gradiente diminui, o Mu aumenta. Como o Mu é um incremento que acontece para se chegar ao mínimo valor, o MSE busca se aproximar do valor mínimo do erro. Durante a verificação de validação, indica que foram realizadas apenas 2 iterações consecutivas e o resultado não sofreu mais alteração, com isso, o teste é finalizado, conforme destacado com um círculo na epoch 9.

4.2.4 Resumo dos Resultados com o Robô físico

Com o objetivo de comparar as melhores topologias, foram simuladas (utilizando o Weka e o MatLab) as topologias de RNA´s 5,3,2 ; 5,4,2; 5,5,2 e 5,4,3,2 . Seus resultados principais, considerando as regressões lineares com os dados de treinamento, validação, teste e saídas das redes, estão organizados nas tabelas abaixo. Os gráficos com o resultado desses

experimentos podem ser vistos no apêndice, junto com o gradiente, regressões lineares, Mu e erro quadrático médio para validação da performance.

Tabela 4.2: Resumo dos Resultados do Experimento com robô físico cenário caixa com diferentes topologias.

Experimento Caixa				
Topologia RNA´s	5,3,2	5,4,2	5,5,2	5,4,3,2
Treinamento	92,38%	94,99%	37,01%	95,12%
Validação	91,44%	100%	16,14%	99,68%
Teste	81,08%	86,85%	100%	69,02%
Saídas da rede	85,13%	88,39%	26,37%	95,38%

Tabela 4.3: Resumo dos Resultados do Experimento com robô físico cenário parede com diferentes topologias.

Experimento Parede				
Topologia RNA´s	5,3,2	5,4,2	5,5,2	5,4,3,2
Treinamento	47,55%	86,67%	72,32%	94,42%
Validação	86,56%	97%	70,73%	86,57%
Teste	96,37%	13,61%	100%	1,44%
Saídas da rede	74,02%	73,38%	73,38%	75,55%

Tabela 4.4: Resumo dos Resultados do Experimento com robô físico cenário cruz com diferentes topologias.

Experimento Cruz				
Topologia RNA´s	5,3,2	5,4,2	5,5,2	5,4,3,2
Treinamento	82,32%	12,30%	61,17%	32,97%
Validação	12,62%	38%	99,95%	99,90%
Teste	97,71%	28,83%	44%	53,37%
Saídas da rede	82,45%	3,82%	59,55%	59,55%

Capítulo 5

Conclusões e Trabalhos Futuros

Após estudo e fundamentação teórica sobre Redes Neurais Artificiais e *Learning from Demonstration* aplicado a robótica móvel e com base nos resultados em várias ferramentas para coleta de dados usando protótipo de robô satisfatórios e no EyeSim, considera-se que os resultados foram satisfatórios.

Os valores encontrados das RNA's, depois de treinadas, estão bem próximos, com precisão de 99% a 96% de acerto, nos experimentos 1 e 2 utilizando a coleta de dados do simulador EyeSim. Observou-se que o coeficiente de correlação de postos de Spearman, que indica que entre as saídas esperadas e as saídas obtidas pela rede existe uma forte correlação (0,99 próxima de 1), portanto, demonstrando a efetividade do processo de treinamento.

Com relação ao processo de imitação foi observado que o erro quadrático médio foi 0,16% e o erro relativo absoluto 2,047%. Os resultados simulados de aprendizagem com RNAs ficaram próximos de 99% de acerto.

O protótipo do robô físico funcionou parcialmente como esperado devido àcomplexidade do algoritmo de backpropagation. Enquanto o robô se locomovia e atualizava os pesos sinápticos com um custo computacional elevado para um arduino Mega, foi necessário reduzir sua velocidade para melhorar seu desempenho. Uma solução melhor para esse problema seria utilizar um microcontrolador com maior capacidade ou até mesmo um sistema embarcado com raspberrypi.

Enquanto o robô simulado no pior dos casos teve precisão de 96%, o robô físico usando o backpropagation no arduino consegue 85,2% de precisão com base nos dados e nas regressões.

Como trabalhos futuros, espera-se realizar as seguintes tarefas: uso de plataformas SoC (System on Chip) de forma a acelerar os algoritmos de treinamento em hardware usando FPGAs (Field Programmable Gate Array) e usar os processadores ARM para executar as redes neurais e realizar a tomada de decisão. Uma possibilidade é usar SoCs da família Zynq, que contém em um único chip um FPGA, além de microprocessadores ARM, comparando o resultado de aprendizagem de redes neurais em diferentes arquiteturas.

Recentemente a Xilinxs lançou, baseado na ZedBoard, o Zrobot, na Xilinx University Program, um robô que desvia de obstáculos feito com FPGA, porém não existe artigo de um robô físico feito usando FPGAs operando com algoritmo de RNA's e LfD, o que pode ser outra possibilidade de trabalho futuro.

Outra opção é fazer toda a parte de treinamento em uma plataforma de desenvolvimento ARM tal como o Raspberry Pi, implementando algoritmos que calculam o peso sináptico das Redes Neurais, não sendo mais necessário a utilização nem do Weka e nem do MatLab para treinamento das redes.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] ABBEEL, Pieter; NG Y., Andrew. Pprenticeship Learning via Inverse Reinforcement Learning. Computer Science Department, Stanford University, Satanford, 2004.

[2] ALMEIDA, Fernanda. O que é Encoder? Para que serve? Como escolher? Como interfacear?. Dez, 2017. Disponível em: https://www.hitecnologia.com.br/blog/o-que-%C3%A9-encoder-para-que-serve-como-escolher-como-interfacear/. Cesso em: 25 ago. 2019.

[3] ARGALL, Brenna D. et al. A survey of robot learning from demonstration. Robotics and Autonomous Systems, vol. 57, pg. 469-483, 2009.

[4] BARRETO, J. M. Inteligência Artificial no Limiar do Século XXI: Abordagem Híbridda, Simbólica, Conexionista e Evolucionária. Florianópolis: UFSC, 2001.

[5] BISHOP, Christopher M. Neural Networks for Pattern Recognition. Oxford University Press, Clarendon Press, 1995.

[6] BITTENCOURT, G. Inteligência Artificial – Ferramentas e Teorias. Florianópolis: UFSC, 1998.

[7] BORENSTEIN J; KOREN, Y. Real-time obstacle avoidance for fast mobile robots in cluttered environments. IEEE, Cincinnati, OH, USA, vol 1, pg. 572-577, 1990.

[8] BRAÜNL, Thomas. Embedded Robotics: Mobile Robot Design and Application with Embedded System. 2ª ed. Alemanha: Springer, 2006.

[9] BROOKS, Rodney A. A Robot that Walks; Emergent Behaviors from a Carefully Evolved Network. Journal of Robotics and Automation, vol. 1 , pg. 253-262, 1989.

[10] BRYS, Tim et al. Reinforcement Learning from Demonstration through Shaping. Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI), pg. 3352-3358, 2015.

[11] BUTTAY, Cyril. Structure of an H-bridge. 9 jun. 2006. Disponível em: https://commons.wikimedia.org/wiki/File:H_bridge.svg. Acesso em: 25 ago. 2019.

[12] DANIEL, C; NEUMANN, G; PETERS, J. Learning Concurrent Motor Skills In Versatile Solution Spaces. IEEE International Conference on Roboticas and Intelligent Systems (IROS 2012), pg. 3591-3597, 2012.

[13] DAUTENHANH, K; NEHANIV, C. Imitation in Animals and Artifacts. Londres: The MIT Press, 2002.

[14] DA SILVA, L. R. (2003). Análise e Programação de Robôs Móveis Autônomos da Plataforma Eyebot. (Dissertação de Mestrado, Programa de Pós-Graduação em Engenharia Elétrica) Universidade do Estado de Santa Catarina – UFSC. Florianópolis SC.

[15] DEJAN. Arduino Robot Car Wireless Controll using HC-05 Bluetooth, NRF24L01 and HC-12 Transceiver Modules. Disponível: https://howtomechatronics.com/tutorials/arduino/arduino-robot-car-wireless-control-using-hc-05-bluetooth-nrf24l01-and-hc-12-transceiver-modules/. Acesso em: 05 de dez. 2018.

[16] DEL LAMA, Rafael. Treinaemento perceptron multicamadas. 10 de nov. 2016. Disponível em: https://commons.wikimedia.org/wiki/File:Imgperceptron.png. Acesso em: 20 de out. 2018.

[17] DEMUTH, H.; BEALE M. Neural Network Toolbox. EUA: The MathWorks, 2016.

[18] FREEMAN, James A.; SKAPURA, David M. Neural Networks: Algorithms, Applications, and Programming Techniques. EUA: Addison-Wesley Publishing Company, Inc., 1991.

[19] GRILLNER, Sten; GEORGOPOULOS, A. P.; JORDAN, L. M. Neurons, networks, and motor behaviour. Londres: The Mit Press, 1997.

[20] GROLLMAN, Daniel H. JENKINS, Odest C. Incremental learning of subtasks from unsegment demonstration. 2010 IEEE/RSJ International Conference on Intelligent Robots ans Systems, Tipei, pg. 261-266, 2010.

[21] HARTMANN, Björn et al. Authoring Sensor-based Interactions by Demonstration with Direct Manipulation and Pattern Recognition. CHI 2007 Proceedings Ubicomp Tools. Califórnia, EUA, pg. 145-154, 2007.

[22] HAYKIN, Simon. Neural Networks and Learning Machines. 3ª Ed. Pearson Education, 2009.

[23] HAYKIN, Simon. Redes Neurais: Princípios e Prática. 2ª Ed. Artmed Editora S.A., 2001.

[24] HOSCH, Willian L. Machine Learning: artificial intelligence. Encyclopaedia Brittannica, 2019. Disponível em: https://www.britannica.com/technology/machine-learning. Acesso em: 25 outubro de 2018.

[25] JONES, Tim M. Um mergulho profundo nas redes neurais recorrentes. 05 de set. de 2017. Disponível em: https://imasters.com.br/data/um-mergulho-profundo-nas-redes-neurais-recorrentes. Acesso em: 20 de out. de 2018.

[26] JUNIOR, Celso de S.; HEMERLY, Elder M. Controle de robôs móveis utilizando o modelo cinemático. Sba Controle e Automação, Campinas, vol.14, no.4, pg. 384-392, outubro, novembro e dezembro, 2003.

[27] KAELBLING, L. P. et al. Reiforcement Learning: A Survey. Jounal of Artificial Intelligence Research (JAIR), vol.4, pg. 237-285, 1996.

[28] KORMUSHEV P.; CALINON, S.; CALDWELL, D. G. Robot Motor Skill Coordination with EM-bases Reinforcement Learning. The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, Taiwan, pg. 3232-3237, out. 2010.

[29] KOVÁCS, Z. L. Redes Neurais Artificiais: Fundamentos e Aplicações, 2ª ed. São Paulo: Academica, 1996.

[30] KULIC, D. et al. Incremental learning of full body motion primitives and their sequencing through human motion observation. The International Journal of Robotics Research, vol. 31, no.3, pg. 330-345, 2011.

[31] LEE, D; OTT, C. Incremental Kinesthetic Teaching of Motion Primitives Using the Motion Refiniment Tube. Autonomous Robots, vol. 31, no. 2, pg. 115-131, 2011.

[32] LIN, Long Ji. Programming Robots Using Reinforcement Learning and Teaching. In AAAI, PG. 781-786, 1991.

[33] MAES, Pattie. Modeling Adaptive Autonomous Agents. MIT Press, Cambridge, 1993.

[34] MCCULLOCH, W. S.; PITTS, W. A. A logical calculus of the ideas immanent in nervous activity. Bulletin of Mathematical Biophysics, vol.5, pg. 115-133, 1943.

[35] MEDINA-SANTIAGO, A., et al. Neural Control System in Obstacle Avoidance in Mobile Robots Using Ultrasonic Sensors. Jpurnal of Applied Research and Technology, vol. 12, pg. 101-110, fev. 2014.

[36] MITCHELL, Tom M. Machine Learning. EUA: McGraw – Hill, Inc, 1997.

[37] MÜLLING, K. et al. Learning to select and generalize striking movements in robot table tennis. The International Journal of Robotics Research, vol. 32, no. 3, pg. 263-279, 2013.

[38] NAKAISHI, Jun et al. Learning from demonstration and adaptation of biped locomotion. Robotics and Autonomous Systems, vol. 4 pg. 79-91, 2004.

[39] NAKASHIMI, J. et al. Learning from demonstration and adaptation of biped locomotion. Robotics and Autonomous Systems - Elsevier, no.47, pg. 79-91, 2004.

[40] NARDI, Marloni. Universo Robôs #1 - Como Fazer um Robô que Desvia de Obstáculos com Arduino e Sensor ultrassônico. Disponível em: https://www.marlonnardi.com/p/universo-robos-1-como-fazer-um-robo-que.html. Acesso em: 05 de dez. 2018.

[41] PASTOR, Peter et al. Learning and Generalization of Motor Skills by Learning from Demonstration.

[42] PERICO, D.; BIANCHI, R. Uso de Heurísticas Obtidas por meio de Demonstraçôes para aceleração do Aprendizado por Reforço. Brazillian Symposium on Intelligent Automation, São João Del Rei, vol. 10, set. 2011.

[43] POTOCNIK, Primoz. Neural Networks: MATLAB examples. University of Ljubljana, Faculty of Mechanical Engeneering, 2012.

[44] RICH, E; KNIGHT,K. Inteligencia artificial. 2ª Ed. Makron Books do Brasil,1994.

[45] ROISENBERG, Mauro. Emergência da inteligência em agentes autônomos através de modelos inspirados na natureza. 1998. no 226. Tese de Doutorado. Departamento de Engenharia Elétrica, Grupo de Pesquisas em Engenharia Biomédica - UFSC, Florianópolis, 1998.

[46] RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagation errors. Natures vo. 323, pg. 533-536, 1986.

[47] SCHAAL S. Learning From Demonstration. ATR Human Information Processing. College of Computing, Georgia Tech. MIT PRESS, 1997.

[48] SCHALKOFF, Robert J. Artificial Intelligence Engine. EUA: McGraw – Hill, Inc, 1990.

[49] SEVERO, Diogo da S. Otimização Global em Redes Neurais Artificiais. 2010. No 63. Projeto de Graduação apresentado no Centro de Informática da Universidade Federal de Pernambuco – UFPE, Recife, 2010.

[50] SILVEIRA, Rodrigo S. et al. Célula Neral Artificial Paraconsistente em Aplicação de Técnicas de Aprendizagem Apliadas à Automação. Anais do Encontro Nacional de Pós-Graduaçãi, vol.2 pg 385-389, 2018.

[51] SILVER, D.; BAGNELL, A. J.; STENTZ, A. Learning from Demonstration for Autonomous Navigation in Complex Unstructured Terrain. The International Jounal of Robotics Reasearch., vol. 29, no 12, pg. 1565-1592, 2010.

[52] SKOGLUND, A et al. Programming by Demonstration of Pick-and-Place Tasks for Industrail Manipulators using Task Primitives. 2007 International Symposium on Computational Intelligence in Robotics and Automation, IEEE, Jacksonville, pg. 368-373, 2007.

[53] SMART, William D. and KAELBLING, Leslie P. Effective Reinforcement Learning for Mobile Robots. Washington, DC, Proceedings of the 2002 IEEE International Conference on Robotics 8 Automation, pg. 3404-3410, maio 2002 .

[54] STUART, Perry; LING, Guan. A Recurrent Neural Ntwork for Detecting Objects in Sequences of Sector-Scan Sonar Images. IEEE Journalof Oceanic Engeneering, vol.24, no. 3, pg. 857-871, 2004.

[55] SUNNY, S. H. et al. An Autonomous Robot: Using ANN to Navigate in a Static Path. 4th International Conference on Advances in Electrical Engineering (ICAEE), Bangladesh, pg. 291-296, set. 2017.

[56] SUTTON, R. S., BARTO, A. G. Reinforcement Learning: An Introduction. 1ª Ed. Londres: The MIT Press, 1998.

[57] TAFNER, Malcon A. Redes neurais artificiais: aprendizado e plasticidade. Cérebro Mente, São Paulo, vol. 5, 1998.

[58] VASSALLO, R. F.; SANTOS-VICTOR, J. A. SCNEEBELI, H. J. Aprendizagem por Imitação através de Mapeamento Visuomotor Baseado em Imagens Omnidirecionais. Revista Controle & Automação, vol. 18, no. 1, pg 1-12, 2007.

[59] VIEIRA, Rodrigo S. Protótipo de um sistema de monitoramento remoto inteligente. 1999. 155p. Curso de pós-graduação em Engenharia de Produção. Repositório Institucional UFSC. Disponível em: https://repositorio.ufsc.br/handle/123456789/81296. Acesso em: 17 de out. de 2018.

[60] WITTEN, Ian H. Data Mining with Weka, 2016. Disponível em: https://www.futurelearn.com/courses/data-mining-with-weka/0/steps/25384. Acesso em: 19 ago. 2019.

[61] WU, Y; DEMIRIS, Y. Towards One Shot Learning by imitation for humanoid robots. IEEE-RAS Int.Conf on Robotics and Automation (ICRA), 2010.

APÊNDICE A

Experimento **topologia de rede neural 5,4,3,2,** gradiente e regressões lineares; validação, treinamento, teste e saída rede dos experimentos:

- Experimento Robô físico contornar caixa:

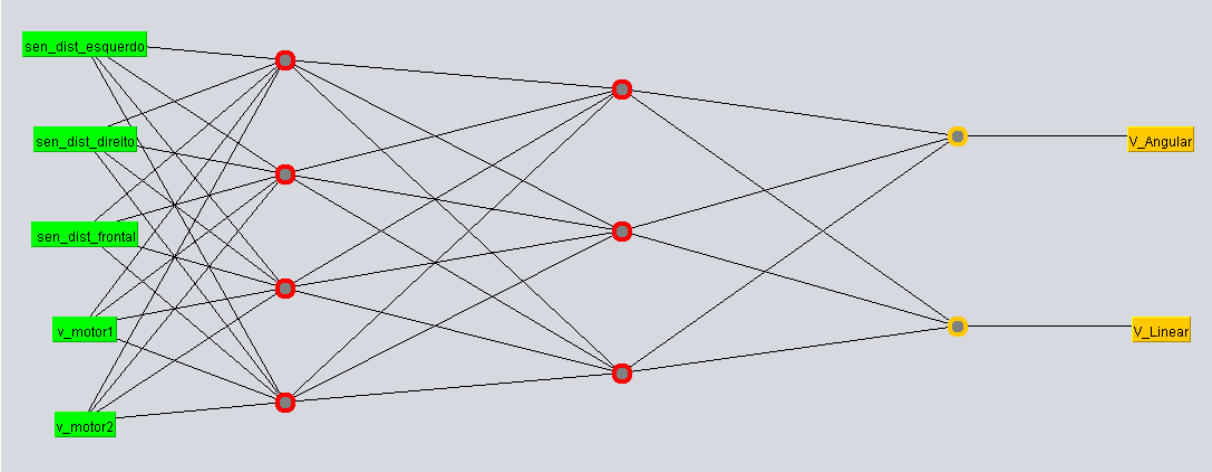

Figura A-1 : Topologia rede neural 5,4,3,2 simulado no Weka. Fonte: Próprio autor.

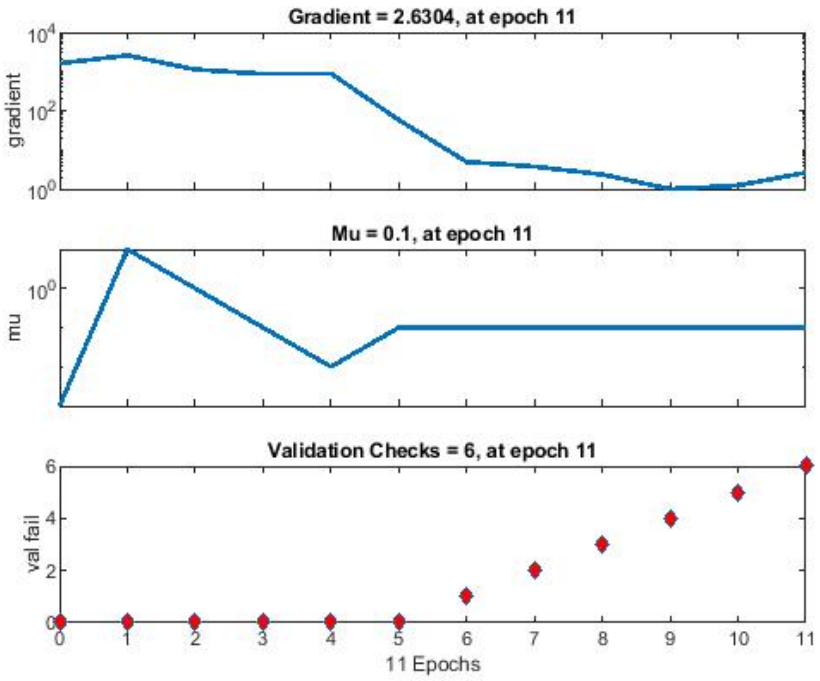

Figura A-2: Desempenho da Rede Neural robô físico topologia 5,4,3,2, experimento caixa. Fonte: Próprio autor.

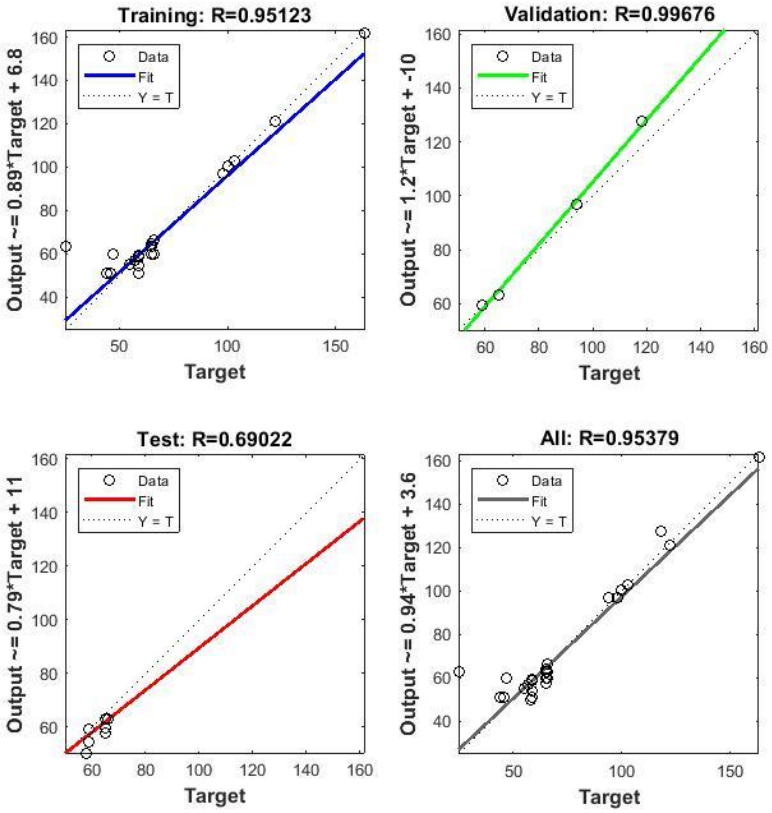

Figura A-3: Regressões Lineares do treinamento, validação saída e de teste experimento caixa: 5,4,3,2. Fonte: Próprio autor.

Experimento Robô físico cruz topologia 5,4,3,2:

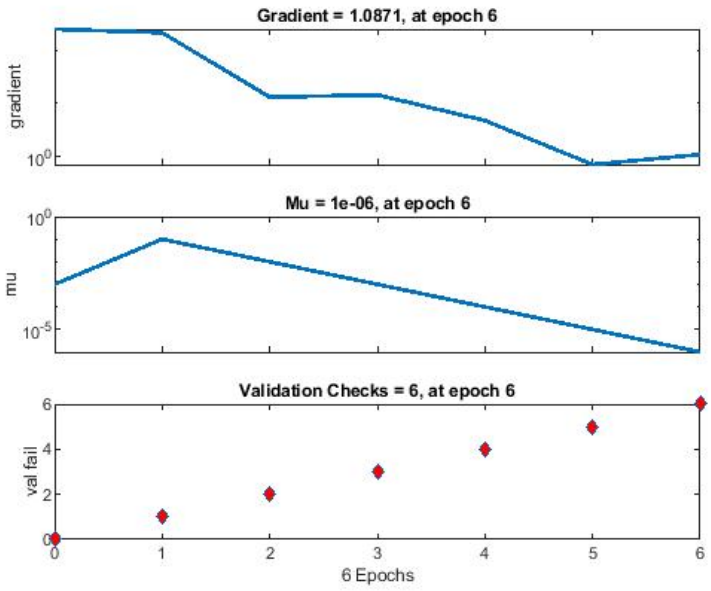

Figura A-4: Desempenho da Rede Neural robô físico topologia 5,4,3,2,.experimento cruz. Fonte: Próprio autor.

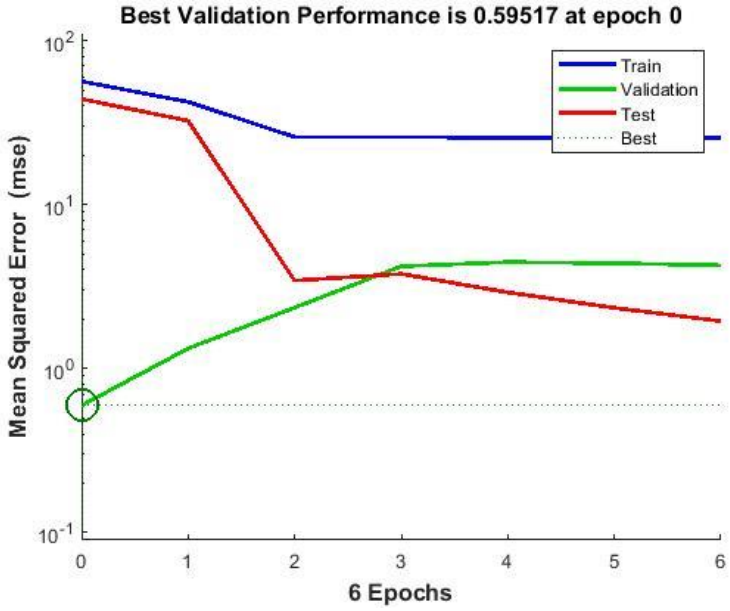

Figura A-5: Resultado do experimento cruz: Erro quadrático médio topologia 5,4,3,2. Fonte: Próprio autor

.

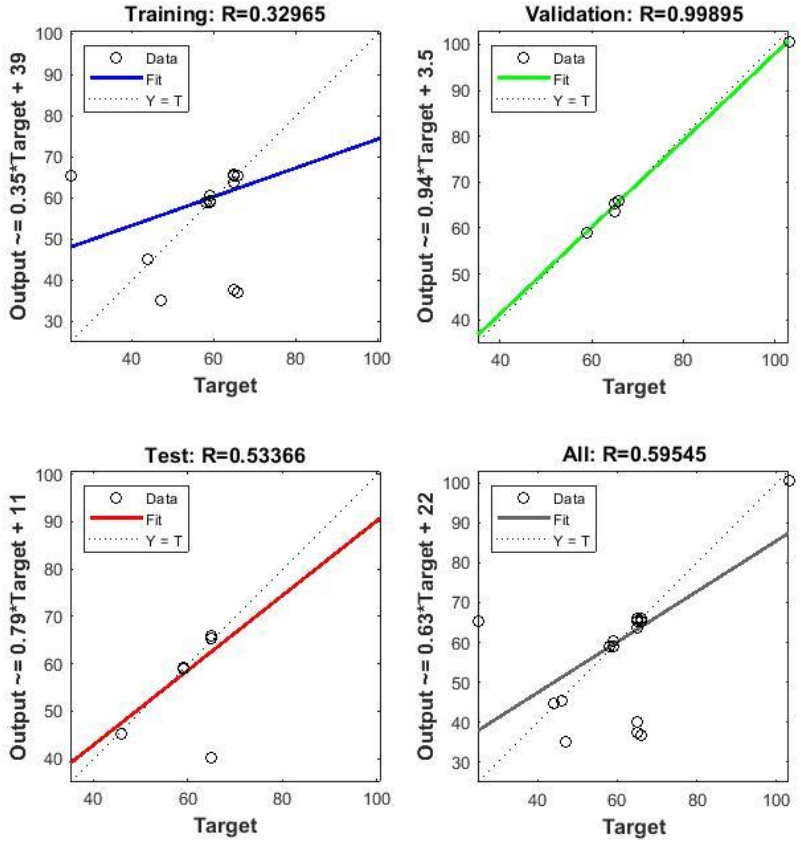

Figura A-6: Regressões Lineares do treinamento, validação saída e de teste, topologia 5,4,3,2, experimento cruz. Fonte: Próprio autor.

Experimento Robô físico parede topologia 5,4,3,2:

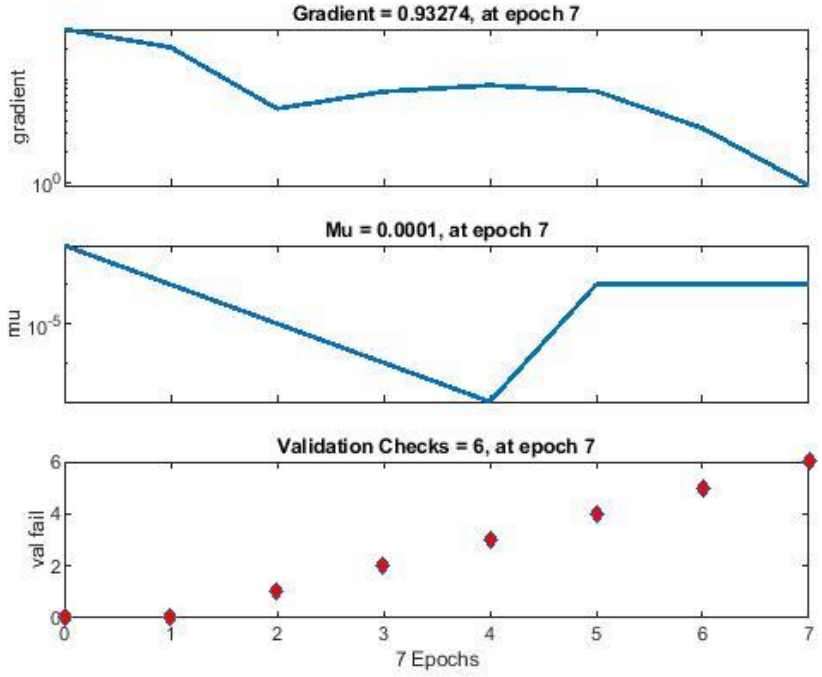

Figura A-7:Desempenho da Rede Neural robô físico topologia 5,4,3,2 experimento parede. Fonte: Próprio autor.

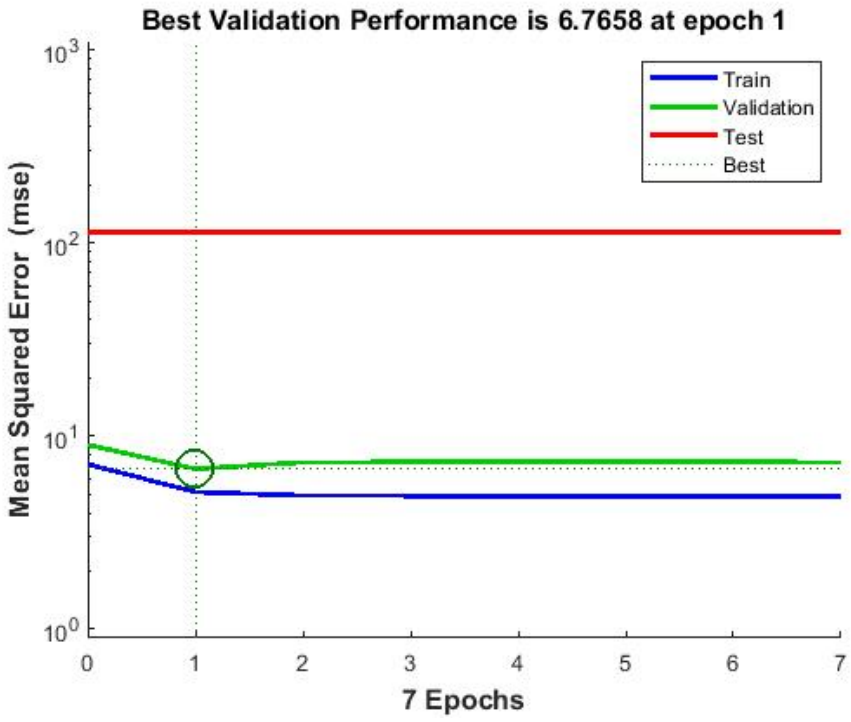

Figura A-8: Resultado do experimento parede : Erro quadrático médio topologia 5,4,3,2 . Fonte: Próprio autor.

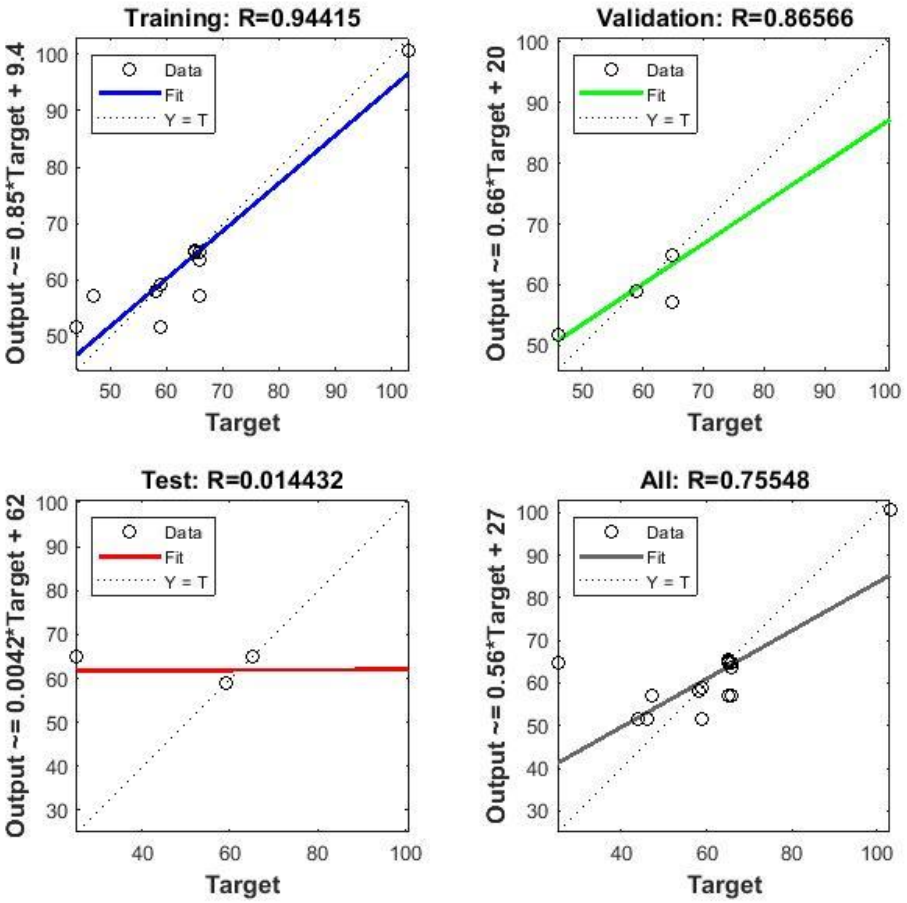

Figura A-9:Regressões Lineares do treinamento, validação saída e de teste, topologia 5,4,3,2, experimento parede.
Fonte: Próprio autor.

APÊNDICE B

Experimento **topologia de rede neural 5,5,2,** gradiente e regressões lineares; validação, treinamento, teste e saída rede dos experimentos:

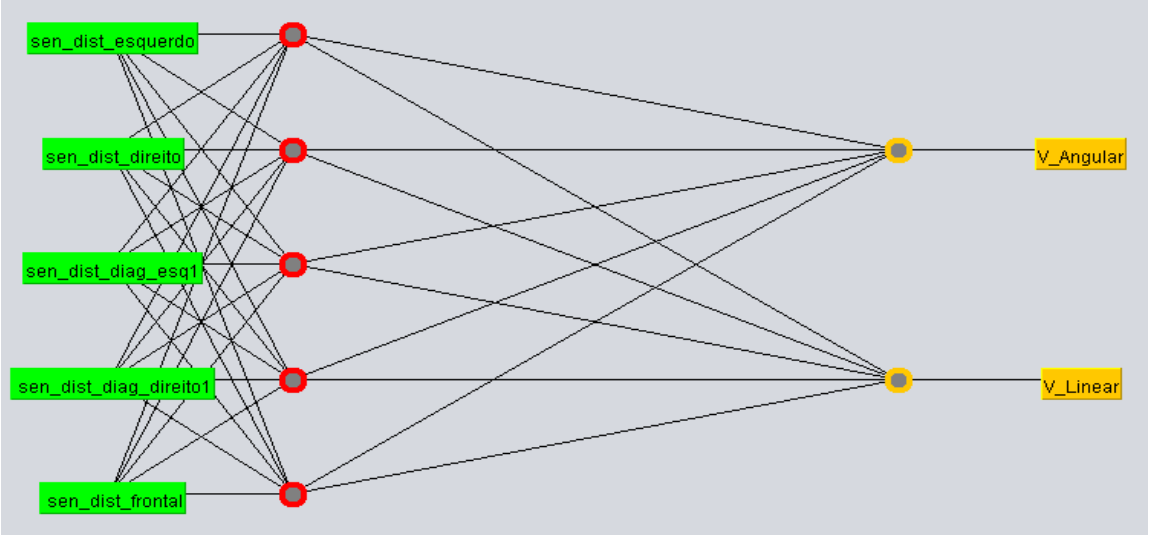

Figura A-10: Topologia rede neural 5,4,3,2 simulado no Weka. Fonte: Próprio autor.

Experimento Caixa topologia 5,5,2:

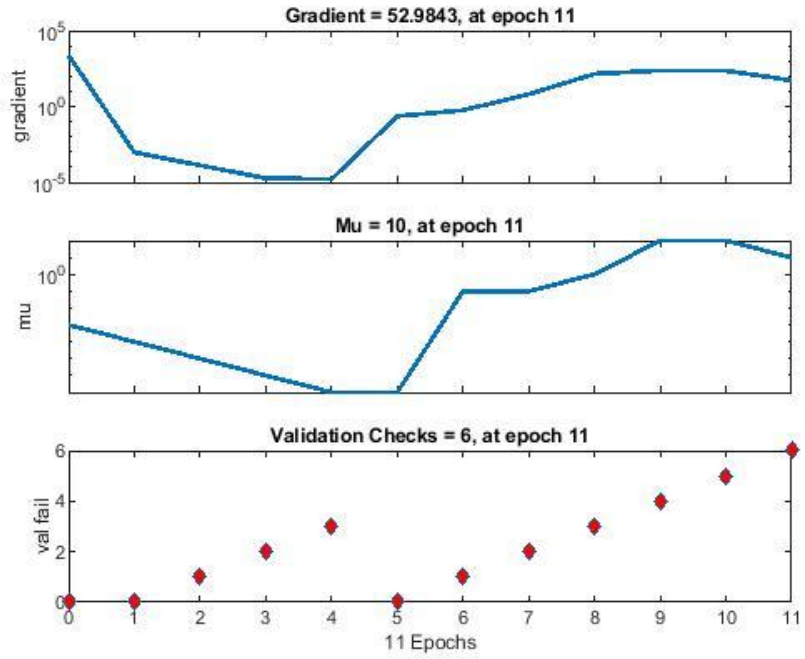

Figura A-11: Desempenho da Rede Neural robô físico topologia 5,5,2 experimento caixa Fonte: Próprio autor.

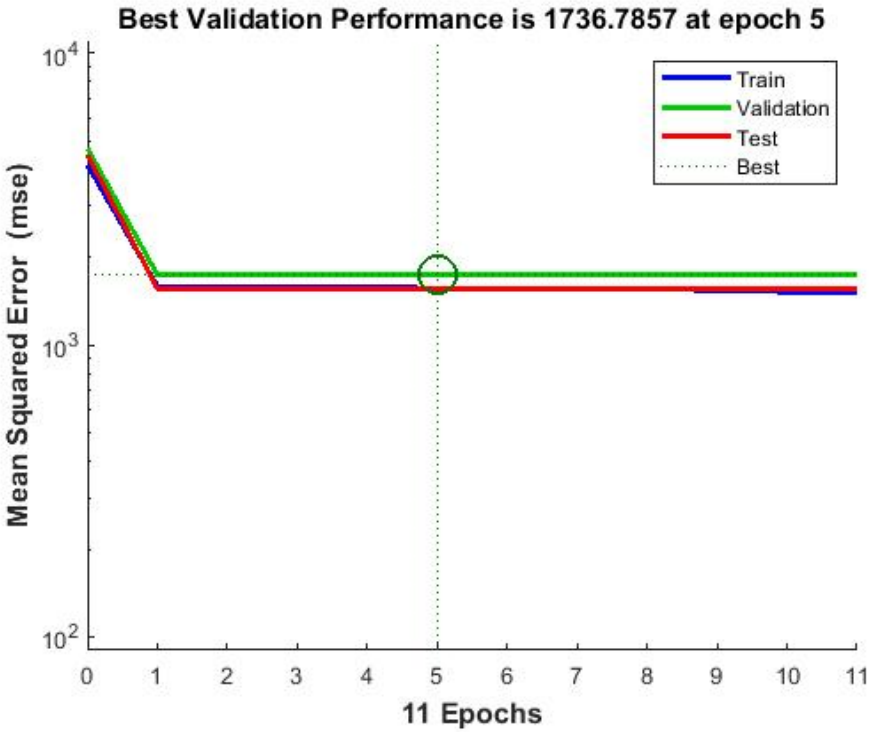

Figura A-12: Resultado do experimento caixa : Erro quadrático médio topologia 5,5,2 . Fonte: Próprio autor.

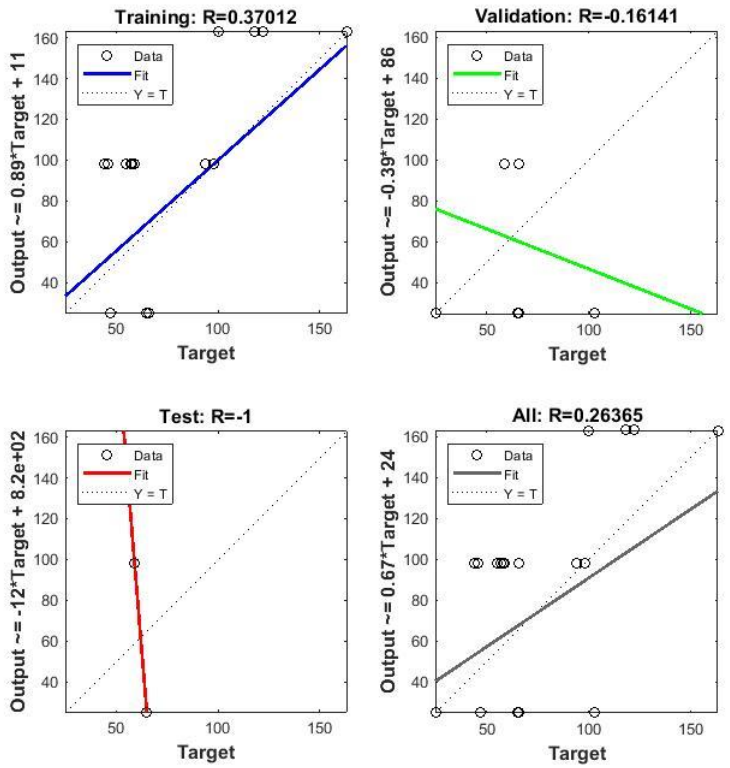

Figura A-13: Regressões Lineares do treinamento, validação saída e de teste, topologia 5,5,2, experimento caixa. Fonte: Próprio autor.

Experimento Cruz topologia 5,5,2:

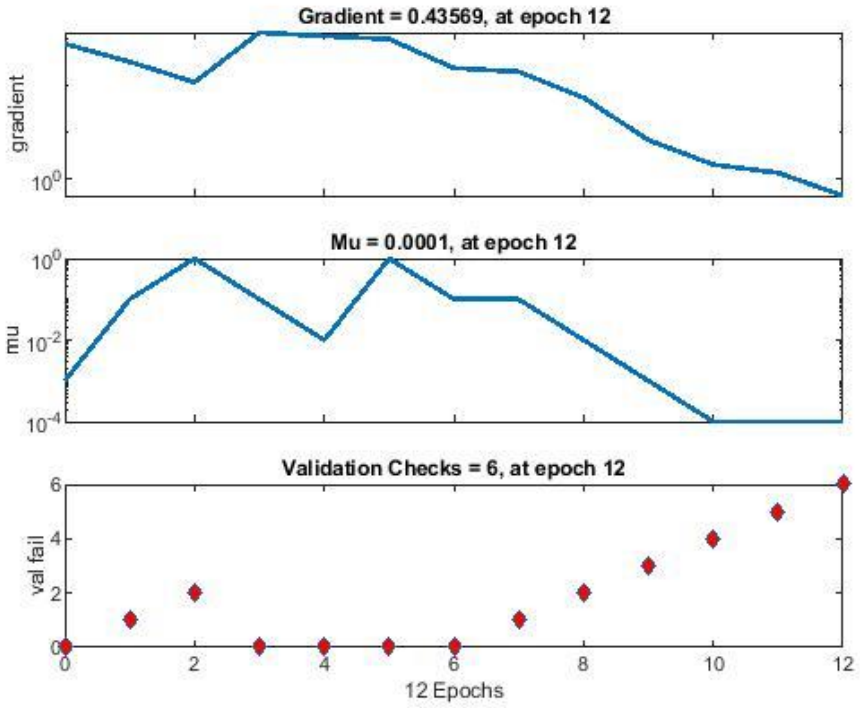

Figura A-14:Desempenho da Rede Neural robô físico topologia 5,5,2 experimento cruz Fonte: Próprio autor.

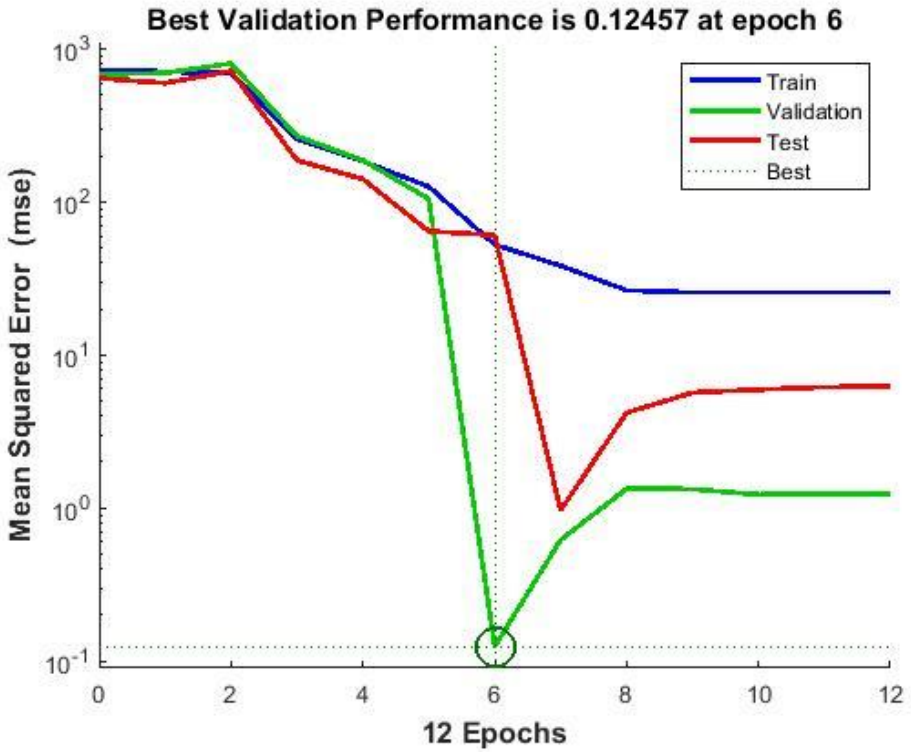

Figura A-15: Resultado do experimento cruz: Erro quadrático médio topologia 5,5,2 . Fonte: Próprio autor.

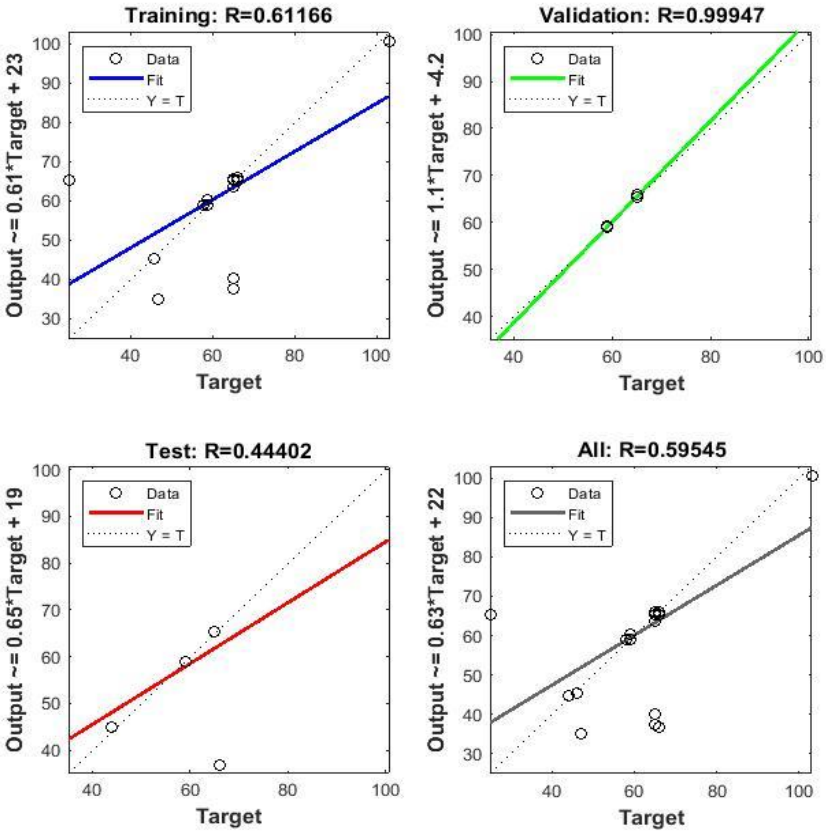

Figura A-16:Regressões Lineares do treinamento, validação saída e de teste, topologia , experimento. Fonte: Próprio autor.

Experimento Parede topologia 5,5,2:

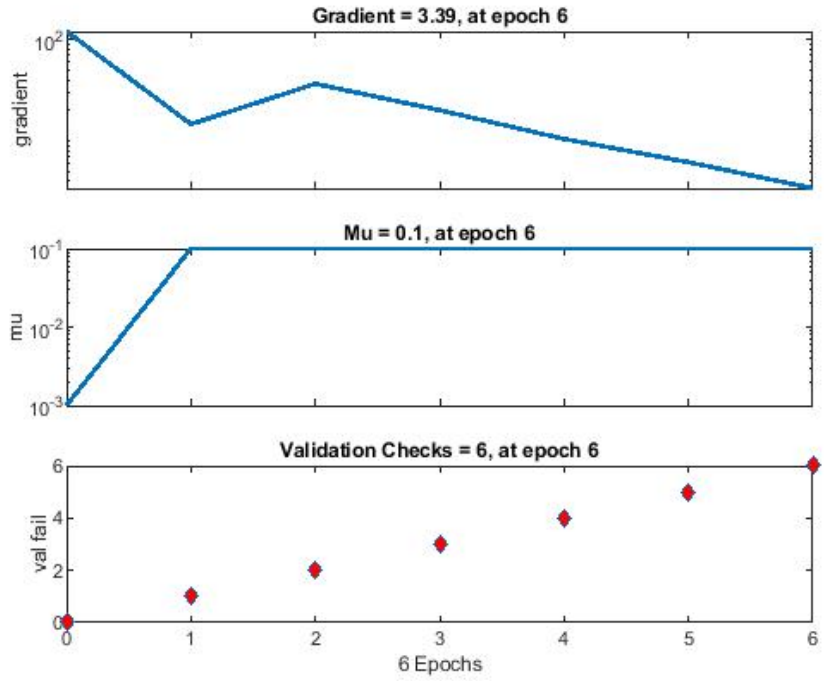

Figura A-17:Desempenho da Rede Neural robô físico topologia 5,5,2 experimento parede Fonte: Próprio autor.

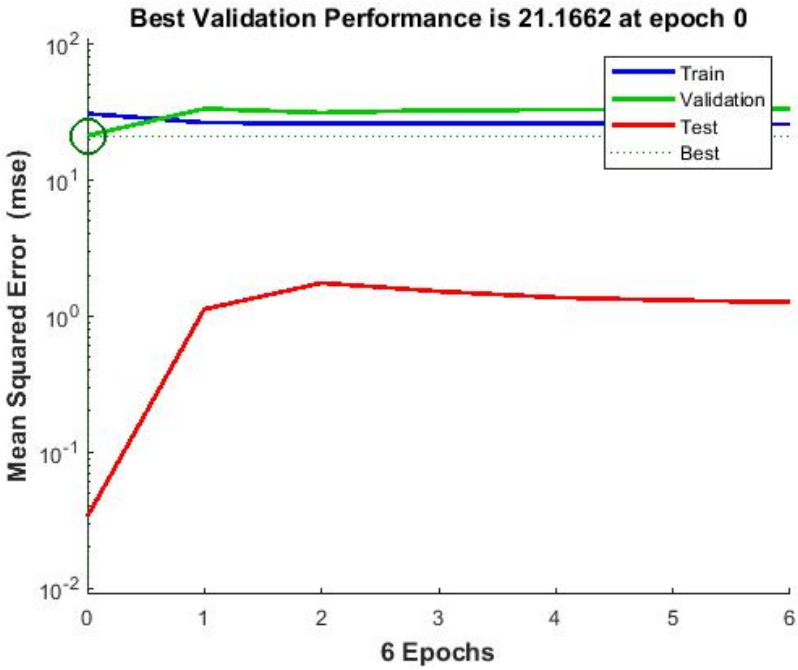

Figura A-18: Resultado do experimento parede : Erro quadrático médio topologia 5,5,2 . Fonte: Próprio autor.

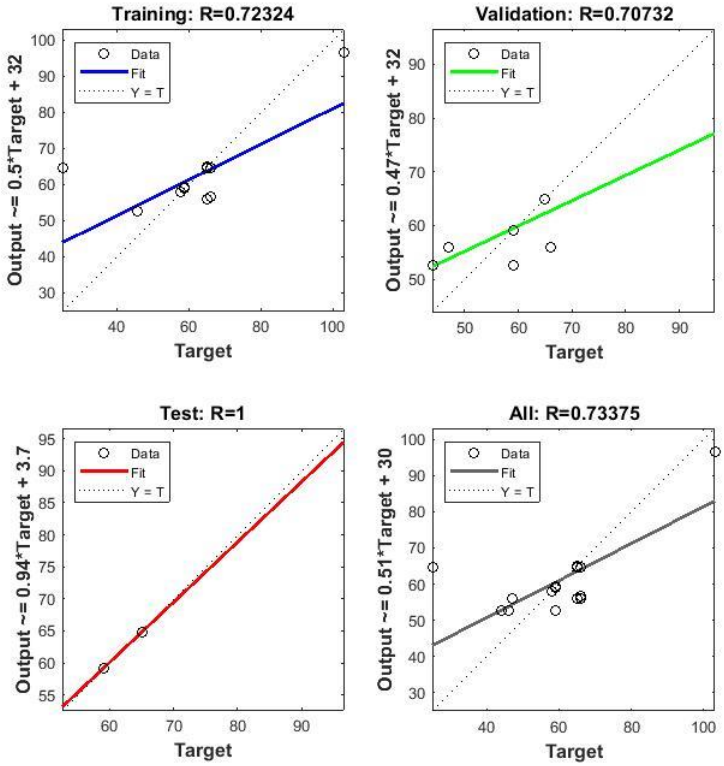

Figura A -19: Regressões Lineares do treinamento, validação saída e de teste, topologia 5,5,2 , experimento parede. Fonte: Próprio autor.

APÊNDICE C

Experimento **topologia de rede neural 5,4,2,** gradiente e regressões lineares; validação, treinamento, teste e saída rede dos experimentos:

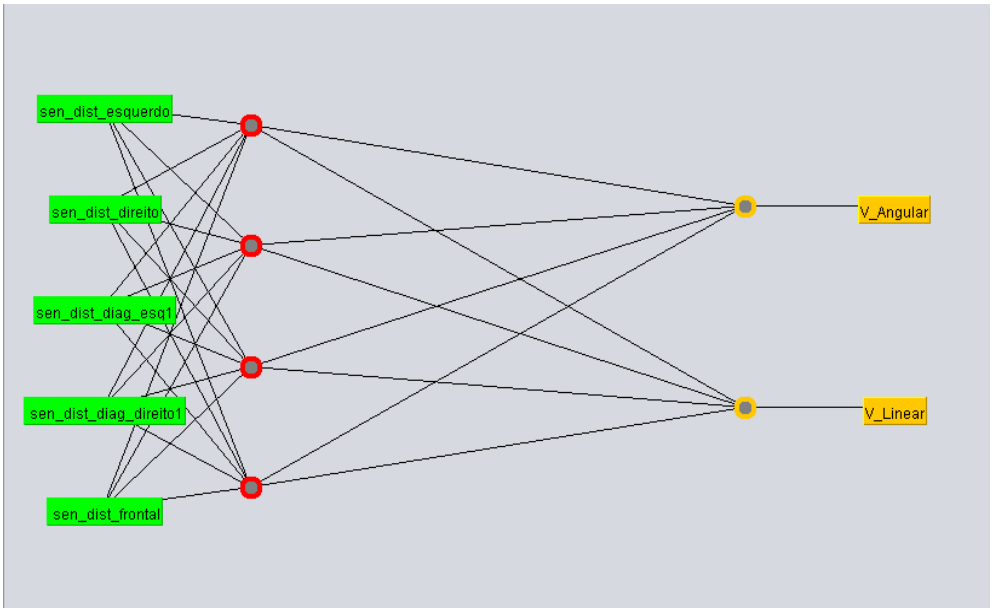

Figura A-20: Topologia rede neural 5,4,2 simulado no Weka. Fonte: Próprio autor.

Experimento Caixa topologia 5,4,2:

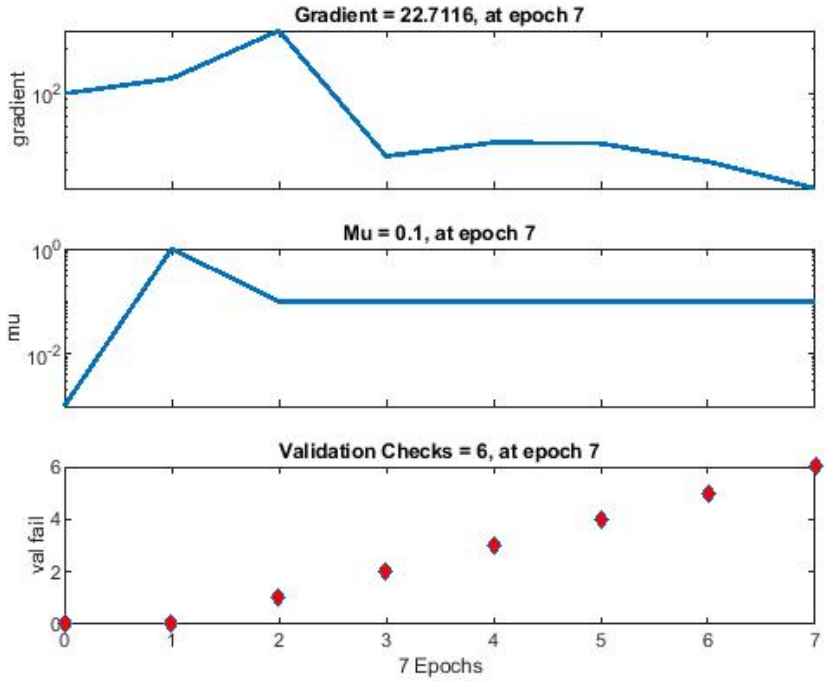

Figura A-21: Desempenho da Rede Neural robô físico topologia 5,4,2 experimento caixa Fonte: Próprio autor.

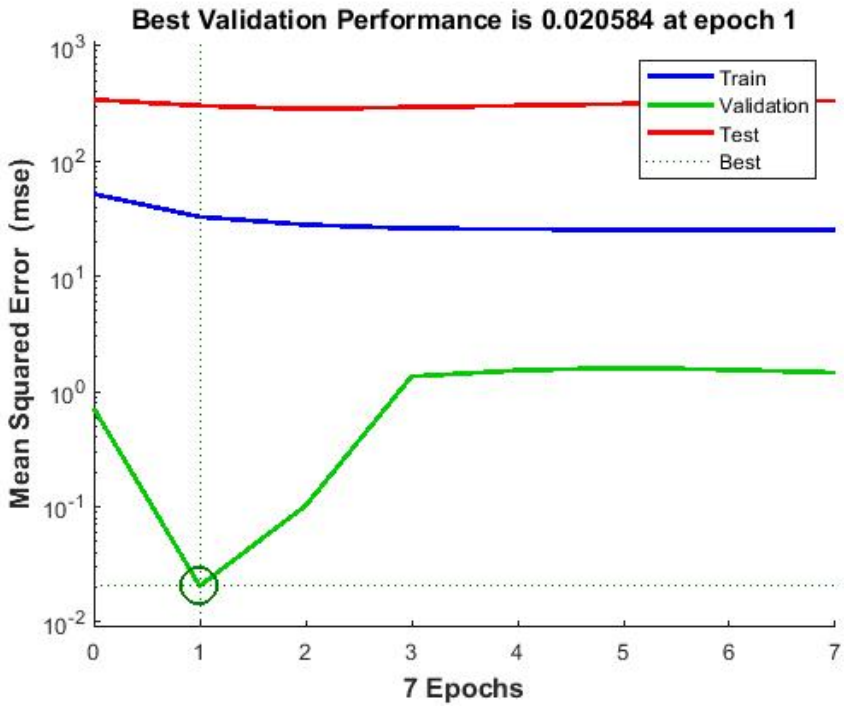

Figura A-22: Resultado do experimento caixa: Erro quadrático médio topologia 5,4,2. Fonte: Próprio autor.

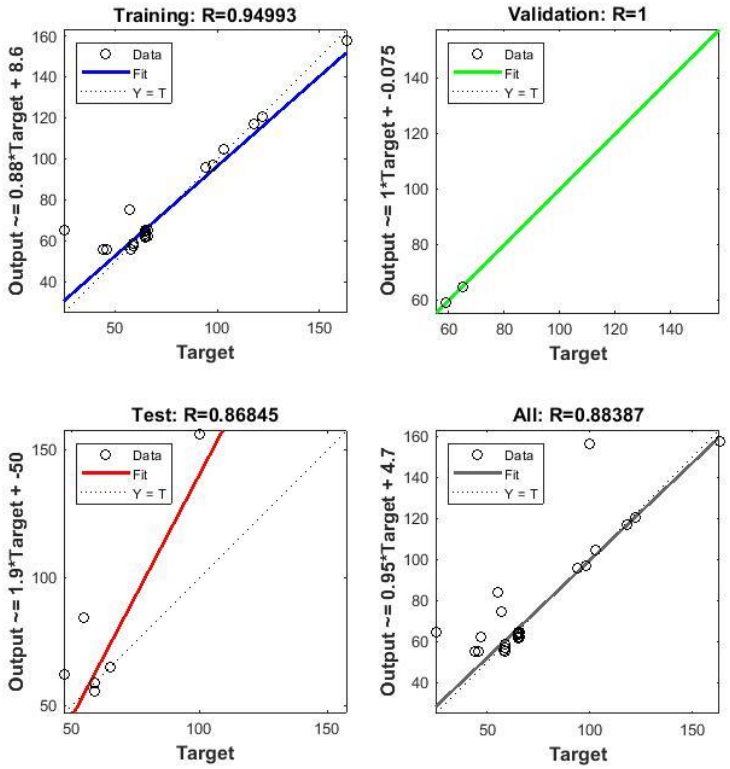

Figura A-23: Regressões Lineares do treinamento, validação saída e de teste, topologia 5,5,2 , experimento parede. Fonte: Próprio autor.

Experimento Cruz topologia 5,4,2:

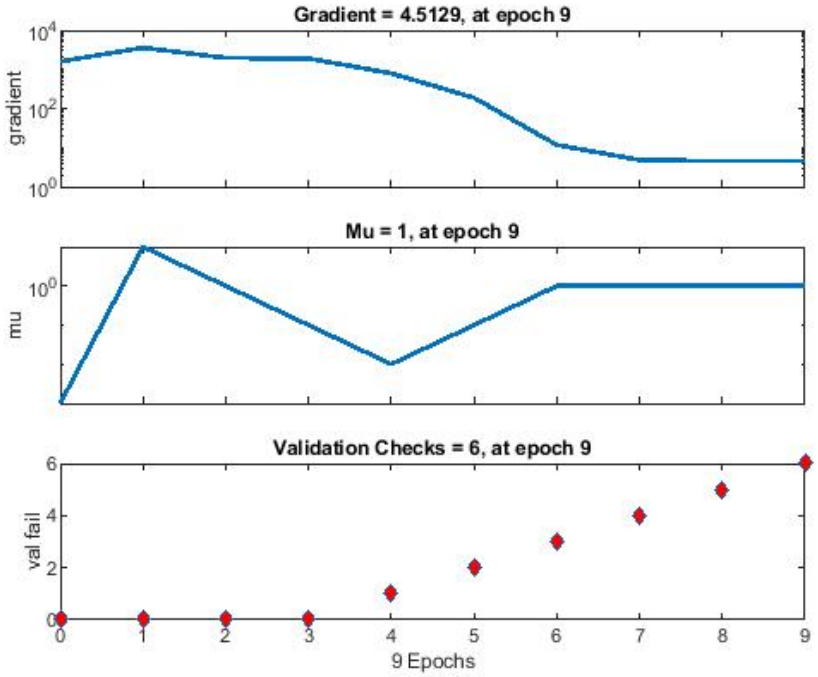

Figura A-24: Desempenho da Rede Neural robô físico topologia 5,4,2 experimento cruz. Fonte: Próprio autor.

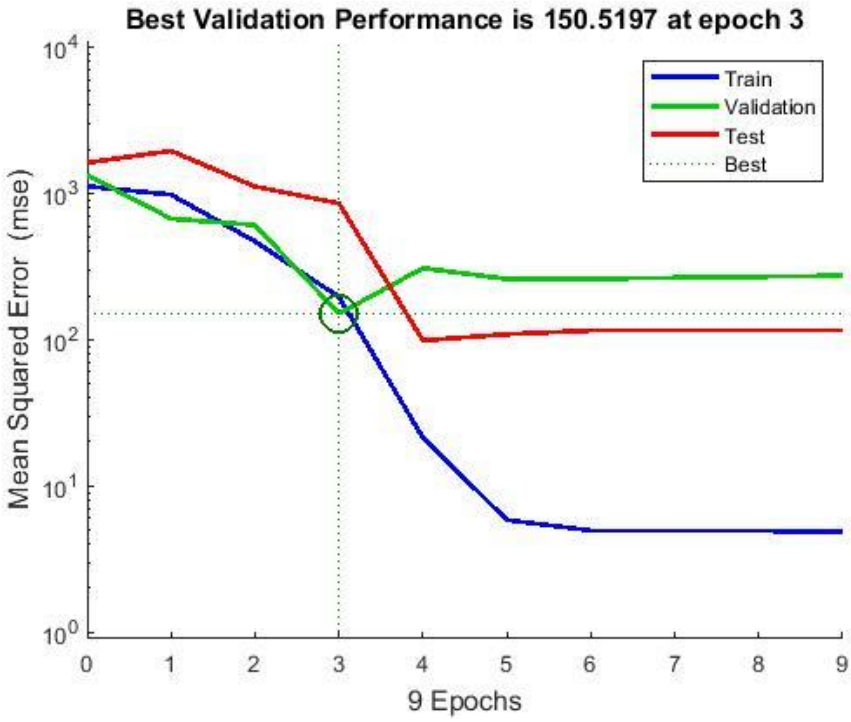

Figura A-25: Resultado do experimento cruz : Erro quadrático médio topologia 5,5,2 . Fonte: Próprio autor.

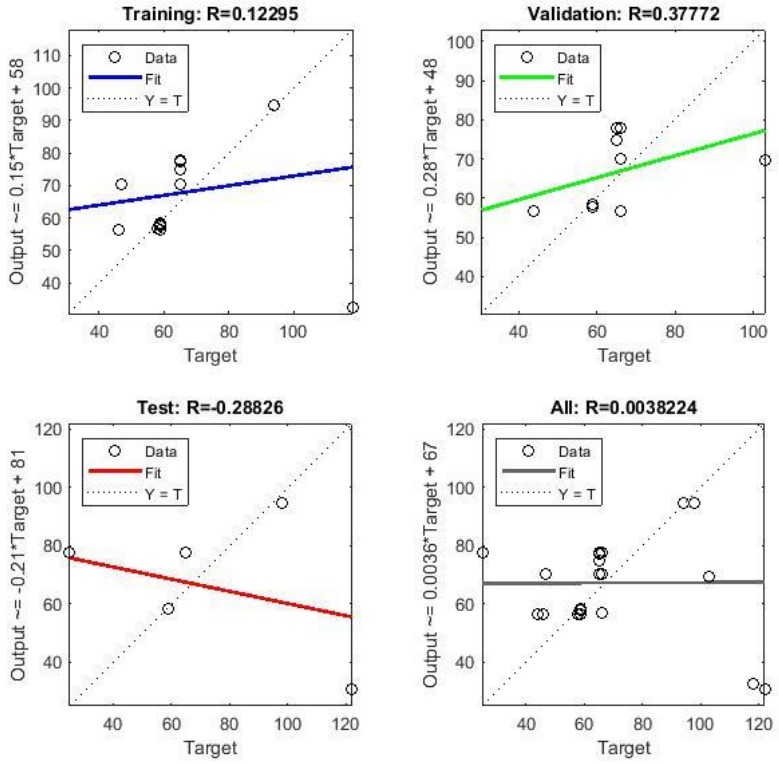

Figura A-26: Regressões Lineares do treinamento, validação saída e de teste, topologia 5,5,2 , experimento cruz. Fonte: Próprio autor

Experimento Parede topologia 5,4,2:

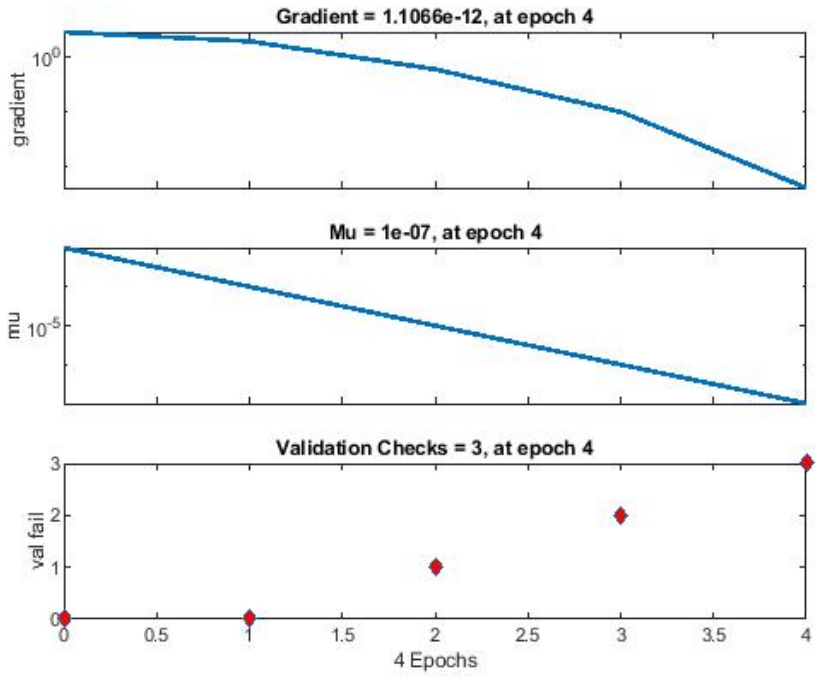

Figura A-27: Desempenho da Rede Neural robô físico topologia 5,4,2 experimento parede Fonte: Próprio autor.

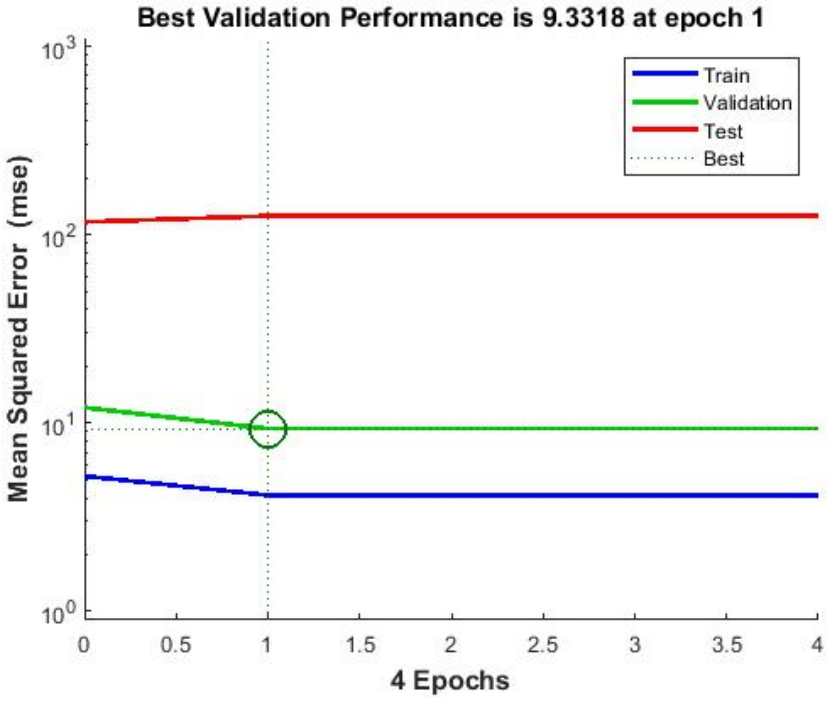

Figura A-28: Resultado do experimento parede : Erro quadrático médio topologia 5,4,2 . Fonte: Próprio autor.

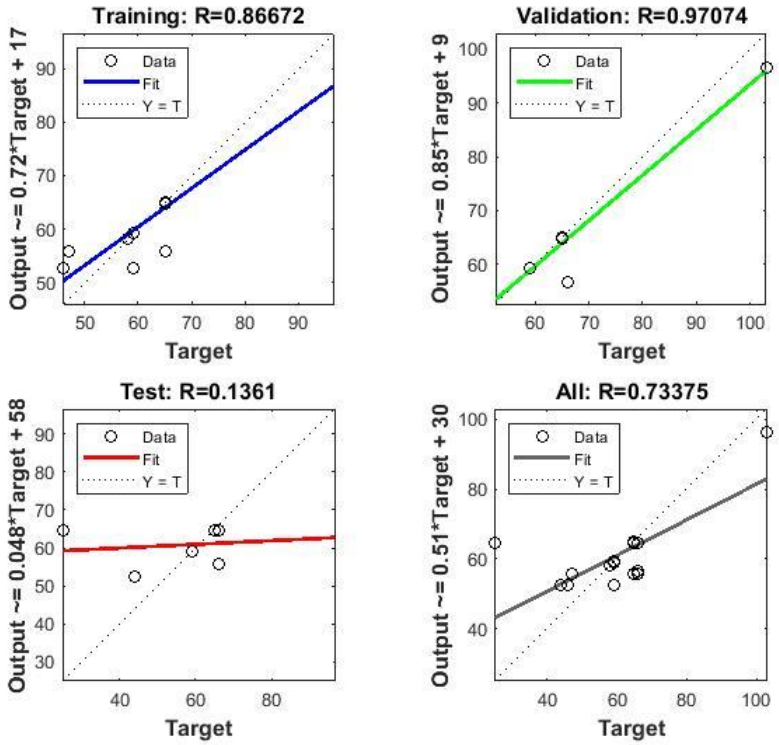

Figura A-29: Regressões Lineares do treinamento, validação saída e de teste, topologia 5,4,2, experimento parede. Fonte: Próprio autor.