UNIVERSIDADE SANTA CECÍLIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA MECÂNICA
MESTRADO EM ENGENHARIA MECÂNICA

SABRINA DE CÁSSIA MARTINEZ

NEUROCONTROLADOR DE VELOCIDADE MICROPROCESSADO

SANTOS/ SP
2015

SABRINA DE CÁSSIA MARTINEZ

NEUROCONTROLADOR DE VELOCIDADE MICROPROCESSADO

Dissertação de Mestrado apresentada à Universidade Santa Cecília como parte dos requisitos para obtenção de titulo de mestre no Programa de Pós-Graduação em Engenharia Mecânica, sob orientação do Prof. Me. Luís Fernando P. Ferrara e coorientação do Prof. Dr. Mauricio Conceição Mario.

SANTOS/ SP

2015

Martinez, Sabrina de Cássia.
Neurocontrolador de Velocidade Microprocessado / Sabrina de Cássia Martinez. -- 2015.
89 p.

Orientador: Prof. Me. Luís Fernando Pompeo Ferrara.
Coorientador: Prof Dr. Mauricio Conceição Mario

Dissertação (Mestrado) -- Universidade Santa Cecília, Programa de Pós-Graduação em Engenharia Mecânica, Santos, SP, 2015.

1. Neurocontrolador. 2. Redes Neurais. 3. *Multilayer* 4. *Backpropagation*. 5. Classificação. I. Ferrara, Luís Fernando Pompeo, II. Mario, Mauricio Conceição. Título: Neurocontrolador de Velocidade Microcontrolado

Elaborada pelo SIBi – Sistema Integrado de Bibliotecas - Unisanta

Dedico este trabalho ao meu marido e amigo, João Marcelo, pelo incentivo, confiança e paciência durante a realização do mesmo.

A diferença entre um homem comum e um guerreiro é que o guerreiro toma tudo como um desafio, e o homem comum toma tudo como uma bênção ou uma maldição.

(Carlos Castañeda)

RESUMO

Desde muito tempo pesquisadores desenvolvem modelos matemáticos que simulam o funcionamento dos neurônios. Tudo começou com o estudo e a simulação de um único neurônio artificial, que acabou por evoluir para as redes neurais artificiais, as quais simulam vários neurônios conectados. Conforme os estudos avançaram, publicações e aplicações surgiram em diversas áreas do conhecimento, como na computacional, na automação com os sistemas inteligentes, na robótica, na biologia, entre outras. Assim esse trabalho apresenta como foco principal, a aplicação de um sistema inteligente na área de controle de processos com a implementação de um neurocontrolador de velocidade microprocessado. O desenvolvimento dessa aplicação se deu através da construção do conhecimento realizado em pesquisas, tais como: a evolução das redes neurais artificias, os tipos de arquiteturas existentes, os métodos de treinamentos e modelos matemáticos. O neurocontrolador trata-se de um dispositivo microcontrolado que foi programado com redes neurais artificiais, que representam o controle do sistema. Para um estudo de viabilidade, inicialmente foram desenvolvidas duas redes neurais, uma de classificação e outra de atuação no processo físico, ambas programadas em linguagem C#. As redes utilizadas foram do tipo *Multlayer Perceptron,* treinadas com o algoritmo *Backpropagation*. Para a análise da aderência das redes utilizaram-se dados gerados a partir de funções matemáticas, as quais simularam cargas com características semelhantes às de um motor alimentando duas cargas diferentes. Após a validação do método proposto, foi adicionada mais uma carga para ser classificada, totalizando assim três curvas para serem aprendidas pelas redes neurais. Pode-se afirmar que os resultados obtidos com a implementação das redes neurais no microcontrolador foram considerados satisfatórios, principalmente quando destacado a eficiência das redes em generalizar o problema proposto. Portanto, a pesquisa não só contribui como literatura na área de automação e controle de processos como amplia o campo de utilização de lógicas não convencionais em diferentes tipos de controle.

Palavras-Chave: Neurocontrolador. Redes Neurais. Multicamadas. Retro propagação.

ABSTRACT

For a long time researchers have being developed mathematical models that simulate the functioning of neurons. All started with the study and the simulation of a single artificial neuron, which eventually evolved into artificial neural networks, which simulates various connected neurons. As studies advanced, publications and applications emerged in several areas of knowledge, as computational area, automation with intelligent systems, robotics, biology, among others. Thus, this work presents, as its main focus, the implementation of an intelligent system in the processes control area with the implementation of a microprocessed speed neurocontroller. The development of this application was made through the construction of knowledge achieved through researches, as the evolution of artificial neural networks, existing architectures types, training methods and mathematical models. The neurocontroller is a microcontrolled device that was programmed with neural networks which represents the control of the system. For a study of feasibility, initially were developed two neural networks, one for classification and another for action in the physical process, being both of them programmed in C # language. The utilized Networks were Multilayer Perceptron type, trained with the Backpropagation algorithm. For adherence analyses of the networks, data generated from mathematical functions were used, which simulated loads with characteristics similar to those of a feeding two different engine loads. After validation of the proposed method, it was added another load to be classified, totaling three curves to be learned by neural networks. It can be said that the results obtained with the implementation of the neural networks on the microcontroller was considered satisfactory, especially when highlighted the efficiency of networks to generalize the proposed problem. Therefore, the research not only contributes as literature on the field of automation and process control as extends the application range of unconventional logic on different types of control.

Keywords: Neurocontroller. Neural Network. Multilayer. Backpropagation.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

P	Ação de controle proporcional
I	Ação de controle integral
D	Ação de controle derivativa
PID	Controlador Proporcional-Integral-Derivativo
MLP	*Multilayer Perceptron* (*Perceptron* Multicamadas)
CC	Corrente contínua
RPM	Rotações por minuto
V	Tensão elétrica
V_d	Valor desejado
V_s	Valor de saída

LISTA DE SÍMBOLOS

ϕ	Função de Ativação
ϕ'	Derivada da Função de Ativação
δ	Gradiente Local
η	Taxa de Aprendizado

SUMÁRIO

Capítulo 1 – INTRODUÇÃO

A simulação do funcionamento do cérebro, principalmente o processo de aprendizagem por experiência, tornou-se um amplo campo de pesquisa. Muitos estudos sobre neurônios artificiais foram desenvolvidos, como por exemplo, pode-se citar os sistemas computacionais inteligentes que trabalham com o conceito de aprendizagem semelhante ao do cérebro humano. Diversos sistemas “inteligentes” podem ser criados, tais como, sistemas especialistas, sistemas baseados em redes neurais artificiais, em algoritmos genéticos, na lógica *fuzzy*, entre outros (CAMPOS, 2004 & TONSIG, 2009). Esses sistemas são capazes de realizar tarefas como a otimização dos controles em diversos processos, a classificação e reconhecimento de padrões e a análise de sinais e imagens (CAMPOS & SAITO, 2004; TONSIG, 2009).

Na área de controle, a teoria mais difundida refere-se aos sistemas de controle moderno. Uma das fundamentações do controle moderno tem como a base, para os ajustes, o modelo matemático do processo. O modelo extrai as informações sobre o comportamento do fenômeno a ser controlado, porém nem sempre esse detalhamento matemático é praticável, principalmente quando o processo apresenta características complexas. Alguns fatores como a precisão de atuação do controle e a quantidade de parâmetros que se alteram em função da posição e tempo tornam a modelagem ainda mais difícil. Assim os sistemas “inteligentes”, como os baseados em redes neurais, tornam-se mais interessantes pelo modo como o problema é representado (REKIK, L. et al., 2010).

O método de aprendizagem das redes neurais artificiais elimina todo o processo de modelagem matemática do fenômeno estudado, pois extrai as características do problema e generaliza a informação aprendida. A generalização é a capacidade que a rede tem de associar as características de um problema e representá-los a partir de um conjunto de informações reduzidas. Esse conceito vai muito além do mapeamento das entradas e saídas, pois quando apresentamos dados não conhecidos à rede, a mesma é capaz de generalizar fornecendo respostas coerentes para o problema (FERRARA, 2005).

1.1 Objetivos do Trabalho

O objetivo principal deste trabalho consiste na demonstração da aplicação de um neurocontrolador, implementado em um microcontrolador, em Sistemas de Automação e Controle. Utiliza-se como rede neural artificial a *Multilayer Perceptron (MLP),* treinada através do algoritmo *Backpropagation*.

Como objetivo secundário pode-se destacar:

- A revisão bibliográfica dos sistemas de controle moderno.
- O estudo das redes neurais do tipo MLP treinada através do algoritmo *Backpropagation*.
- Desenvolvimento do algoritmo de treinamento das redes neurais e aplicação da mesma na classificação dos padrões e atuação no sistema.
- Verificação da viabilidade da utilização das redes neurais artificiais na automação para controle da velocidade.
- Desenvolvimento do algoritmo para implementação no microcontrolador.

Para atingir os objetivos propostos no trabalho alguns estudos teóricos são fundamentais, como a base conceitual dos sistemas de controle moderno e das redes neurais artificiais. Uma breve abordagem a respeito do histórico das redes neurais artificias, arquiteturas e métodos de aprendizagem são citados no decorrer do trabalho e fazem-se necessários para a compreensão do modelo proposto na metodologia.

É importante ressaltar que, por se tratar de um estudo inicial sobre a viabilidade da construção de um neurocontrolador fundamentado nas redes MLP, este trabalho não tem por objetivo a comparação de desempenho com técnicas já existentes.

Capítulo 2 – CONTROLE MODERNO

Este capítulo inicia-se com a discussão da evolução homem-máquina e a necessidade de aperfeiçoar os processos industriais ao longo da história. Em seguida é apresentado um breve estudo dos sistemas de controle moderno: ação de controle ON/OFF (LIGA/ DESLIGA), ação de controle proporcional (P), ação de controle integral (I), ação de controle derivativa (D) e Controlador proporcional-integral-derivativo (PID).

2.1 Evolução tecnológica homem – máquina

A pré-história é o período em que antecede a invenção da escrita, desse período só existem vestígios materiais, ou seja, somente objetos que comprovam a existência dessa parte da história. Durante esse período, o homem desenvolveu diversas soluções práticas para os problemas encontrados, com isso venceu diversas barreiras impostas pela natureza e prosseguiu com o desenvolvimento da humanidade (PINTO, 2012).

Desde a pré-história o homem procura meios de substituir o trabalho braçal ou torná-lo mais fácil, tanto que esse período é dividido de acordo com os instrumentos de trabalho utilizados naquela época. O Paleolítico ou Idade da Pedra Lascada foi o período onde os instrumentos utilizados eram extremamente rústicos, feitos a partir de pedaços de pedra e ossos. No Mesolítico, o domínio do fogo foi o grande marco e no período Neolítico ou Idade da Pedra Polida houve o desenvolvimento da metalurgia, criando objetos de metais, como machados, lanças e ferramentas (UNESCO, 2010).

Na história um pouco mais recente tem-se o desenvolvimento da mecanização, alguns estudiosos indicam que seu início deu-se a partir da invenção dos moinhos de hidráulicos com a procura do homem em moer grãos, vegetais para extração de sucos e irrigar plantações com menos esforço. Cada moinho era capaz de substituir o trabalho de cerca de 20 homens e com a disseminação dos moinhos pela Europa Ocidental houve um crescimento nunca visto antes na produção de alimentos. A partir então, a automação ganhou destaque na sociedade, uma vez que o sistema de produção agrário e artesanal transformou-se em industrial (GOEKING, 2010 & MUSITANO, 2012). Assim o homem direcionou seu conhecimento para desenvolver outros meios e

mecanismos que exonerassem as atividades braçais, surgindo assim as máquinas a vapor, tais como o martelo a vapor que era utilizado para cravar estacas e forjar metais e as locomotivas a vapor que substituíram os cavalos que antes puxavam os vagões (MÖDERLER , 2012).

Esse processo de transformação homem-máquina foi acompanhado por uma notável evolução tecnológica, como o regulador de velocidade criado por James Watt, que tornou a máquina a vapor mais eficiente. O mecanismo criado por Watt, em 1775, consistia em um regulador centrífugo para efetuar o controle de velocidade da máquina a vapor, criando assim um sistema que unia a hidráulica e pneumática. O sistema foi considerado pelo filósofo alemão Karl Marx o invento mais importante da indústria. Alguns autores consideram esse um dos primeiros sistemas de controle com realimentação (GOEKING, 2010). Na figura 2.1 está demonstrando o esquema de controle centrífugo.

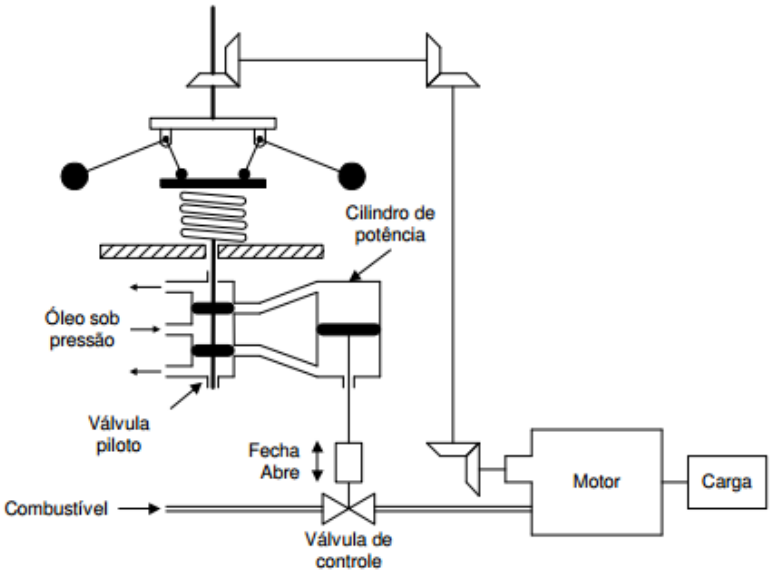

Figura 2.1 - Sistema de controle criado por James Watt.
Adaptado de: SOARES, 2010.

O controlador centrífugo de James Watt controlava o fluxo de combustível no motor de acordo com a diferença entre a velocidade esperada e efetiva do motor. O controlador possui um eixo central com alavancas nas duas pontas e cada uma delas com uma esfera. Conforme a rotação no eixo central diminui, o óleo pressurizado é liberado para a câmara do cilindro, fazendo com que a válvula de controle se abra

injetando mais combustível no motor. E assim, a medida que a rotação aumenta, as esferas afastam-se do eixo, a válvula piloto é fechada e o cilindro fica parado na posição, garantindo a velocidade de rotação correta. Enfim, tem-se o primeiro sistema autorregulado (feedback), onde a saída de um mecanismo é a entrada de dados para o outro, agindo de forma a manter o sistema estável (LAURENTIZ, 2011).

Porém, somente por volta de 1900 apareceram outros reguladores e servomecanismos aplicados a turbinas, máquinas a vapor e entres outros processos. Os pesquisadores como Minorsky, Nyquest, Hazen e entre outros, forneceram valiosas contribuições ao estudo dos sistemas de controle (SOARES, 2010).

- Em 1922, Minorsky sistemas de direção automática para os navios. (SOARES, 2010).
- Nyquist, em 1932 desenvolveu a primeira teoria geral de controle, que determinava a estabilidade de sistemas de malha fechada com base na resposta de malha aberta. (SOARES, 2010).
- Hazen, que em 1934 inseriu o termo servomecanismo para o controle de posição. (SOARES, 2010).
- Em 1942, John Ziegler e Nathaniel Nichols publicam o primeiro método de ajuste para os parâmetros dos controladores PID, os quais são utilizados até hoje na indústria (LOTUFO, 2012).

Na década de 40, diversas técnicas foram desenvolvidas para a área de controle. Os métodos praticamente foram à essência da teoria de controle moderno. Um desses métodos é o de resposta em frequência, chamado diagrama de BODE, que tornou possível projetar sistemas de controle lineares de malha fechada com realimentação (SOARES, 2010). Outro método importante é o do lugar das raízes em projeto de controle, uma técnica gráfica que permite visualizar de que forma os “pólos” de um sistema de malha fechada varia em função do parâmetro K, o “ganho”, que permite ao projetista definir adequadamente a estrutura do controlador apropriado para cada sistema (LOTUFO, 2012).

2.2 Ação de controle ON/ OFF (LIGA/ DESLIGA);

A ação de duas posições é uma das soluções mais utilizadas devido ao baixo custo e a simplicidade, pois compara apenas o sinal de entrada com dois sinais de referência, denominados limite inferior e superior. Esta diferença entre os extremos é chamada de histerese, que normalmente é ajustada para que o valor desejado (*set-point*) fique entre os limites inferior e superior. Esse intervalo diferencial faz com que a saída do controlador mantenha o valor atual até que o sinal de erro tenha se movido ligeiramente além do valor de zero. O diagrama de blocos deste controlador está indicado na figura 2.2 (a) e (b) (PINTO, 2005; OLIVEIRA, 1999).

$$m(t) = M1, \text{ para } e(t) > 0$$
$$m(t) = M2, \text{ para } e(t) < 0$$

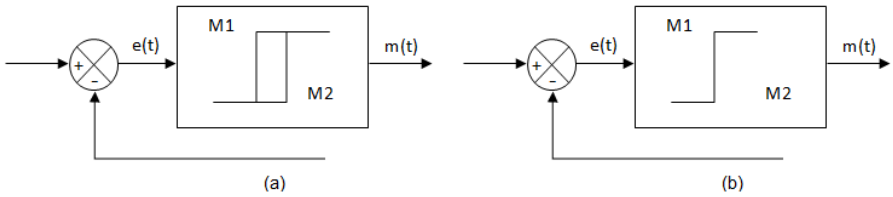

Figura 2.2 - (a) - Diagrama de blocos de um controlador on-off com histerese diferencial. (b) - Diagrama de blocos de um controlador on-off.
Fonte: OGATA, 2003.

Se por um lado este tipo de sistema apresenta uma grande vantagem em relação a custo-benefício, por outro, apresenta certas desvantagens, uma delas é que o valor da grandeza controlada não estabiliza em um ponto e sim oscila em torno do valor desejado provocando um desvio residual denominado erro de *offset*. O outro ponto é o chamado tempo morto que é o período que um processo leva para “sentir” uma variação da entrada na saída ($t - t_0$). Esses erros só são eliminados com ações de controle mais complexas, que serão abordadas a seguir (OLIVEIRA, 1999).

2.3 Ação de controle proporcional (P);

Como foi visto anteriormente, o controlador *ON/OFF* provoca um desvio residual devido as bruscas mudanças de movimento de *ON* para *OFF*. Assim foi desenvolvido um tipo de ação corretiva que é proporcional ao valor do desvio, ou seja, se a entrada for de baixa intensidade a saída será de baixa intensidade, se a entrada for de alta intensidade a resposta será de alta intensidade também. Basicamente esse tipo de ação é, na verdade, um amplificador. Porém o sinal não pode ser amplificado indefinidamente, por isso são criados limites inferiores e superiores que quando são ultrapassados dizemos que o sistema está saturado. Essa região entre os limites é chamada de banda proporcional, que é dada de forma percentual e está relacionada com o ganho K. O diagrama de blocos deste controlador está indicado na figura 2.3. A equação 2.1 determina a ação proporcional (PINTO, 2005; OGATA, 1999).

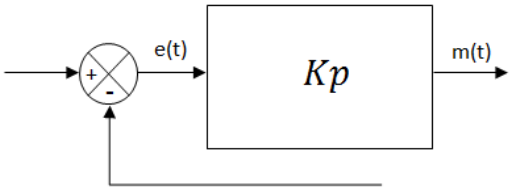

Figura 2.3 - Diagrama de blocos de um controlador proporcional.
Fonte: OGATA, 2003.

$$m(t) = Kp.e(t) \qquad (2.1)$$

Sendo, $\mathrm{m(t)}$ o sinal de saída do controlador, o ganho proporcional Kp e $\mathrm{e(t)}$ o sinal do erro atuante. A função de transferência do controlador proporcional: $\frac{\mathrm{M(s)}}{\mathrm{E(s)}} = \mathrm{Kp}$.

Os sistemas de ação proporcional devem ser utilizados em processos onde não há grandes variações de carga, pois este tipo de ação não consegue manter o equilíbrio gerando um novo erro de offset. Com isso o valor do ganho K tem que ser reajustado pelo operador tornando inviável esse tipo de ação em alguns processos devido ao constante ajuste (OLIVEIRA, 1999).

2.4 Ação de controle integral (I);

A ação de controle integral atua diferentemente da ação proporcional, enquanto a proporcional é ajustada instantaneamente, a integral atua no processo ao longo do tempo, eliminando qualquer desvio que permaneça, ou seja, atua enquanto existir a diferença do valor desejado e valor medido. O diagrama de blocos deste controlador está indicado na figura 2.4. A equação 2.2 determina a ação integral (OGATA, 2003).

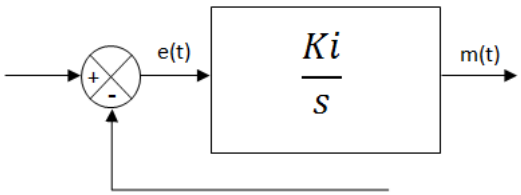

Figura 2.4 - Diagrama de blocos de um controlador integral.
Fonte: OGATA, 2003.

$$m(t) = Ki \int_0^t e(t).dt \qquad (2.2)$$

Sendo, $m(t)$ o sinal de saída do controlador, o ganho integral Ki e $e(t)$ o sinal do erro atuante. A função de transferência do controlador integral: $\frac{M(s)}{E(s)} = \frac{Ki}{s}$

Sendo a ação integral uma função no domínio do tempo, sua resposta é lenta e por isto, grandes desvios em curto período de tempo não são devidamente corrigidos. Este tipo de ação de controle não é utilizado sozinho, comumente ele está sempre associado à ação proporcional, assim tem-se o melhor das duas ações (OLIVEIRA, 1999).

2.5 Ação de controle derivativa (D);

A ação derivativa atua em função da velocidade em que o desvio aparece, ou seja, quanto mais rápida a razão do desvio maior será a correção. Este tipo de ação atua somente em momentos em que há transições bruscas, portanto se houver um erro muito grande, mas variando lentamente o sinal de saída do derivativo será baixa, pois a

ação derivativa não atua no erro e sim deixa o sistema mais rápido. Por isso a ação derivativa normalmente não é utilizada sozinha é sempre utilizada associada a ação proporcional + integral (PINTO, 2005; OGATA, 2003).

Uma das desvantagens da ação derivativa é a amplificação dos sinais de ruído, o qual “engana” o sistema, fazendo-o interpretar que há uma transição brusca no sistema (OLIVEIRA, 1999).

2.6 Controlador proporcional-integral-derivativo (PID)

A ação de controle PID é uma ação que combina os três elementos de controle vistos anteriormente, proporcional + integral + derivativo. Com este tipo de controlador é possível obter qualquer outra combinação (P, PI ou PD), apenas zerando o ganho da ação que não convém. Entretanto, essa é uma das opções com o custo mais elevado e mais difícil de ajustar, pois o sistema precisa ser modelado adequadamente, levando em conta os parâmetros de desempenho, como, tempo de acomodação, erros em regime, etc. A equação 2.3 determina a ação PID, proporcional + integral + derivativa. O diagrama de blocos deste controlador está indicado na figura 2.5 (PINTO, 2005; OGATA, 2003).

$$m(t) = Kp.e(t) + Kp.\frac{1}{Ti}\int_0^t e(t).dt + Kp.T\mathrm{d}\frac{de(t)}{dt} \qquad (2.3)$$

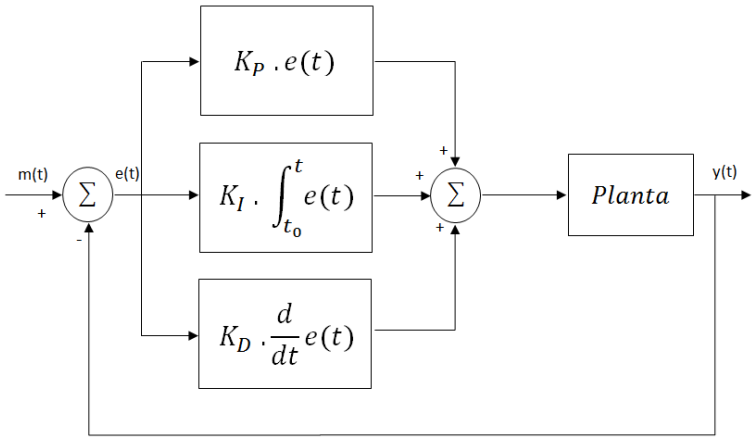

Figura 2.5 - Diagrama de blocos de um controlador PID (proporcional + integral + derivativo). Adaptado de: DEAN, 2003.

Função de transferência do controlador PID: $\frac{\mathrm{M(s)}}{\mathrm{E(s)}} = \mathrm{Kp}\left(1 + \frac{1}{\mathrm{Tis}} + \mathrm{Tds}\right)$

A Implementação dos blocos do PID pode ser feita de forma analógica, utilizando circuitos eletrônicos, amplificadores operacionais, para processar os sinais dos transdutores ou de forma digital, com o uso de microcontroladores e microprocessadores, os quais recebem diretamente os sinais analógicos dos sensores de temperatura, vazão e etc. Nesses casos, os blocos PID são apenas *softwares*, a vantagem desse tipo implementação é a facilidade de se modificar o projeto do controlador quando necessário, uma vez que basta reprogramá-lo (OLIVEIRA, 1999).

Atualmente, os controladores PID são largamente utilizados nas indústrias e são encontrados na maioria das plataformas comerciais. Essa popularidade pode ser atribuída pelo seu algoritmo robusto e de fácil manipulação, além de satisfazer a maioria dos processos industriais. Porém, a teoria clássica trata apenas de processos de entrada-simples e saída-simples, em casos mais complexos de múltiplas-entradas e múltiplas-saídas sua modelagem torna-se quase impossível, devido ao grande número de equações. A modelagem de sistemas complexos depende de diversas informações provenientes do sistema, os quais muitas vezes não apresentam um grau de confiabilidade ou características quantitativas suficientes, fazendo-se necessário o desenvolvimento de novas técnicas para os sistemas de controle, como os estudos das redes neurais artificiais, a lógica *fuzzy* e entre outros. Os estudos relacionados ao controle moderno não são mais ferramentas de aplicação exclusiva da indústria, áreas como biologia, economia, medicina tem-se utilizado desses modelos demonstrando técnicas inovadoras e resultados significativos em suas pesquisas (OLIVEIRA, 1999).

Capítulo 3 – REDES NEURAIS ARTIFICIAIS

O cérebro humano é uma máquina tão complexa que alguns enigmas do seu funcionamento ainda intrigam pesquisadores e cientistas. Um simples ato de ler uma frase e conseguir interpretar o seu significado envolve várias etapas como o processamento da imagem visualizada, a transformação de cada letra em sinais elétricos, a formação das palavras e por fim buscar na memória o significado de uma frase. Todo esse processamento é realizado por uma rede formada por bilhões de células nervosas que se interconectam, conhecidas também como redes neurais. O neurônio é uma célula especializada na transmissão de informações, cada informação é transmitida a outros neurônios através de suas conexões (sinapses), as quais estão relacionadas à capacidade de aprendizagem. Conforme ocorrem interações do ser humano com o ambiente as sinapses são fortalecidas ou enfraquecidas, quando as conexões são fortalecidas ocorre o aprendizado resultado de uma troca constante de sinais elétricos entre os neurônios. Pode-se dizer que quando se pretende melhorar uma habilidade, deve-se praticar inúmeras vezes para evoluir, ou seja, as informações são trocadas diversas vezes entre os neurônios até que sejam criados padrões e quanto mais padrões se criam mais se aprende (OLIVEIRA, 2012).

O neurônio biológico está dividido basicamente em três partes sendo o corpo celular, os dendritos e o axônio. Como pode ser observado na figura 3.1.

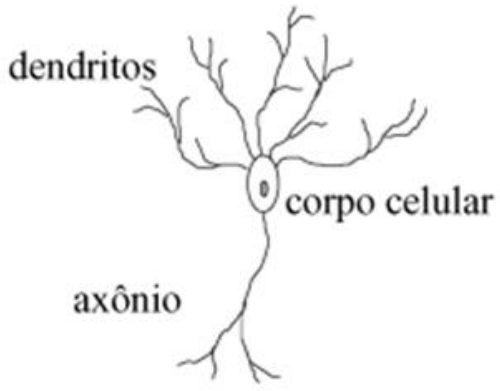

Figura 3.1 - Representação matemática do neurônio biológico.
Adaptado de: CERQUEIRA, E. O. de et al., 2001.

Sem entrar na parte físico-química do processo, o neurônio basicamente recebe informações (sinais elétricos) de outros neurônios através dos seus inúmeros dendritos (entradas), cada uma dessas conexões são chamadas de sinapses e é a partir dela que é determinado o peso sináptico, ou seja, em que grau o neurônio deve considerar os sinais daquela conexão. A quantidade de sinais trocada em uma sinapse é também chamada de intensidade sináptica. O corpo celular é o responsável por modular esses estímulos e se o somatório dos sinais de entrada ultrapassar determinado linear, os sinais são transmitidos aos dendritos de outros neurônios através do axônio. O axônio é a saída que transmite o sinal do corpo celular, onde estão conectados a dendritos de outros neurônios pelas sinapses (MACHADO, 2012; CAMPOS & SAITO, 2004).

3.1 Histórico das redes neurais artificiais

As primeiras analogias entre as células nervosas, os neurônios, e a neuro computação foram publicadas em 1943 através do artigo "A *Logical Calculus of the Ideas Immanenet in Nervous Activity*", pelo neurofisiologista Warren McCulloch e pelo matemático Walter Pitts. O estudo desenvolvido tratava o cérebro como um sistema operacional e tinha a sua representação baseada no que se sabia a respeito dos neurônios biológicos naquela época. Assim toda a pesquisa foi voltada para a descrição de um modelo de neurônio artificial, não abordando técnicas de aprendizado (BRAGA, A. P. et al., 2000).

O conceito de aprendizado só foi motivo de estudo alguns anos depois da publicação de McCulloch e Pitts. As primeiras ideias foram expostas por Donald Hebb em 1949 em seu livro "*The Organization of Behavior*", que diz que o aprendizado ocorre com o fortalecimento das conexões entre os elementos da rede que estivessem ativos simultaneamente. Essa concepção de fortalecimento das conexões em reposta a atividades relacionadas de unidades conectadas, ainda continua sendo a essência das teorias de aprendizado, porém o método de como o ajuste é realizado podem ser realizados de forma diferente (MUELLER, 1996).

O primeiro neurônio acrescido com sinapses ajustáveis, que até então eram mantidas fixas, foi representado por Frank Rosenblatt no final dos anos 50,o modelo

proposto ficou conhecido como *Perceptron* e foi baseado nas linhas de pesquisa de McCulloch e Pitts. A atualização das sinapses na rede neural era realizada com um algoritmo de ajuste automático que comparava a resposta, se a mesma fosse positiva os pesos eram mantidos, caso contrário os pesos eram alterados. Com a técnica de ajuste dos pesos as redes neurais do tipo *Perceptron* foram treinadas para classificar padrões linearmente separáveis, convergindo assim em um número reduzido de passos. Com essa e outras inúmeras contribuições, Rosenblatt é considerado por muitos pesquisadores como fundador da neurocomputação, pois suas pesquisas foram fundamentais para a evolução dos estudos das redes neuras (MUELLER, 1996).

Na década de 60 surgiram diversos estudos relacionados as redes neurais, como a do neurônio artificial denominado ADELINE (*Adaptive Linear Element*) publicado por Windrow e Hoff, que se destacou pela proposta de aprendizado que apresentava, o método ficou conhecido como a regra delta que mais tarde tornou-se a regra delta generalizada. Um artigo de grande relevância nessa mesma década foi o *Perceptron* de Minsky e Papert que demonstrava que o estudo proposto por Resenblatt não era capaz de evoluir, pois matematicamente as redes de uma camada não conseguem resolver problemas que não são linearmente separáveis (BRAGA, A. P. et al., 2000). Essa publicação fez com que poucos pesquisadores continuassem interessados nas redes neurais artificiais, o que reduziu consideravelmente os estudos nos anos seguintes. Porém por volta de 1986 com o aperfeiçoamento do modelo *Perceptron* com o algoritmo *Backpropagation*, por Rumelhart, Hinton e Williams, foi possível realizar o treinamento supervisionado das redes multicamadas, o que resultou em um grande poder de generalização. O método de atualização dos pesos do algoritmo *Backpropagation* se baseia no calculo das derivadas parciais do erro de saída em relação a cada uma das sinapses da rede e na retro propagação desses erros até a camada de entrada. Esse método de cálculo dos pesos despertou novamente grande interesse no desenvolvimento das redes neurais artificiais, surgindo assim novos modelos cognitivos e uma variedade de aplicações como na Biologia, Física, Matemática e entre outras áreas (MUELLER, 1996).

3.2 Arquitetura de uma Rede Neural Artificial

A arquitetura de uma rede neural é um dos aspectos mais relevantes na sua concepção e pode variar de acordo com o tipo de problema em questão. Essas variações incluem o número de camadas, a quantidade de nodos em cada camada, o tipo de conexão entre os nodos e o método de treinamento.

Existem redes de camada única, que apresenta apenas um neurônio entre as entradas e as saídas, como na figura 3.2:

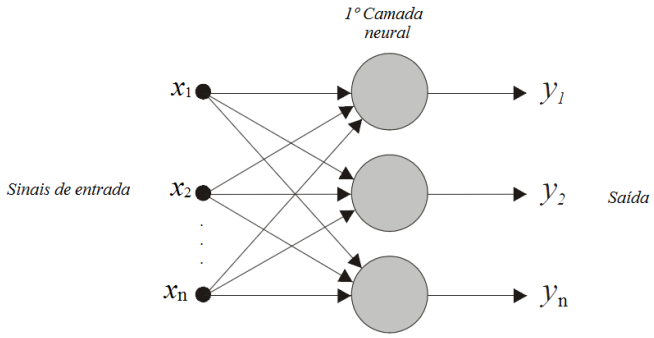

Figura 3.2 - Rede de camada única.
Fonte: adaptado de SILVA, 2003.

E as redes multicamadas que possuem mais de uma camada de neurônio entre as entradas e saídas, demonstrada na figura 3.3:

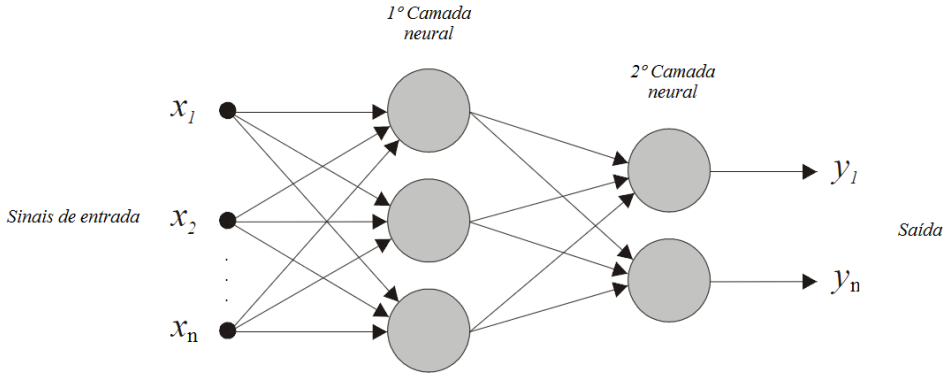

Figura 3.3 - Rede multicamadas.
Fonte: adaptado de SILVA, 2003.

Dependendo do número de camadas adotadas em uma rede neural a rede torna-se mais complexa, podendo interferir diretamente no seu tempo de treinamento e no resultado final (BRAGA, A. P. et al., 2000).

As conexões entre os nodos podem ser do tipo:

- ***Feedforward:*** Nesse tipo de topologia todos os neurônios são conectados aos da camada posterior, porém não há conexões entre neurônios de uma mesma camada e a saída de cada um obedece a um único sentido: da entrada para saída. Uma aplicação muito usual desta topologia é no reconhecimento de padrões através de modelos não lineares.

- ***Feedback:*** Nessa estrutura o fluxo dos sinais entre neurônios ou camadas não obedecem um sentido único. Cada camada pode ter conexões entre os neurônios da mesma camada, da camada posterior e da anterior. O neurônio também pode ser retroalimentado de forma direta ou indiretamente pela sua saída.

De acordo com Braga, et al. (2000), os principais métodos de treinamento são:

- ***Aprendizagem supervisionada:*** nesse método de aprendizagem a rede neural é treinada com a ajuda de um supervisor externo, ou seja, os ajustes de pesos são obtidos da diferença entre as saídas obtidas e as respectivas saídas desejadas, ocorrendo assim o armazenamento de conhecimento. Conforme a diferença é minimizada a cada etapa de treinamento a taxa de erro vai se aproximando de uma possível solução ou uma faixa considerada satisfatória.

- ***Aprendizagem não supervisionada:*** é conhecido também como auto supervisionado, nesse caso somente os padrões de entrada são apresentados a rede, que os classifica formando representações internas para codificar as características.

3.3 Redes Multicamadas *Perceptron*

As redes neurais multicamadas têm como propriedade mais importante a capacidade de aprender em seu ambiente e com isso aprimorar seu desempenho, isso é feito através de um processo iterativo de ajustes dos pesos, o treinamento. Assim o aprendizado só ocorre quando a rede neural alcançar uma resposta generalizada para uma classe de problemas.

O algoritmo de aprendizado é composto por um conjunto de regras bem-definido. Para ajustar os parâmetros de uma rede neural para que a mesma aprenda uma determinada função, existem diversos modelos de algoritmos de aprendizado cada um com suas vantagens e desvantagens diferindo entre si basicamente pelo modo como o ajuste de pesos é modificado (BATISTA, 2012).

Um dos modelos de redes neurais multicamadas mais conhecidos é o *Multilayer Perceptron* (MLP) que seria uma extensão do *Perceptron* de camada única. O que as diferencia são as camadas intermediárias e o aprendizado, nas redes de camada única o aprendizado é realizado pelo cálculo do erro entre a saída da rede e a saída esperada. E nas redes com multicamadas é necessário um algoritmo de aprendizado, pois nas unidades da camada intermediaria não existe uma saída esperada para serem utilizadas de parâmetro, impossibilitando o cálculo do erro nestas unidades (BATISTA, 2012).

Para realizar o treinamento de uma rede *Multilayer Perceptron* utiliza-se o processo de aprendizado supervisionado. Ou seja, a rede é treinada com a ajuda de um supervisor externo que compara a saída obtida com a desejada e a partir dessa diferença o ajuste dos pesos é realizado, conforme essa diferença é minimizada a taxa de erro tende a se aproximar de uma solução. O algoritmo *Backpropagation* é o mais conhecido para o treinamento, este funciona através da relação não linear entre a entrada e a saída, ajustando os valores de peso internamente (FERNANDES, 2004).

3.3.1 Algoritmo *Backpropagation*

O algoritmo *Backpropagation* é um dos métodos mais difundidos para o treinamento de redes do tipo MLP é um importante algoritmo que utiliza o aprendizado do tipo supervisionado. O treinamento funciona através da relação não linear entre a entrada e a saída, ajustando os valores de peso internamente (FERNANDES, 2004).

O treinamento com o algoritmo *Backpropagation* segue uma sequência de duas etapas: uma de propagação (*feedforward)* e outra de retro propagação do erro (*Backpropagation)*. Na etapa *feedforward* um padrão é apresentado à entrada e sua resposta é propagada da entrada para as camadas seguintes, camada por camada, até que a resposta seja produzida pela camada de saída, durante essa etapa os pesos das conexões são mantidos fixos. A etapa de retro propagação do erro utiliza a saída obtida e a saída desejada para atualizar os pesos, de forma a diminuir esta diferença. Os pesos das camadas internas vão sendo modificados conforme o erro é retro propagado até a rede convergir para um estado em que todos os padrões sejam codificados (FERRARA, 2005).

As redes que utilizam o *Backpropagation* para o treinamento, geralmente trabalham com a regra delta generalizada, que é baseada na regra delta desenvolvida por Windrow e Hoff para treinar o neurônio Adaline. A regra delta padrão consiste em um método de gradiente descendente para minimizar o erro quadrado total para a função de ativação linear, como por exemplo, threshold, porém se a superfície de erro for complexa não há garantias de solução ótima, levando o algoritmo a convergir para uma solução estável (mínimos locais) (BRAGA, et al. 2000). Entretanto a regra delta generalizada é utiliza uma função de ativação semi-linear, no caso a sigmoide que é diferencial e não decrescente e é com essa função que o neurônio identifica o nível de atividade de sua entrada e a partir disso é definida sua saída. Não é raro o algoritmo convergir para os mínimos locais e para contornar esse problema, algumas técnicas são utilizadas, como por exemplo, adotar a taxa de treinamento decrescente, o termo momentum, adicionar nós intermediários e ruído aos dados. Essas soluções fazem com que a rede acelere o algoritmo *Backpropagation* reduzindo a incidência dos mínimos locais. (ARAÚJO. et al., 2012).

3.3.1.1 Descrição do método de treinamento

O processo de treinamento de uma rede neural com o algoritmo *Backpropagation* envolve duas fases, a de propagação onde os dados são apresentados a entrada e propagados até a saída. E a de retro propagação do erro, que a partir da saída obtida compara com a saída desejada e enquanto houver diferença os pesos são modificados até a saída generalizar.

Para exemplo do cálculo da propagação do erro e ajustes dos pesos consideramos a figura 3.4:

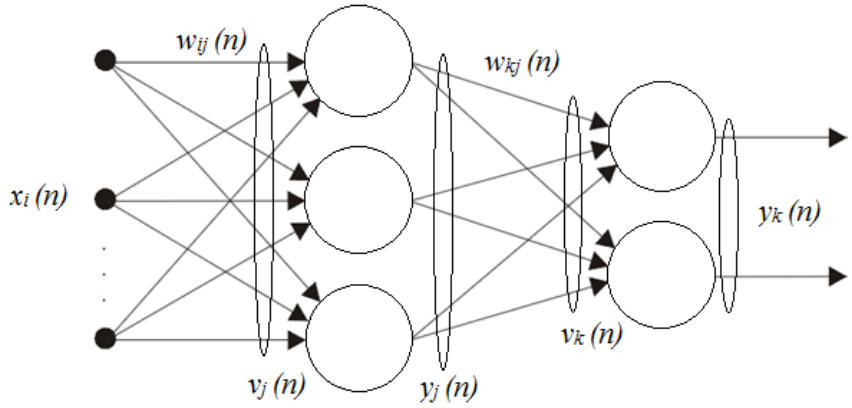

Figura 3.4 - Rede multicamadas com os termos para o treinamento.

Todo o processo de treinamento é baseado em função do sinal de erro calculado, que é o responsável pela atualização dos pesos (SILVA, 2014). Na saída do neurônio k considerando a $k-ésima$ amostra de treinamento o sinal de erro é dado pela diferença entre a saída desejada e a saída da função de ativação do neurônio k:

$$e_k(n) = d_k(n) - y_k(n) \qquad (3.1)$$

O termo e_k corresponde ao erro calculado na saída do neurônio k, a resposta desejada é dada por $d_k(n)$ e a resposta calculada para o neurônio k é $y_k(n)$. O cálculo de $y_k(n)$ é dado por:

$$y_k(n) = \phi(v_k(n)) \qquad (3.2)$$

Onde a função de ativação ϕ associada ao neurônio k é a sigmoide:

$$\frac{1}{1+e^{\alpha v_k(n)}} \qquad (3.3)$$

Nesse caso a suavidade da curva é determinada pelo fator α e a orientação da sigmoide é determinada pela direção do vetor peso w, ou seja, define a posição da função sigmoide com relação ao eixo da ordenada (BRAGA, et al., 2000). O sinal que será aplicado na função de ativação é definido por:

$$v_k(n) = \sum_{j=0}^{m} w_{kj}(n)\, y_j(n) \qquad (3.4)$$

Sendo w_{kj} o peso da sinapse conectada a saída j e a entrada do neurônio k. O termo $y_j(n)$ corresponde a saída do neurônio j, m representa o número total de entradas, excluindo o bias. O processo de treinamento é iniciado com a propagação do erro, no caso apresentam-se estímulos de entrada $x_i(n)$ e nessa fase do cálculo os pesos das sinapses são iniciados com valores aleatórios e permanecem inalterados e em seguida são propagados pelas camadas intermediárias até a camada de saída.(CARRARA, 1997).

$y_j(n) = \phi(v_j(n))$, onde

$y_j(n)$ é a saída do neurônio j, ou seja,

$$y_j(n) = \phi(\sum_{j=0}^{m} w_{ji}(n)\, x_i(n)) \qquad (3.5)$$

Para medirmos o desempenho local se faz necessário à utilização de uma função de custo, o erro quadrático (SILVA, 2014). Com essa função todos os valores dos erros de cada neurônio durante o treinamento são computados por:

$$E(n) = \frac{1}{2}\sum_{j\in C} {e_k}^2 (n) \qquad (3.6)$$

Assim:

$$E(n) = \frac{1}{2}\sum_{j=1}(d_k(n) - y_k(n))^2 \qquad (3.7)$$

A cada ciclo de treinamento a função assume um novo valor, pois os pesos estão sendo atualizados, é dessa forma que se verifica como os erros vão decrescendo durante a fase de treinamento.

3.3.1.1.1 Ajustes dos pesos da camada de saída

Na fase de retro propagação do erro, o ajuste dos pesos e bias é baseado no gradiente, regra Delta, da função do erro quadrático, equação (4.6). O cálculo do gradiente é recursivo para cada neurônio e faz com que os pesos na superfície do erro caminhem na direção na qual deverá diminuir, ou seja, na direção contrária ao gradiente da função custo (CARRARA, 1997).

Para o ajuste dos pesos sinápticos da camada de saída, o algoritmo *Backpropagation* aplica uma correção proporcional à derivada parcial do erro quadrático. A relação $\frac{\partial E(n)}{\partial w_{kj}}$ da equação 4.8, determina a direção de busca no espaço da sinapse w_{kj}.

$$\frac{\partial E}{\partial w_{kj}} = \frac{\partial E}{\partial y_k}\frac{\partial y_k}{\partial v_k}\frac{\partial v_k}{\partial w_k} \qquad (3.8)$$

Logo, diferenciando as equações tem-se:

$$\frac{\partial E}{\partial w_{kj}} = -e_k \phi'(v_k) y_j \qquad (3.9)$$

Assim a correção aplicada à sinapse é definida pela Regra Delta:

$$\Delta w_{kj}(n) = -\eta \frac{\partial E}{\partial w_{kj}} \quad (3.10),$$

onde η é a taxa de aprendizado, substituindo 3.9 em 3.10:

$$\Delta w_{kj}(n) = \eta e_k \phi'(v_k) y_j \quad (3.11)$$

ou,

$$\Delta w_{kj}(n) = \eta \delta_k y_j \quad (3.12),$$

onde o gradiente local é descrito pelo δ_k . Após ter calculado a correção sináptica o valor atualizado do peso é descrito por:

$$w_{kj}(n+1) = w_{kj}(n) + \Delta w_{kj}(n) \quad (3.13)$$

3.3.1.1.2 Ajustes dos pesos da camada escondida

Para o cálculo do ajuste das sinapses $\Delta w_{ji}(n)$ seria necessário o erro da saída $e_j(n)$, porém nesse caso como a camada é escondida então não existe uma resposta desejada para o calculo do erro, recebendo assim uma estimativa do erro calculado na camada de saída (CARRARA, 1997). Para o ajuste do peso $w_{ji}(n)$ em relação à retropropagação do erro tem-se:

$$\frac{\partial E}{\partial w_{ji}} = \frac{\partial E}{\partial y_j} \frac{\partial y_j}{\partial v_j} \frac{\partial v_{kj}}{\partial w_{kj}} \quad (3.14),$$

assim diferenciando as equações tem-se:

$$\frac{\partial y_j}{\partial v_j} = x_i \quad (3.15)$$

$$\frac{\partial v_{kj}}{\partial w_{kj}} = \phi'(v_j) \quad (3.16)$$

O cálculo da derivada parcial $\frac{\partial \mathrm{E}}{\partial \mathrm{y_j}}$ é dado por:

$$\frac{\partial E}{\partial y_j} = \sum_k \frac{\partial E}{\partial v_k} \frac{\partial v_k}{\partial y_j} \qquad (3.17),$$

onde,

$$\frac{\partial v_k}{\partial y_j} = \sum_k \frac{\partial (w_{kj}\, y_j)}{\partial y_j} = w_{kj} \qquad (3.18),$$

assim o cálculo do termo $\frac{\partial E}{\partial v_k}$:

$$\frac{\partial E}{\partial v_k} = \frac{\partial E}{\partial y_k} \frac{\partial y_k}{\partial v_k} = -(d_k - y_k)\, \phi'(v_k) \qquad (3.19),$$

Dessa forma define-se o gradiente local sendo:

$$\delta_k = -(d_k - y_k)\, \phi'(v_k) \qquad (3.20),$$

substituindo as equações 3.18 e 3.20 em 3.17:

$$\frac{\partial E}{\partial y_j} = -\sum_k w_{kj}\, \delta_k \qquad (3.21),$$

logo se as equações 3.15, 3.16 e 3.21 forem substituídas em 3.14, tem-se:

$$\frac{\partial E}{\partial w_{ji}} = -\left(\sum_k w_{kj}\, \delta_k\right) x_i\, \phi'(v_j) \qquad (3.22)$$

O ajuste de pesos w_{ji} deve ser efetuado na direção oposta ao gradiente para minimizar o erro, assim:

$\Delta w_{ji}(n) = -\eta \frac{\partial E}{\partial w_{ji}}$ (3.23), substituindo 3.22 em 3.23:

$$\Delta w_{ji}(n) = \eta\left(\sum_k w_{kj}\ \delta_k\right)x_i \qquad (3.24),$$

então o gradiente local é definido por:

$$\delta_j = \left(\sum_k w_{kj}\ \delta_k\right)\phi'(v_j) \qquad (3.25),$$

assim a correção aplicada à sinapse é dada por:

$$\Delta w_{ji}(n) = \eta\ \delta_j x_i \qquad (3.26)$$

Portanto, o valor atualizado do peso na primeira camada é descrito por:

$$w_{ji}(n+1) = w_{ji}(n) + \Delta w_{ji}(n) \qquad (3.27)$$

Como mencionado anteriormente, os pesos iniciais da rede são inicializados com valores aleatórios e somente depois que os primeiros valores de entrada são propagados e os desvios calculados é que os pesos são atualizados com valores relativos aos dados do problema apresentado.

Capítulo 4 – ESTUDO DA CLASSIFICAÇÃO DE PADRÕES COM REDES NEURAIS *MULTILAYER PERCEPTRON* TREINADA COM O ALGORITMO *BACKPROPAGATION*.

Seguindo os objetivos propostos neste trabalho, o neurocontrolador de velocidade trata-se de um conjunto de redes neurais de multicamadas do tipo *Perceptron*, que serão treinadas com o algoritmo *Backpropagation* para realizar o ajuste inteligente da velocidade. Na fase anterior a da implementação do neurocontrolador no microcontrolador, optou-se por realizar alguns estudos para viabilizar a construção do mesmo, as etapas envolvem o desenvolvimento das redes neurais, o treinamento e os testes. Inicialmente as redes neurais foram desenvolvidas em C# no software Visual Studio, optou-se por esse software devido a sua interface gráfica para auxílio nos testes que foram realizados futuramente. O treinamento foi realizado com valores gerados a partir de funções matemáticas e não os reais, devido ao fato dos mesmos não estarem disponíveis na etapa inicial. O conjunto de valores que foi gerado representa uma curva característica semelhante a do sistema que se pretende controlar, dessa forma planeja-se simular de maneira mais fiel possível as respostas do neuroconrolador.

No modelo proposto o neurocontrolador de velocidade é composto por duas redes, uma de classificação de padrões e outra de atuação no sistema. A rede de classificação tem como finalidade identificar qual o padrão que aquele conjunto de valores, que são apresentados à rede, pertence. Essa característica é o cerne do neurocontrolador, pois será necessário que a rede neural, antes de efetuar o ajuste de velocidade, identifique qual a carga que o motor está submetido naquele instante. Deve-se considerar que o motor pode operar com cargas diferentes, onde cada carga gera um valor de rotação diferente para uma mesma tensão aplicada. Para que a rede de classificação conseguisse simular qual a carga ou padrão que o motor está operando naquele instante, de maneira simples, optou-se por desenvolver um conjunto de valores de treinamento provenientes de duas funções matemáticas, que nesse caso simulam duas cargas distintas. O padrão 1 corresponde a função 4.1 e o padrão 2 a função 4.2.

$$y = 2x + (x^2) \qquad (4.1) \qquad\qquad y = -2x + (x^2) \qquad (4.2)$$

Logo após o treinamento da rede de classificação com os valores gerados pelas funções 4.1 e 4.2, se faz necessário o ajuste da velocidade, que ocorrerá depois que o padrão de referência da planta for identificado, ou seja, padrão 1 ou padrão 2. A figura 4.1 demonstra que é utilizada para o ajuste da velocidade outra rede neural, nesse caso podemos nomear como rede neural de atuação.

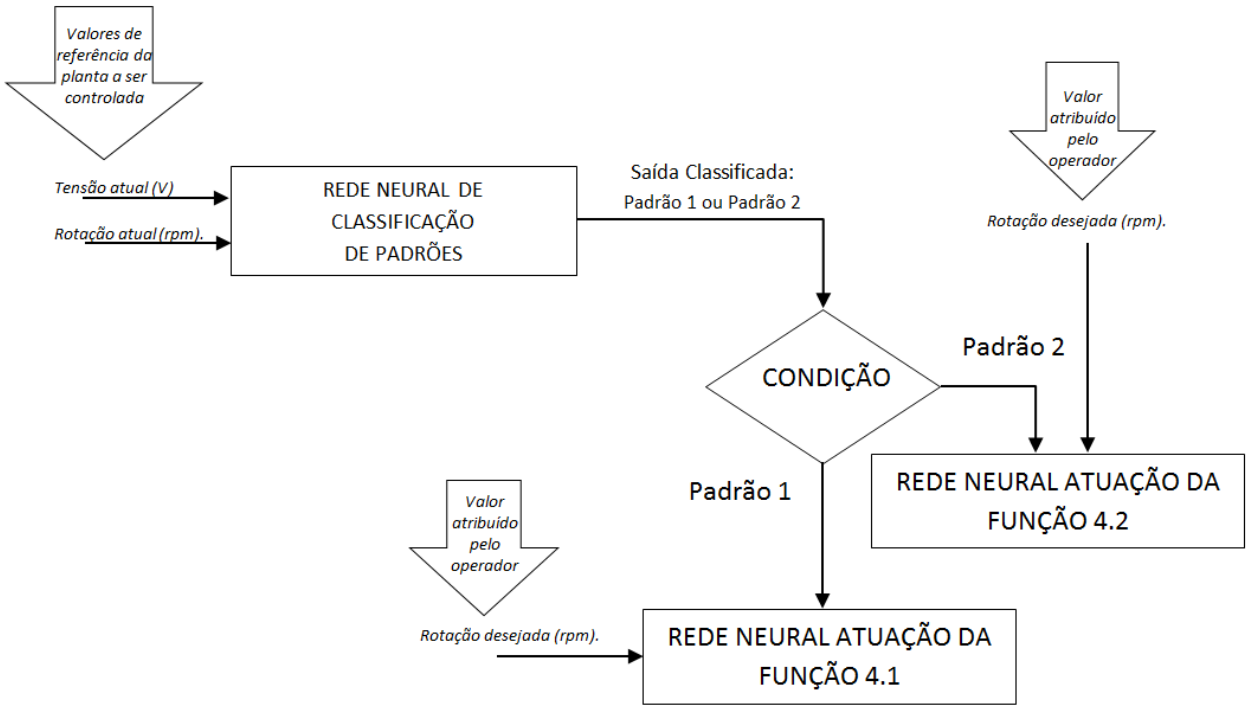

Figura 4.1 – Diagrama de blocos da rede neural de classificação e atuação

O modelo da rede neural de atuação é semelhante ao utilizada para a rede de classificação, mas treinada de forma diferente. A rede de classificação é treinada para identificar o padrão de referência que vem do sistema, ou seja, determinar qual a carga que está agindo naquele instante baseada na tensão e rotação atual do sistema. Logo, a rede de atuação através de seu treinamento deverá aprender a curva característica de cada padrão ou carga individualmente, o que possibilitará realizar efetivamente o ajuste da velocidade atual para desejada uma vez que a rede já conhece as propriedades de cada padrão.

4.1 Treinamento da rede neural para classificação de padrões

Para que a rede neural de classificação consiga distinguir os padrões de entrada, se faz necessário o seu treinamento, que consiste no ajuste dos pesos de cada sinapse e no processo repetitivo de apresentação de um conjunto de valores, pré-definidos, na entrada da rede. Esse conjunto de treinamento representa as características individuais do padrão que se deseja aprender. A amostra dos valores de entrada que foram gerados a partir das funções 4.1 e 4.2, para serem aplicados na rede de classificação podem ser observadas na tabela 4.1. No total foram obtidos 120 valores, 60 para x e 60 para o y correspondente, os valores de x são comuns às duas funções, porém cada uma resulta em um y diferente.

Tabela 4.1 - Amostra dos padrões de entrada utilizados no treinamento da rede de classificação

x	yi (Função 4.1)	yi (Função 4.2)
0.00	0.00	0.00
0.10	0,21	-0,19
0.20	0,44	-0,36
0.30	0,69	-0,51
0.40	0,96	-0,64
0.50	1,25	-0,75
0.60	1,56	-0,84
0.70	1,89	-0,91
0.80	2,24	-0,96
0.90	2,61	-0,99
1.00	3,00	-1,00
1.10	3,41	-0,99
1.20	3,84	-0,96
1.30	4,29	-0,91
1.40	4,76	-0,84
1.50	5,25	-0,75

1.60	5,76	-0,64
1.70	6,29	-0,51
1.80	6,84	-0,36
. .	. .	. .
5.90	46,61	23,01

Em relação ao modelamento da rede neural, não há uma fórmula especifica para determinar, por exemplo, o número de camadas ou o número ideal de neurônios, deve-se levar em conta que cada problema a ser representado é diferente em cada caso. A seguir foram selecionadas algumas sugestões de modelagem:

- Cybenko (1989) sugere que apenas uma camada intermediária é suficiente para aproximar uma função contínua e duas para programar qualquer função matemática.

- Segundo Braga et al (2000) quando utilizamos muitas camadas à rede se torna menos precisa, devido ao erro médio calculado durante o treinamento, somente a camada de saída tem o valor real do erro cometido pela rede, a camada anterior recebe apenas uma estimativa do erro e as camadas escondidas vão receber a estimativa da estimativa e assim por diante.

- Silva et al (2006) preconiza que o número de neurônios em cada camada geralmente é definido empiricamente e deve-se ter cuidado para não exagerar e a rede armazenar os dados do treinamento ao invés de extrair suas características e nem uma quantidade pequena, que elevaria o tempo de generalização da rede.

A configuração adotada para a rede de classificação apresenta uma camada de entrada com dois parâmetros, uma camada escondida com cinco neurônios e um neurônio de saída, como ilustra a figura 4.2. Durante o treinamento foi possível acrescentar mais um neurônio na rede neural de classificação, bem como diminuir.

Com o acréscimo de um neurônio não se obteve resultados significativos que culminasse na alteração da topologia, porém com um neurônio a menos foi necessário mais ciclos para que a rede conseguisse convergir, aumentando assim o tempo de treinamento.

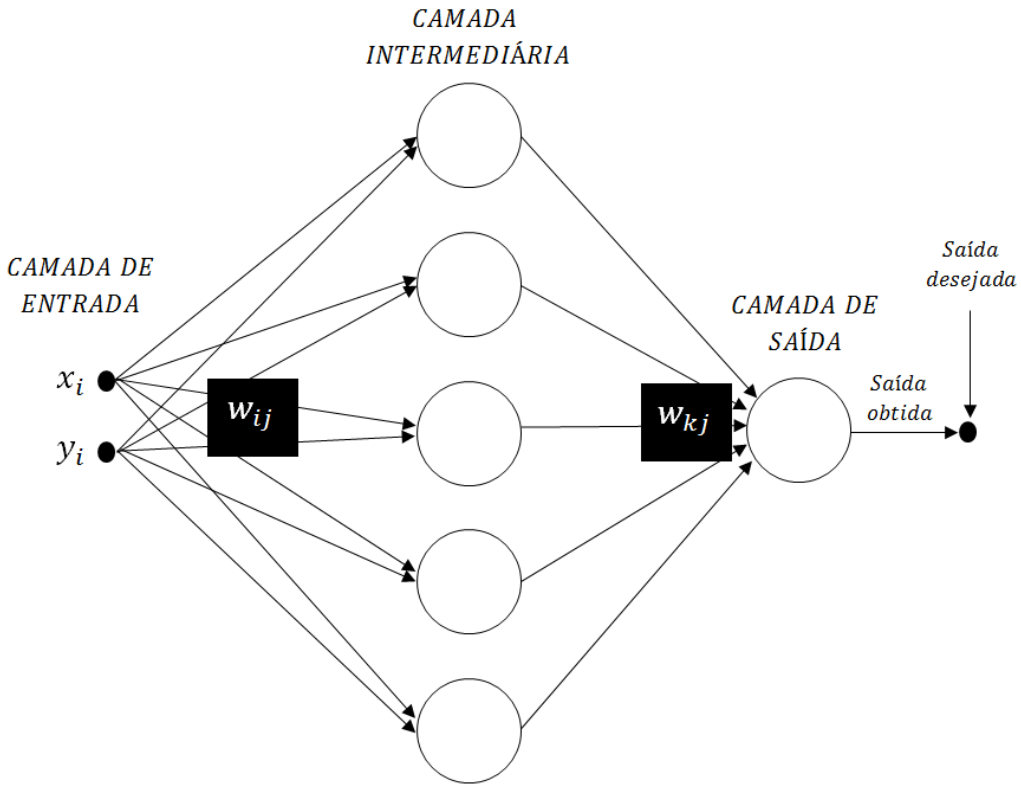

Figura 4.2 - Rede neural de classificação.

O treinamento da rede é dividido em duas partes a de propagação dos valores de entrada até a camada de saída e a de retro propagação do erro calculado, caso a resposta na saída da rede neural não seja a esperada. A propagação consiste em apresentar, um a um, os valores gerados para o treinamento na camada de entrada da rede neural, esse processo é realizado inúmeras vezes ou por n ciclos que são definidos de acordo com a aderência da rede durante o treinamento, ou seja, até que a rede aprenda a classificar ou diferenciar uma função da outra.

Durante o treinamento da rede de classificação observou-se que, se fosse inserido todos os valores gerados da função 4.1 e depois o da função 4.2 a rede de classificação primeiramente convergiria para a função 4.1 e depois para a 4.2, ou

seja, ficariam armazenadas somente as características da função 4.2, que no caso iria sobrepor às características da primeira. Porém como o objetivo da rede é a de classificação dos padrões, então os valores foram inseridos alternadamente como mostrados na figura 4.3.

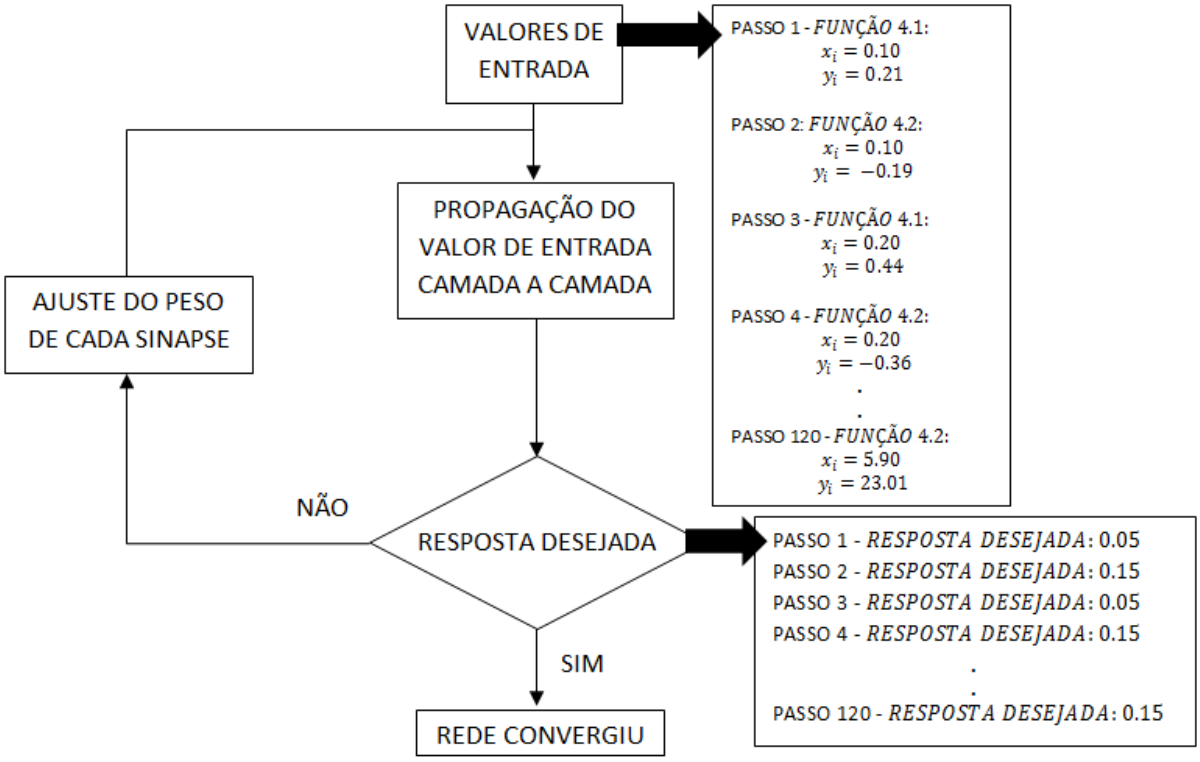

Figura 4.3 - Diagrama de blocos de treinamento da rede de classificação.

Adotou-se como valor esperado para representar a função 4.1 →0.05 e para a função 4.2 →0.15. Se ao final da apresentação dos 60 valores a rede neural não conseguir distinguir as funções repete-se novamente o processo de apresentação, por n ciclos. Os ciclos de treinamento indicam quantas vezes foi necessário apresentar os valores, da tabela 4.1, para que a rede conseguisse convergir. Nos testes realizados foram considerados 15.000 ciclos com uma taxa de aprendizado de 0.2, o ideal para que a rede apresentasse uma boa generalização. É essa a taxa que controla a intensidade das variações dos pesos. O algoritmo desenvolvido para o treinamento da rede de classificação pode ser visualizado no apêndice I.

Depois do treinamento da rede é necessário apresentar valores diferentes dos utilizados no treinamento para verificar se realmente a rede neural respondeu aos estímulos. Uma amostra dos valores de saída da rede depois de treinada pode ser visualizada na tabela 4.2.

Tabela 4.2 - Amostra dos valores de saída da rede de classificação

VALOR DE REFERÊNCIA	SAÍDA DA REDE TREINADA	
0.05	0.103	REPROVADO
0.15	0.103	REPROVADO
0.05	0.095	APROVADO
0.15	0.109	APROVADO
0.05	0.087	APROVADO
0.15	0.115	APROVADO
0.05	0.080	APROVADO
0.15	0.121	APROVADO
0.05	0.073	APROVADO
0.15	0.126	APROVADO
0.05	0.067	APROVADO
0.15	0.132	APROVADO
0.05	0.061	APROVADO
0.15	0.137	APROVADO
0.05	0.057	APROVADO
0.15	0.141	APROVADO
0.05	0.053	APROVADO
0.15	0.145	APROVADO
0.05	0.050	APROVADO
0.15	0.149	APROVADO
0.05	0.048	APROVADO
0.15	0.152	APROVADO
0.05	0.046	APROVADO
0.15	0.154	APROVADO
0.05	0.045	APROVADO
0.15	0.156	APROVADO
0.05	0.044	APROVADO
0.15	0.156	APROVADO
0.05	0.044	APROVADO
0.15	0.157	APROVADO
0.05	0.044	APROVADO
0.15	0.156	APROVADO
0.05	0.044	APROVADO
0.15	0.156	APROVADO

A resposta da rede de classificação pode ser considerada satisfatória, quando analisado a tabela 4.2 somente os dois valores iniciais não conseguiram ser classificados, porém, isso ocorre porque os valores iniciais das duas funções coincidem. Como a rede conseguiu diferenciar as duas funções, os pesos da última alteração antes da rede convergir são armazenados, pois os mesmos representam a

solução do problema apresentado à rede. A mesma rede do treinamento é utilizada para se obter a saída classificada, porém agora são utilizados os pesos fixos.

4.2 Treinamento da rede neural de atuação.

A rede de atuação tem por finalidade efetuar o ajuste da velocidade atual para a velocidade desejada, assim o seu treinamento foi realizado baseado nas características individuais de cada carga. Como foram utilizados dois padrões, um para cada carga, a rede será treinada duas vezes, gerando um conjunto de pesos para representar cada um dos padrões. Assim quando a rede de atuação receber a saída classificada, a mesma realizará o ajuste da velocidade com o conjunto de pesos que representa aquele sinal. Para o aprendizado da rede de atuação, também foi utilizado o método *Backpropagation* em uma *Multilayer Perceptron*, porém a configuração é diferente da apresentada na rede de classificação. A rede de atuação apresenta uma camada de entrada com um parâmetro, uma camada intermediária com cinco neurônios e um neurônio de saída conforme ilustrado na figura 4.4.

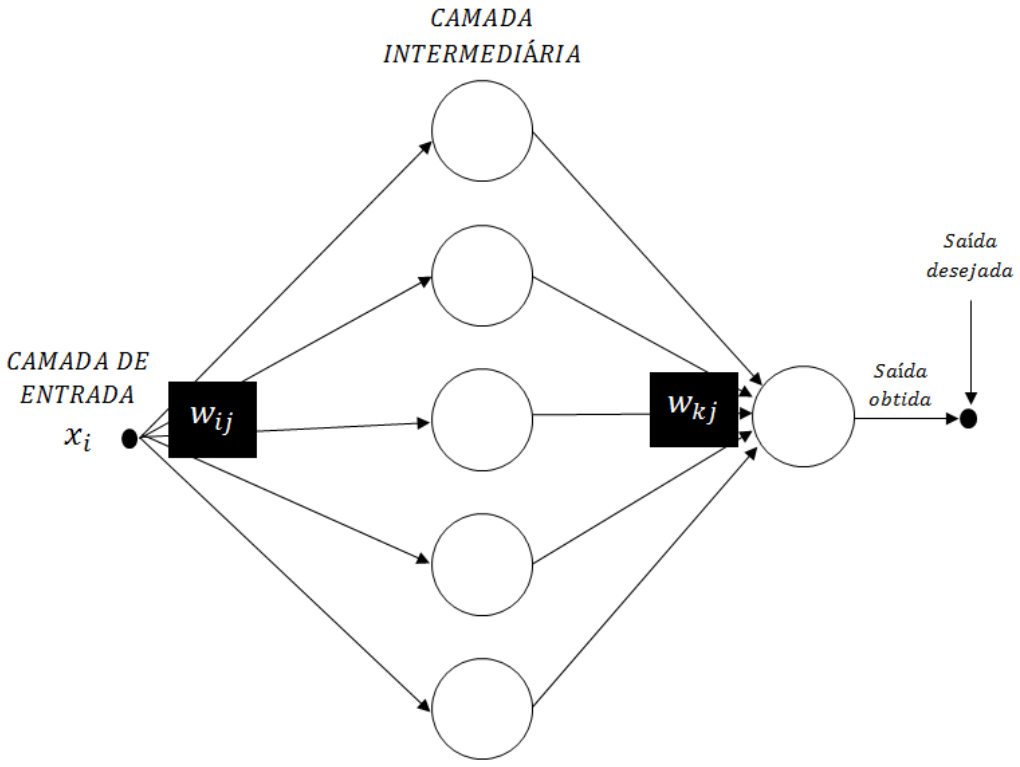

Figura 4.4 - Rede neural de atuação.

O treinamento da rede de atuação também é dividido em duas partes, a de propagação e a de retro propagação. A apresentação dos padrões segue o treinamento da rede de classificação, ou seja, apresentar por n ciclos, um a um, os valores gerados para o treinamento na camada de entrada, até que a rede aprenda as características das funções. Porém a grande diferença entre os treinamentos, da rede de classificação e a de atuação, é que uma é treinada para diferenciar as funções e a outra para aprender as funções. No apêndice II é possível analisar o algoritmo de treinamento dos dois padrões na rede de atuação. Primeiramente a rede é treinada para aprender a função 4.1 e depois a função 4.2, na figura 4.5 pode-se observar como os dados que representam o padrão 1 foram inseridos, o mesmo se repete para o padrão 2.

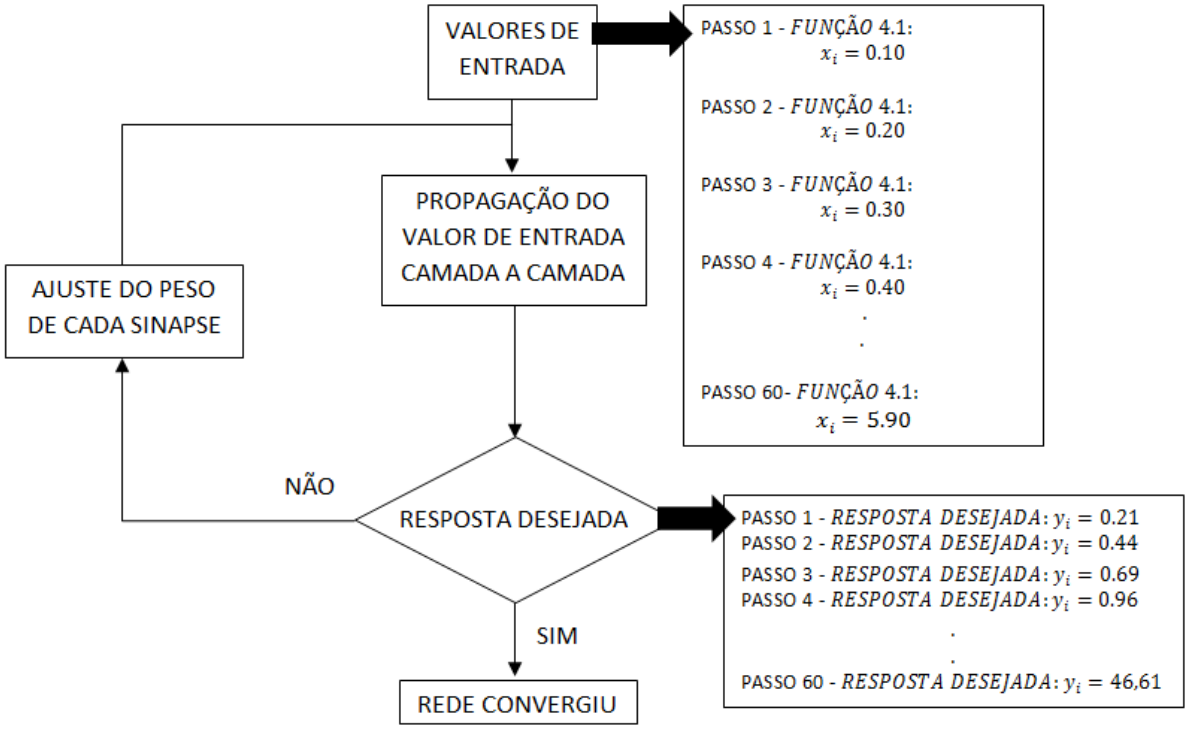

Figura 4.5 - Diagrama de blocos de treinamento da rede de Atuação.

Conforme podemos observar na figura 4.5, somente um valor será propagado da entrada x_i até resultar em um valor de saída e esse valor será comparado ao desejado y_i. Assim que a rede convergir, nesse caso representar internamente o padrão 4.1, os pesos são armazenados, em seguida a rede é treinada novamente, mas para aprender o novo padrão a função 4.2.

Para esse treinamento foram utilizados os 120 valores gerados anteriormente para a rede de classificação. Durante o treinamento da rede com os valores de entrada referente a função 4.1 observou-se que 0 até 5000 ciclos a rede não apresenta resultados significativos. Porém quando analisado os valores de 5000 até 10.000 ciclos nota-se que a rede representa os padrões de forma coerente, acima de 10.000 ciclos a rede não apresentou alterações consideráveis, mas quando a taxa de aprendizado passou de 0.2 para 0.3 observa-se uma leve aproximação do padrão de entrada. Na figura 4.6 pode-se observar a aderência da rede quando treinada para representar a função 4.8.

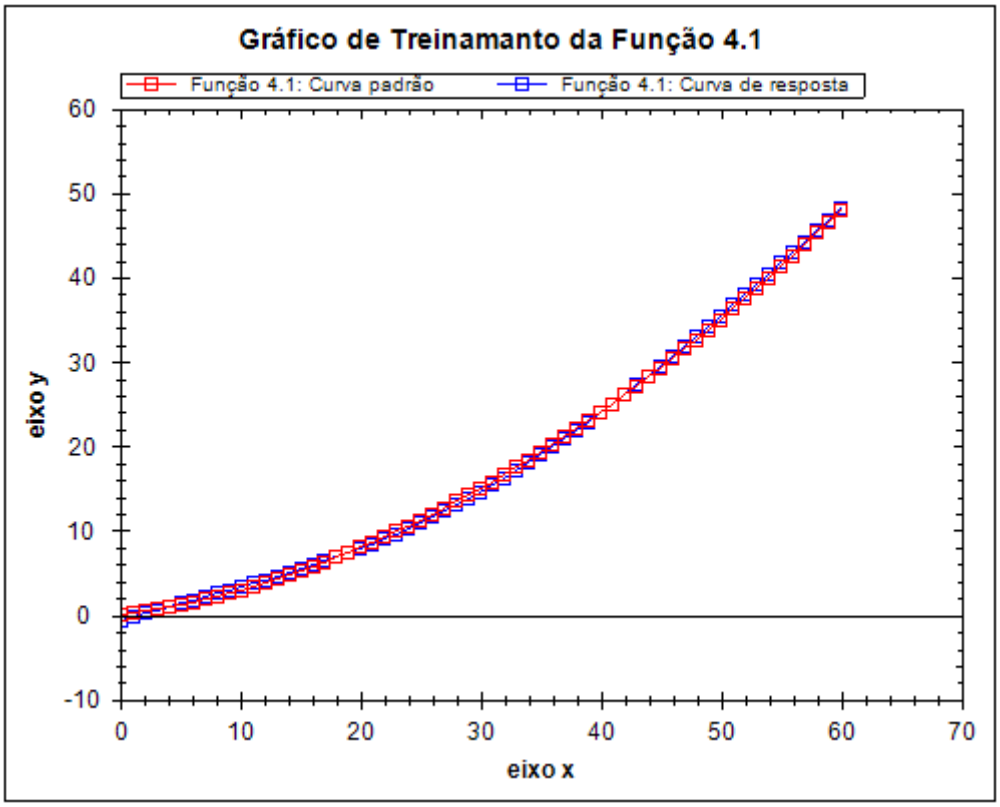

Figura 4.6 - Gráfico da curva padrão e curva de resposta da rede neural referente a função 4.1.

No treinamento da rede para a função 4.2 somente quando atingido os 35.000 ciclos com uma taxa de aprendizado de 0.2 que a generalização foi atingida. Na

figura 4.7 é possível notar a aderência da rede através da comparação da curva padrão e da curva de resposta da rede treinada.

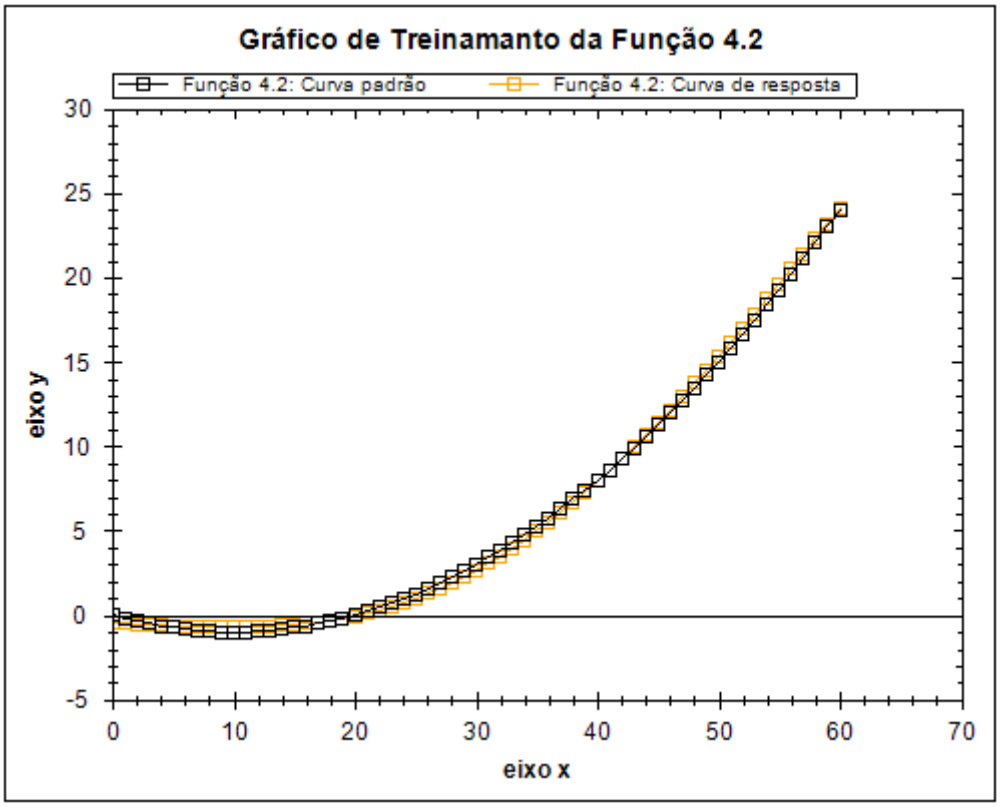

Figura 4.7 - Gráfico da curva padrão e curva de resposta da rede neural referente a função 4.2

Os resultados obtidos com a rede neural de atuação estão dentro do esperado, pois a resposta da rede para ambas as funções, conforme apresentado graficamente nos resultados parciais, demonstrou a viabilidade de representá-las sem a necessidade do modelo matemático, sendo utilizados, no treinamento apenas os valores gerados a partir das funções. Nota-se também que a curva de resposta representa de forma satisfatória a curva padrão, pois devemos considerar que a convergência da rede é uma representação das características dos dados e não um modelo fiel.

Capítulo 5 – ESTUDO PARA IMPLEMENTAÇÃO DO NEUROCONTROLADOR DE VELOCIDADE MICROCONTROLADO.

Como o objetivo principal do trabalho é a implementação do neurocontrolador no microcontrolador, serão apresentados nesse capítulo o resultados obtidos com o treinamento das redes neurais com os dados do sistema que será controlado e o algoritmo para implementação no microcontrolador.

5.1 Aquisição dos dados para treinamento do neurocontrolador.

Para a aquisição dos dados reais, tensão e rotação, necessários para o treinamento das redes neurais, foi montando uma planta utilizando um motor CC de 5V, uma fonte de alimentação DC Minipa (MPL-1303M), um multímetro digital Minipa (Et-1110) e um tacômetro digital EGA Master. Os dados foram extraídos do sistema de forma manual, ou seja, conforme a tensão era modificada na fonte de alimentação, os valores do multímetro e do tacômetro eram registrados. No capítulo anterior foram utilizadas duas funções para representar duas cargas distintas, porém para o treinamento com os dados reais foi adicionada mais uma carga, assim durante o experimento foram geradas curvas para três situações diferentes, motor sem carga, com carga de 100g e com carga de 300g. Na figura 5.1 é possível notar o comportamento da variável controlada quando submetida a cada uma das cargas.

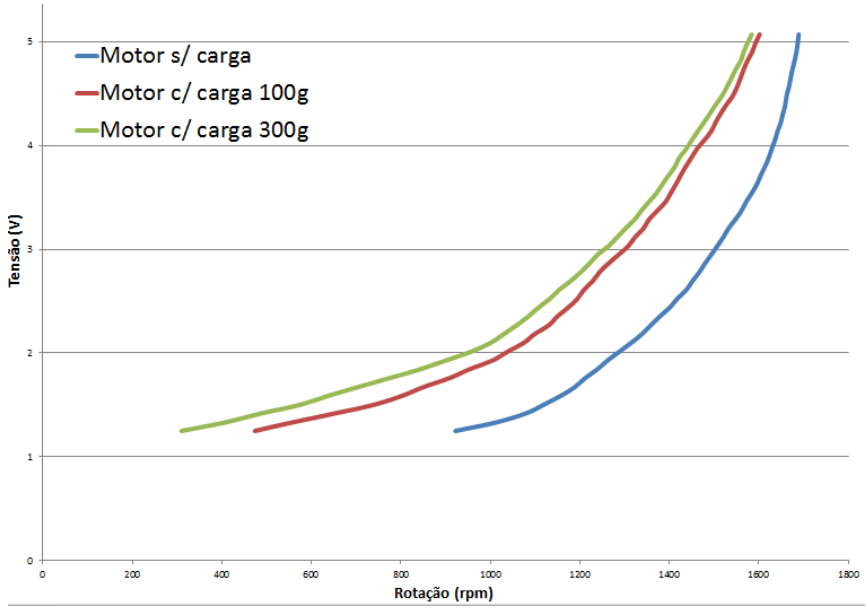

Figura 5.1 - Curvas características da variável controlada com 3 cargas distintas.

No experimento foram gerados um total 46 valores para tensão e 46 valores de rotação para cada situação simulada, na tabela 5.1 verifica-se que a tensão foi mantida fixa provocando uma variação na rotação do motor de acordo com a carga inserida.

Tabela 5.1 - Amostra dos padrões gerados no experimento.

Tensão (V)	Rotação (rpm) s/ carga	Rotação (rpm) c/ carga 100g	Rotação (rpm) c/ carga 300g
1,25	922	474	310
1,33	1011	555	405
1,42	1080	654	489
1,5	1118	738	573
1,59	1158	804	639
1,67	1188	849	702
1,76	1212	909	775
1,84	1238	951	840
1,93	1262	1005	901
2,01	1287	1037	955
2,1	1315	1075	1001
2,18	1338	1098	1029
2,27	1359	1131	1059
2,35	1378	1149	1084
2,44	1401	1174	1108
2,52	1417	1194	1132
2,61	1438	1209	1154
2,69	1451	1228	1178
2,78	1467	1244	1202
2,86	1479	1263	1221
2,95	1494	1287	1240
3,03	1507	1308	1264
3,12	1521	1324	1284
3,2	1532	1342	1302
3,29	1548	1355	1323
3,37	1561	1373	1337
3,46	1572	1392	1355
3,54	1584	1403	1371
3,63	1596	1415	1385
3,71	1604	1425	1398
3,8	1614	1437	1413
3,88	1622	1449	1422
3,97	1629	1462	1438
4,05	1636	1478	1450

4,14	1642	1494	1464
4,22	1649	1503	1477
4,31	1654	1515	1491
4,39	1659	1527	1503
4,48	1662	1542	1518
4,56	1667	1551	1528
4,65	1671	1559	1539
4,73	1675	1566	1548
4,82	1680	1575	1560
4,9	1684	1585	1566
4,99	1687	1593	1575
5,07	1689	1602	1584

Nota-se que na tabela, a tensão não começa em 0 volts, isso deve-se ao fato do motor não apresentar rotação antes do valor de 1.25 volts.

5.2 Treinamento da rede neural de classificação com os dados obtidos da variável controlada.

O treinamento da rede de classificação foi realizado utilizando o algoritmo do apêndice III, o código desenvolvido é o mesmo do apêndice I com algumas alterações, pois foi acrescida uma carga para a classificação. No treinamento anterior foram adotados dois valores, 0.05 e 0.15, para representar as saídas classificadas, como foram identificadas três cargas, os valores adotados foram:

- Motor sem carga: 0,1;
- Motor com carga 1 (100 g): 0,2;
- Motor com carga 2 (300 g): 0,3.

Durante o treinamento, os valores foram inseridos alternadamente, exatamente como o treinamento realizado anteriormente, dessa forma a rede aprende a diferenciar as curvas do motor e não a extrair as características individuais de cada uma. Na figura 5.2, observa-se as entradas dos dados na rede e a resposta esperada para cada situação apresentada.

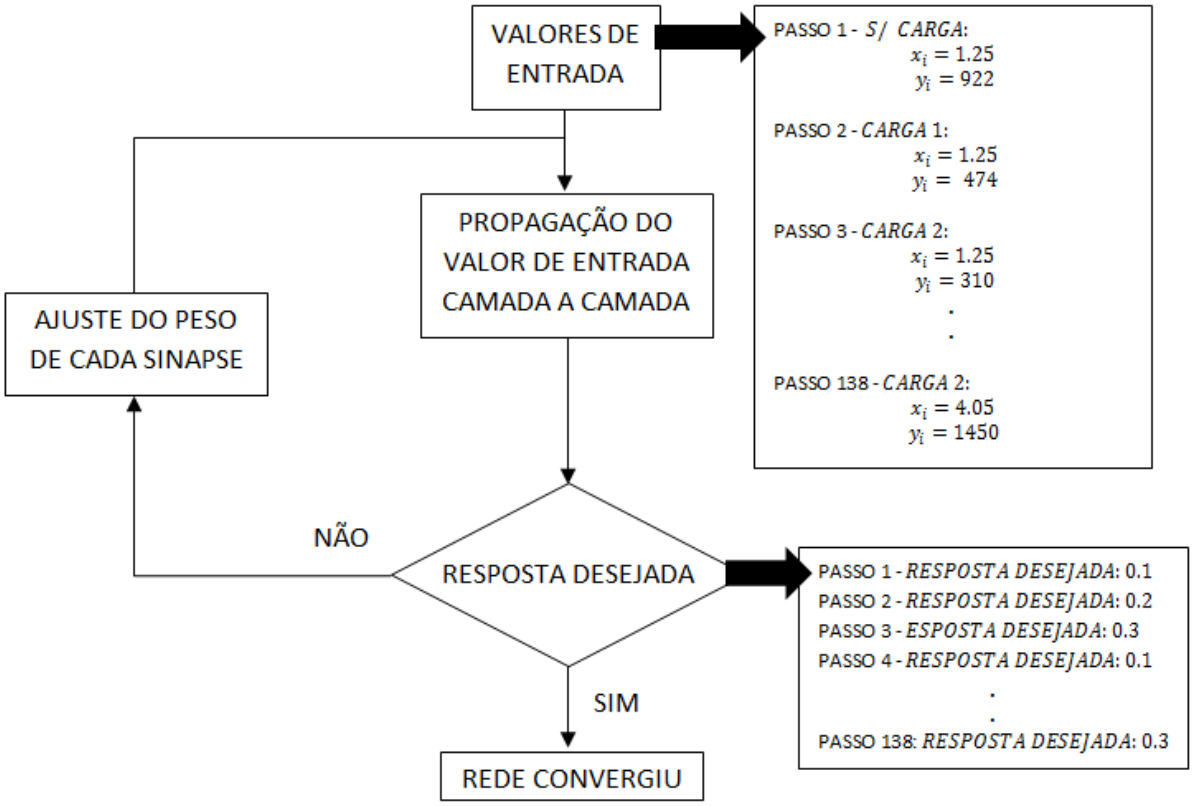

Figura 5.2 - Diagrama de blocos de treinamento da rede de classificação com os valores da variável controlada.

Conforme demonstrado na tabela 5.2, os resultados obtidos foram considerados satisfatório, após serem executados 150.000 ciclos, com uma taxa de aprendizado de 0,2.

Tabela 5.2 - Amostra dos valores de saída da rede de classificação treinada com os dados da variável controlada.

VALOR DE REFERÊNCIA	SAÍDA DA REDE TREINADA	
0,1	0,072453	APROVADO
0,2	0,20999	APROVADO
0,3	0,306537	APROVADO
0,1	0,098093	APROVADO
0,2	0,192021	APROVADO
0,3	0,291727	APROVADO
0,1	0,106672	APROVADO
0,2	0,178138	APROVADO
0,3	0,286339	APROVADO
0,1	0,107486	APROVADO
0,2	0,179566	APROVADO

0,3	0,280433	APROVADO
0,1	0,107178	APROVADO
0,2	0,19786	APROVADO
0,3	0,294789	APROVADO
0,1	0,105982	APROVADO
0,2	0,215839	APROVADO
0,3	0,307177	APROVADO
0,1	0,10246	APROVADO
0,2	0,220898	APROVADO
0,3	0,317002	APROVADO

Após a rede ser treinada a mesma foi implementada no microcontrolador, para isso os pesos resultantes do processo de treinamento foram armazenados. É importante ressaltar que a rede implementada no microcontrolador não será treinada, a mesma se utilizará dos pesos fixos para responder aos estímulos externos. Na tabela 5.3 estão representados os pesos referentes à rede de classificação.

Tabela 5.3 - Pesos da rede de classificação

Vij	{6.60,0.42,7.05,4.45,1.38,5.99}
Wij	{1.64,0.00,-1.91,2.13,-0.00,-1.08}
Vkj	{-6.00,-0.21,-20.15,-7.37,-1.42,-3.63}
V0	{-3.81,-0.37,-1.86,-5.00,-1.09,-6.30}
W0	{1.53}

5.3 Treinamento da rede neural de atuação com os dados obtidos da variável controlada.

O treinamento da rede de atuação foi realizado em três etapas distintas, pois cada carga deverá ter seu próprio conjunto de pesos para sua representação. O algoritmo utilizado está demonstrado no apêndice IV, que no caso é o mesmo utilizado para o treinamento das funções 4.1 e 4.2 que está no apêndice II, exceto por possuir uma rotina a mais devido à carga que foi incluída. A rede de atuação extrai as características de cada curva individualmente, dessa forma cada treinamento irá convergir em um ciclo e com uma taxa de aprendizado diferente. A seguir estão listados os parâmetros utilizados no treinamento, até se obter resultados considerados satisfatórios, para cada uma das curvas treinadas.

- Rede treinada sem carga: 200.000 ciclos–taxa de aprendizado: 0,2
- Rede treinada com carga 100g: 50.000 ciclos–taxa de aprendizado: 0,2
- Rede treinada com carga 300g: 45.000 ciclos–taxa de aprendizado: 0,2

Os conjuntos de pesos, referentes a rede de atuação, que serão implementados no miccrocontrolador podem ser visualizados nas tabelas 5.4, 5.5 e 5.6.

Tabela 5.4 - Pesos da rede de atuação s/ carga

Vij	{8.09,9.80,0.12,2.35,5.26,1.07 }
Wij	{2.04,7.66,0.05,0.98,-3.85,0.58 }
V0	{-11.21,-18.57,-0.14,-1.98,-7.88,-0.87 }
W0	{6.23}

Tabela 5.5 - Pesos da rede de atuação c/ carga 100g

Vij	{0.23,0.67,0.24,3.19,0.25,1.10 }
Wij	{0.28,0.49,0.26,3.10,0.26,-0.44 }
V0	{-0.02,-0.42,-0.05,-6.95,-0.09,-1.46 }
W0	{3.13}

Tabela 5.6 - Pesos da rede de atuação c/ carga 300g

Vij	{0.50,0.28,1.65,2.94,0.15,0.58};
Wij	{0.64,0.38,-0.69,3.17,0.21,0.73 }
V0	{0.01,0.07,-2.29,-6.33,0.05,0.05 }
W0	{2.71}

5.4 Implementação do algoritmo das redes neurais de classificação e atuação no microcontrolador.

O microcontrolador utilizado para a implementação do neurocontrolador é o Atmega328, essa escolha se deu pela facilidade de programação e pela interface de comunicação serial disponível entre esse microcontrolador e o computador, que permite o controle de dispositivos, aquisição e entrada de dados.

Na figura 5.3 pode-se observar que a tensão e rotação atual seriam obtidas através de sensores e inseridas direto no neurocontrolador, mas como a planta de controle não está disponível fisicamente, a entrada dos valores será realizada

manualmente, através da interface serial. A rotação desejada será inserida manualmente pelo operador, como previsto na planta.

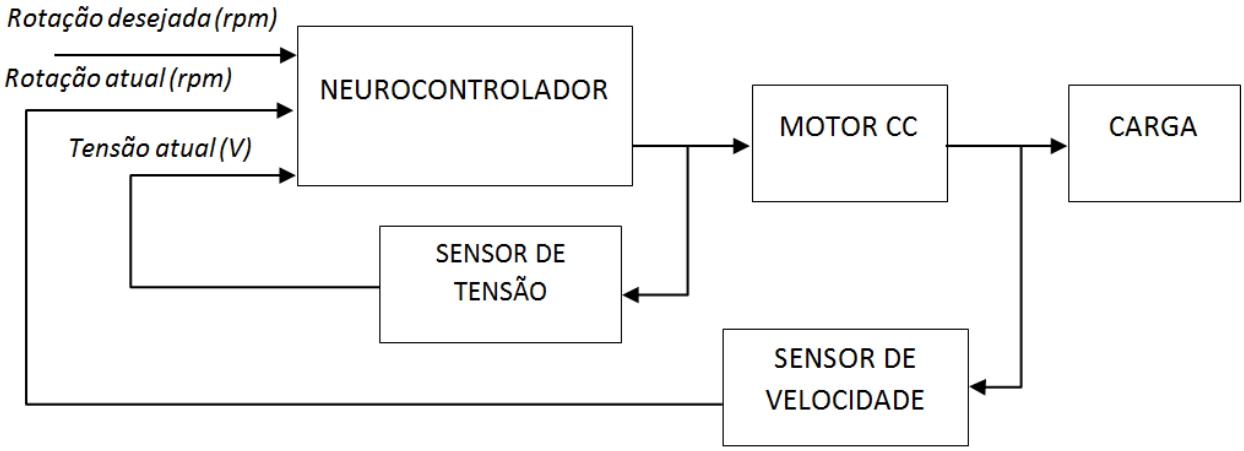

Figura 5.3 - Planta de controle

A interação da interface serial com o neurocontrolador acontece da seguinte forma, inicialmente são inseridos três valores, o valor da rotação desejada e os dois valores de referência da rede de classificação, a tensão e a rotação atual. Esses valores serão inseridos através de uma *string* como mostra a figura 5.4.

Figura 5.4 - Entrada dos valores de referência da rede de classificação e atuação.

Note que os valores inseridos são separados por identificadores, a letra V refere-se à tensão atual, a R está relacionada à rotação atual e a D à rotação desejada. Esses identificadores são necessários para indicar qual é o valor referente a cada parâmetro individualmente, pois inicialmente todos os dados são armazenados em uma sequência de caracteres e para que os mesmos possam ser utilizados pelas redes se faz necessário a separação de cada termo, para isso o identificador evidencia onde começa e termina o conteúdo.

Assim que os valores são enviados para o microcontrolador os valores são separados, as informações são processadas pelas redes neurais e os resultados são mostrados na tela conforme pode se observar na figura 5.5.

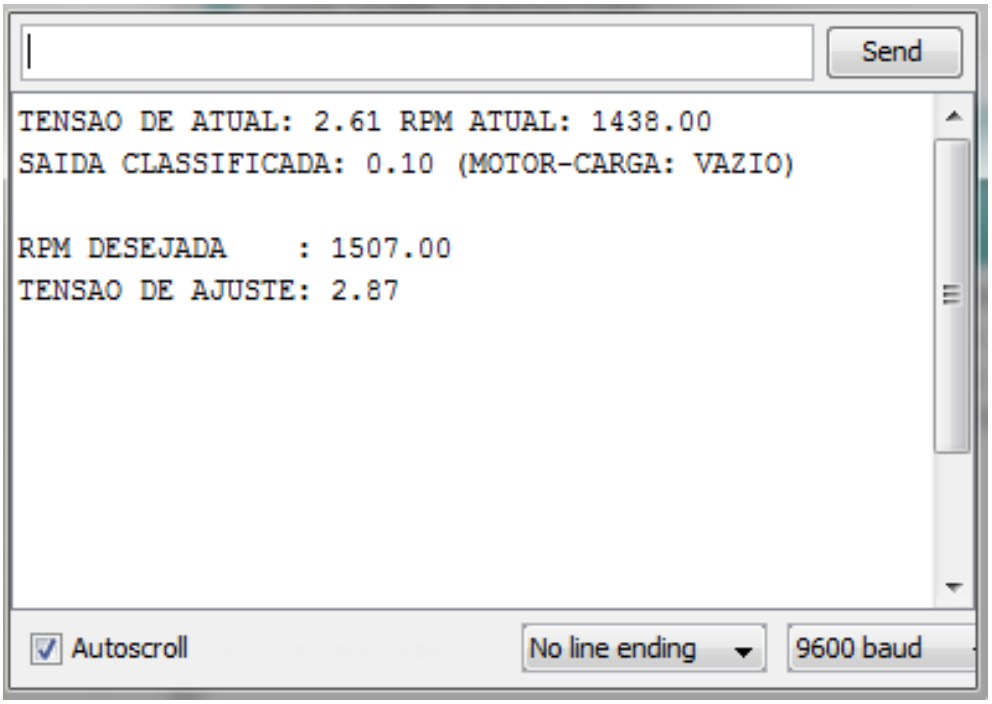

Figura 5.5 - Interface gráfica da comunicação serial.

Como se observa, os valores são inseridos inicialmente na rede de classificação, e o resultado indica qual é o padrão da carga a qual o motor está submetido. Como três cargas foram treinadas, se faz necessário saber qual a carga que está atuando para poder utilizar o conjunto de pesos correspondentes a ela na rede de atuação, dessa forma a rede responderá somente para esse caso específico, se em outra simulação for identificada uma carga diferente então a rede utilizará o outro conjunto de pesos obtidos previamente nos treinamentos. A partir dessa classificação o valor da rotação desejada é inserido na rede de atuação para realizar o ajuste da velocidade. O ajuste indica qual o valor da tensão tem que ser adotado para atingir a rotação desejada, esse valor seria aplicado diretamente no motor.

5.4.1 Resultados obtidos na implementação do Neurocontrolador

Para comprovar a generalização do neurocontrolador foram inseridos diversos valores de entrada, que foram retirados da tabela 5.1. Os testes foram realizados adotando a mesma tensão atual para as três redes, dessa forma o resultado para cada uma deverá ser diferente devido ao fato da carga ser diferente. Nas tabelas 5.7, 5.8, 5.9, 5.10, 5.11 e 5.12 é possível observar os valores que foram inseridos na

rede e o valor esperado na saída da rede de atuação para cada simulação. Os resultados das simulações podem ser visualizados nas figuras 5.6, 5.7, 5.8, 5.9, 5.10 e 5.11.

Tabela 5.7 - Valores referentes a simulação 1

Valores de entrada na interface serial			
Tensão Atual (v)	Rotação Atual (rpm) - motor s/ carga	Rotação Desejada (rpm)	Tensão de Saída esperada para o ajuste (V)
1,93	1262	1401	2,44

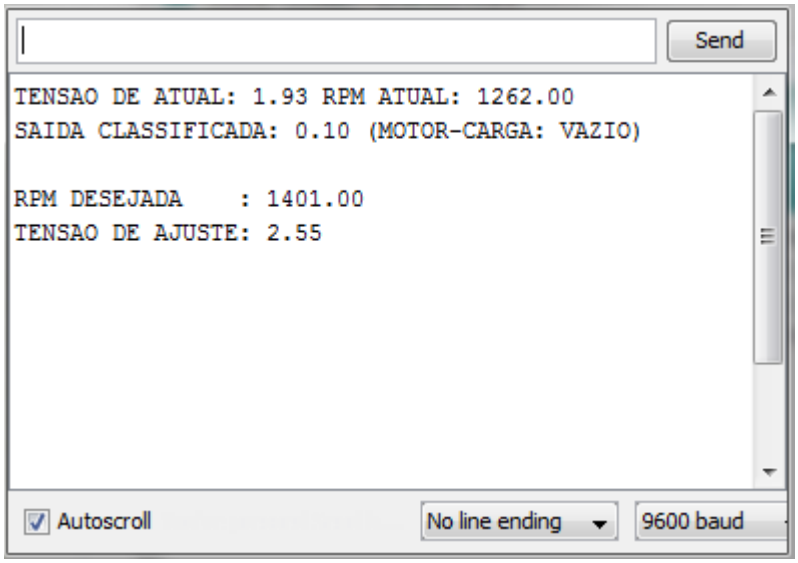

Figura 5.6 - Resposta de saída da simulação 1

$$Erro\ relativo\ percentual\ (simulação\ 1) = \frac{(2.55 - 2.44)}{2.44} x\ 100 = 4.5\%$$

Tabela 5.8 - Valores referentes a simulação 2

Valores de entrada na interface serial			
Tensão Atual (v)	Rotação Atual (rpm) - (motor c/ carga 100g)	Rotação Desejada (rpm)	Tensão de Saída esperada para o ajuste (V)
1,93	1005	1308	3,03

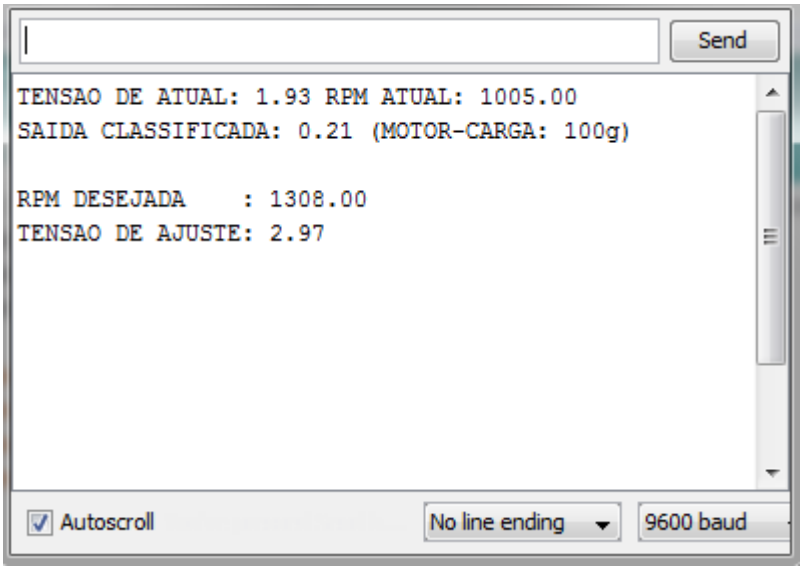

Figura 5.7 - Resposta de saída da simulação 2

$$Erro\ relativo\ percentual\ (simulação\ 2) = \frac{(2.97 - 3.03)}{3.03} x\ 100 = 1{,}98\%$$

Tabela 5.9 - Valores referentes a simulação 3

Valores de entrada na interface serial			
Tensão Atual (v)	Rotação Atual (rpm) - (motor c/ carga 300g)	Rotação Desejada (rpm)	Tensão de Saída esperada para o ajuste (V)
1,93	901	1308	3,12

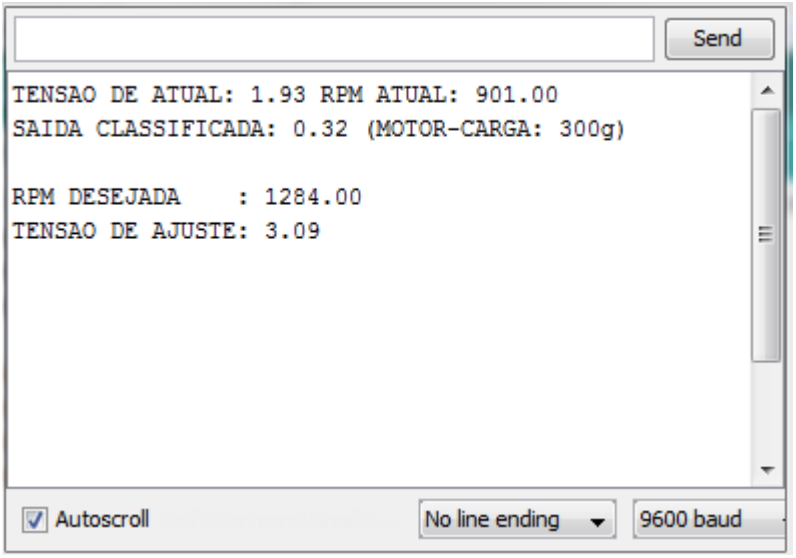

Figura 5.8 - Resposta de saída da simulação 3

$$Erro\ relativo\ percentual\ (simulação\ 3) = \frac{(3.09 - 3.12)}{3.12} x\ 100 = 0{,}96\%$$

Tabela 5.10 - Valores referentes a simulação 4

Valores de entrada na interface serial			
Tensão Atual (v)	Rotação Atual (rpm) - motor s/ carga	Rotação Desejada (rpm)	Tensão de Saída esperada para o ajuste (V)
2,52	1417	1080	1,42

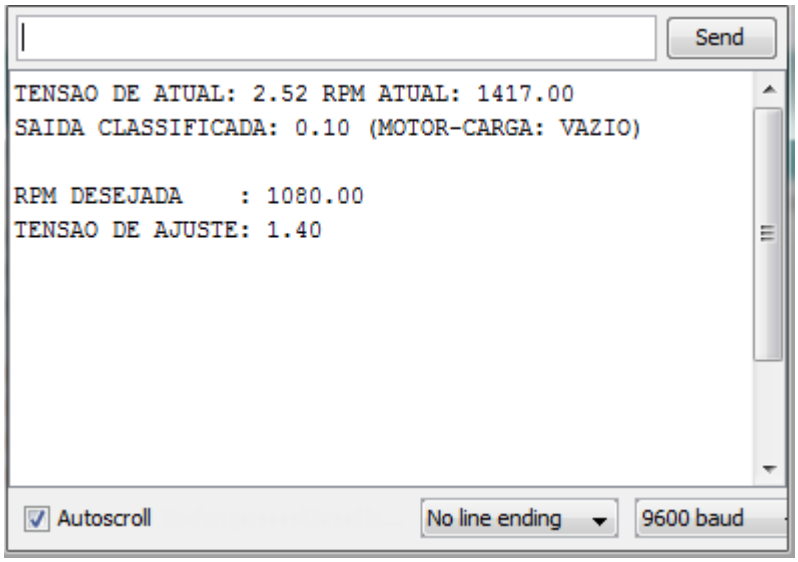

Figura 5.9 - Resposta de saída da simulação 4

$$Erro\ relativo\ percentual\ (simulação\ 4) = \frac{(1.40 - 1.42)}{1.42} x\ 100 = 1.41\%$$

Tabela 5.11 - Valores referentes a simulação 5

Valores de entrada na interface serial			
Tensão Atual (v)	Rotação Atual (rpm) - (motor c/ carga 100g)	Rotação Desejada (rpm)	Tensão de Saída esperada para o ajuste (V)
2,52	1194	1437	3,80

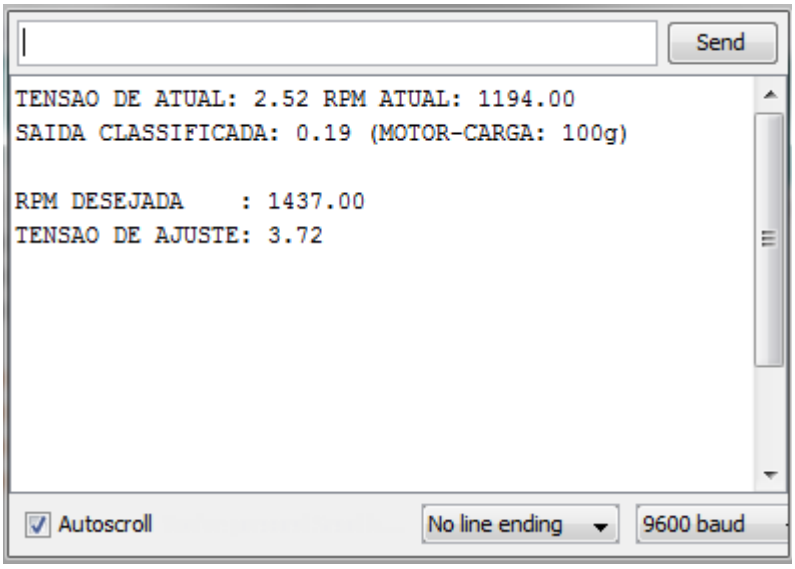

Figura 5.10 - Resposta de saída da simulação 5

$$Erro\ relativo\ percentual\ (simulação\ 5) = \frac{(3.72 - 3.80)}{3.80} x\ 100 = 2{,}10\%$$

Tabela 5.12 - Valores referente a simulação 6

Valores de entrada na interface serial			
Tensão Atual (v)	Rotação Atual (rpm) - (motor c/ carga 300g)	Rotação Desejada (rpm)	Tensão de Saída esperada para o ajuste (V)
2,52	1132	1548	4,73

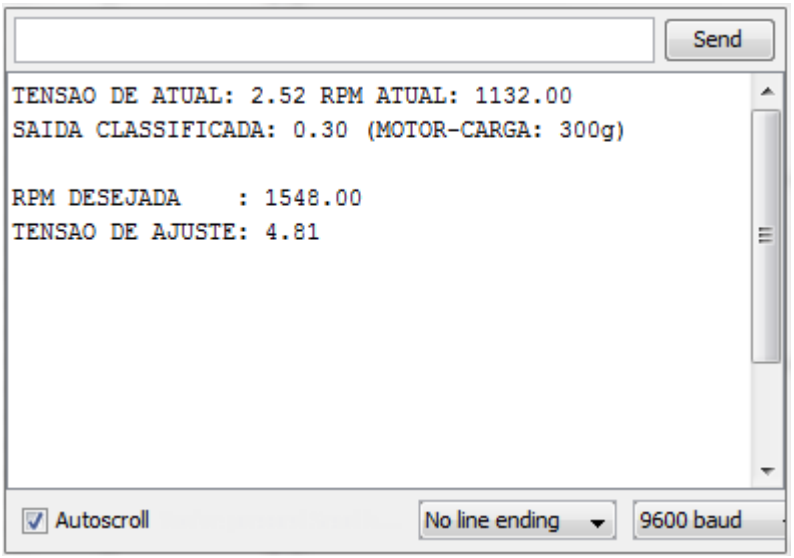

Figura 5.11 - Resposta de saída da simulação 6

$$Erro\ relativo\ percentual\ (simulação\ 6) = \frac{(4.81 - 4.73)}{4.73} x\ 100 = 1{,}69\%$$

Capítulo 6 – CONCLUSÃO

Esta dissertação apresentou o estudo e a implantação das redes neurais artificiais no controle de velocidade de um motor de corrente contínua. O neurocontrolador, como foi denominado esse processo, é composto de duas redes neurais, sendo uma de classificação e outra de atuação, as quais foram implementadas em um microcontrolador. As redes neurais que foram utilizadas são as *Multilayers Perceptron*, treinadas com o algoritmo *Backppropagation*, esse modelo foi escolhido devido ao fato de ser bastante conhecido pela alta capacidade de generalização no reconhecimento de padrões, processamento de sinais, controle de processos entre outros.

Para o desenvolvimento do trabalho optou-se por desenvolver um modelo para teste de viabilidade, das redes neurais, na linguagem C#. Os dados utilizados para o treinamento e testes, foram obtidos através de duas funções matemáticas que apresentam curvas características similares às geradas pela planta de controle. A resposta das redes para ambas as funções, conforme apresentado graficamente, demonstrou a possibilidade de representá-las sem a necessidade da função matemática. Pode-se afirmar seguramente que essa etapa do desenvolvimento foi imprescindível para a continuidade do trabalho, pois ficou demonstrada notoriamente a viabilidade da aplicação das redes neurais artificiais para a solução do problema proposto.

Antes da programação das redes neurais no microcontrolador, optou-se por adicionar mais uma carga no treinamento. Essa inclusão não alterou a estrutura da rede, porém a forma como os dados foram inseridos na rede de classificação, desenvolvida no modelo anterior, teve que ser remodelada. Em relação às modificações da rede de atuação foi realizada uma única alteração, a execução de um treinamento a mais, nesse caso gerando três conjuntos de pesos, um para cada carga.

Conclui-se que quando analisado o comportamento do microcontrolador, programado com as redes neurais, notam-se resultados bastante satisfatórios e observando as semelhanças entre os resultados esperados e os das simulações, pode-se afirmar que o neurocontrolador foi aplicado com sucesso, pois fica evidente a

eficiência das redes na modelagem do fenômeno a ser controlado e atuação no processo.

Ressalta-se também a contribuição deste trabalho nas pesquisas relacionadas a área de controle e automação de processos, podendo o mesmo ser aplicado em diferentes tipos de controle devido a robustez apresentada pelo neurocontrolador.

REFERÊNCIAS BIBLIOGRÁFICAS

ARAÚJO, A. B. et al. **Aproximação de Funções de Dados Meteorológicos usando Redes Neurais *Backpropagation***. Belém, PA – 2012.

BATISTA, B. C. F. **Soluções de Equações Diferenciais Usando Redes Neurais de Múltiplas camadas com os métodos da Descida mais íngreme e Levenberg-Marquardt**, 2012. Programa de Pós-Graduação em Matemática e Estatística (PPGME) da Universidade Federal do Pará – Belém – PA.

BRAGA, A. P.; CARVALHO, A. P. L. F. de; LUDERMIR, T. B. **Redes Neurais Artificiais: Teoria e Aplicações** – Rio de Janeiro: LTC – Livros Técnicos e Científicos Editora S.A., 2000.

CAMPOS, M. M. de; SAITO, K. **Sistemas inteligentes em controle e automação de processos** – Rio de Janeiro: Editora Ciência Moderna Ltda., 2004.

CARRARA, V. **Análise e Controle de Sistemas Lineares**, 2013.

CARRARA, V. **Redes Neurais Aplicadas ao Controle de Atitude de Satélites com Geometria Variável. Tese de Doutorado em Mecânica Espacial e Controle**, 1997 – INPE - São José dos Campos - SP.

CERQUEIRA, E. O. de; ANDRADE, J. C. de; POPPI, R. J. **Redes Neurais e suas aplicações em calibração multivariada**. Química Nova vol.24 nº 6 – São Paulo Nov./Dec. 2001.

CYBENKO, G. **“Approximation by Superpositions of a Sigmoidal Function”**,Math. Control Signals Systems (1989) 2: 303-314.

DEAN, T. **Building Intelligent Robots**, 2002. Brown University – Computer Science.

FAINGOLD, R. **A Revolução Industrial**, 2012.

FERNANDES, A. M. da R. **Inteligência Artificial: noções gerais** – Florianópolis: VisualBooks, 2005.

FERRARA, M. R. **Utilização de Redes Neurais Artificiais para a identificação do diabetes Mellitus**. Marília, SP - 2005.

GOEKING, W. **Da Máquina a vapor aos softwares de automação**, 2010.

LAURENTIZ, S. **Sistemas autônomos, processos de interação e ações criativas**. In: ARS. São Paulo, 2011 - Ano 9 Nº 17.

LOTUFO, F. A. **Método do lugar das Raízes (Root Locus)** – Guaratingueta – SP: Departamento de engenharia Elétrica, 2012.

MACHADO, L. K. **Rede Neural Artificial Embarcada em Robótica Móvel**, 2012.

MARTINS, G. M. **Princípios de Automação Industrial**, 2007.

MÖDERLER, C. 1804: **Viagem inaugural da primeira locomotiva do mundo**, 2012
.

MUELLER, A. **Uma Aplicação de Redes Neurais Artificiais na Previsão do Mercado Acionário** – Florianópolis, 1996.

MUSITANO, M. **Moinhos: energia hidráulica ou eólica**, 2012.

NETO, J. F. G. **Usar Controle Avançado é fácil... e lucrativo!** – Paraná: Revista Intech Brasil, 2006.

OGATA, K. **Engenharia de Controle Moderno**, Prentice Hall Brasil - 4ª edição, 2003.

OLIVEIRA, l. H. **Cérebro humano**, 2012.

OLIVEIRA, A. L. L. **Instrumentação – Fundamentos de Controle de Processo** – SENAI/ CST – ES, 1999.

PINTO, T. S. **As ferramentas na Pré-história**, 2012.

PINTO, F. C. **Sistemas de Automação e Controle – SENAI/ CST** – ES, 2005.

REKIK, L. T.; CHTOUROU***. Fuzzy supervised nonlinear PID control of a class of unknown nonlinear systems***, 2010. Journal of Circuits, Systems, and Computers - Vol. 19, No. 8 (2010) 1847_1862.

SILVA, L. F. C. e. **Modelo de Rede Neural Artificial Treinada com o Algoritmo *Backpropagation*** –Juiz de Fora – MG, 2003.

SILVA, J. M. M. **Introdução as Redes Neurais Artificiais *Perceptrons* de Múltiplas Camadas**, 2014. Universidade Federal Fluminense – Niterói – RJ

SOARES, D. R. **Sistema Inteligente com entrada e saída remota sem fio** – Rio de Janeiro: Universidade do Estado do Rio de Janeiro, 2010.

TONSIG, L. S. **Redes Neurais Artificiais Multicamadas e o Algoritmo Backpropagation**, 2009

UNESCO, **História geral da África, I: Metodologia e pré-história da África** / editado por Joseph Ki-Zerbo.– 2.ed. rev. – Brasília, 2010. 992p

APÊNDICE I – ALGORITMO DE TREINAMENTO DA REDE DE CLASSIFICAÇÃO

```
#region REDE DE CLASSIFICAÇÃO
privatevoid btn_class_Click(object sender, EventArgs e)
      {
          lbl_class.Text = ":";

int cic;
int NN = 6;


for (int j = 0; j < 60; j++) {
              Xc[j] = Xi[j];
              Xc[j + 1] = Xj[j];
              Yc[j] = Yi[j];
Yc[j + 1] = Yj[j];
              j++;
         }


         #region Treinamneto da Rede de Clasificação
for (int i = 0; i <60; i++)
              {
                 c_rede1[i] = 0.05;
c_rede1[i+1] = 0.15;
                 i++;
              }

                  alfa3 = Convert.ToDouble(txt_taxac.Text);
                  cic = Convert.ToInt32(txt_ciclosc.Text);
```

```
double[] DVij3 = newdouble[6];
double[] DVij4 = newdouble[6];
double[] DWjk = newdouble[6];
double[] DVo = newdouble[6];
double DWo;
double[] Zin = newdouble[6];
double[] Zj = newdouble[6];
double dk = 0, Yin = 0, Yk = 0;
double[] dj = newdouble[6];
double ax = 0, axx = 0;

for (int s = 1; s <= cic; s++)
{
for (int n = 0; n < 60; n++)
                        {

for (int j = 0; j < NN; j++)
                            {
                                Zin[j] = 0;
                                Zj[j] = 0;
                                Yin = 0;
                                DVij3[j] = 0;
                                DVij4[j] = 0;
                                DWjk[j] = 0;
                            }

for (int j = 0; j < NN; j++)
                            {
                                Zin[j] = Zin[j] + (Xc[n] * Vij3[j]) + (Yc[n] * (Vij4[j]) + Vo3[j]);
                                Zj[j] = Zin[j];
```

```
                    f_sig(ref Zj[j]);
                    Yin = Yin + Zj[j] * Wjk3[j];
                }

                Yin = Yin + Wo3;
                Yk = Yin;
                ax = Yin;
                f_sig(ref Yk);
f_dsig(ref ax);
                dk = (c_rede1[n] - Yk) * ax;

for (int j = 0; j < NN; j++)
                {
                    DWjk[j] = alfa3 * dk * Zj[j];
                    axx = Zin[j];
                    f_dsig(ref axx);
                    dj[j] = dk * Wjk3[j] * axx;
DVij3[j] = alfa3 * dj[j] * Xc[n];
                    DVij4[j] = alfa3 * dj[j] * Yc[n];
                    DVo[j] = alfa3 * dj[j];
                    Wjk3[j] = Wjk3[j] + DWjk[j];
                    Vij3[j] = Vij3[j] + DVij3[j];
                    Vij4[j] = Vij4[j] + DVij4[j];
                    Vo3[j] = Vo3[j] + DVo[j];
}
                DWo = alfa3 * dk;
                Wo3 = Wo3 + DWo;
}
            }
            lbl_treinamento1.Text = ":OK";
#endregion
```

APÊNDICE II – ALGORITMO DE TREINAMENTO DA REDE ATUAÇÃO

```
#region Treinamento da Rede de Atuação- Função 4.1
        privatevoid btn_trede_Click(object sender, EventArgs e)
        {
                lbl_treinamento1.Text = ":";

        int cic;
        int NN = 6;

        for (int i = 0; i <= 60; i++)
                {
                   Xi[i] = i * 0.1;
                   Yi[i] = ((2 * Xi[i]) + Math.Pow(Xi[i],2))*0.005;
                }

                alfa1 = Convert.ToDouble(txt_taxa1.Text);
                cic = Convert.ToInt32(txt_ciclos1.Text);

        double[] DVij = newdouble[6];
        double[] DWjk = newdouble[6];
        double[] DVo = newdouble[6];
        double DWo;
        double[] Zin = newdouble[6];
        double[] Zj = newdouble[6];
        double dk = 0, Yin = 0, Yk = 0;
        double[] dj = newdouble[6];
        double ax = 0, axx = 0;


        for (int s = 1; s <= cic; s++)
```

```
        {
for (int n = 0; n <= 60; n++)
            {

for (int j = 0; j < NN; j++)
                {
                    Zin[j] = 0;
                    Zj[j] = 0;
                    Yin = 0;
                    DVij[j] = 0;
                    DWjk[j] = 0;
                }

for (int j = 0; j < NN; j++)
                {
                    Zin[j] = Zin[j] + (Xi[n] * Vij1[j]) + Vo1[j];
                    Zj[j] = Zin[j];
                    f_sig(ref Zj[j]);
                    Yin = Yin + Zj[j] * Wjk1[j];
                }

                Yin = Yin + Wo1;
                Yk = Yin;
                ax = Yin;
                f_sig(ref Yk);
                f_dsig(ref ax);
                dk = (Yi[n] - Yk) * ax;

for (int j = 0; j < NN; j++)
                {
                    DWjk[j] = alfa1 * dk * Zj[j];
```

```
                    axx = Zin[j];
                    f_dsig(ref axx);
                    dj[j] = dk * Wjk1[j] * axx;
DVij[j] = alfa1 * dj[j] * Xi[n];
                    DVo[j] = alfa1 * dj[j];
                    Wjk1[j] = Wjk1[j] + DWjk[j];
                    Vij1[j] = Vij1[j] + DVij[j];
                    Vo1[j] = Vo1[j] + DVo[j];
}
                DWo = alfa1 * dk;
                Wo1 = Wo1 + DWo;
}
          }
          lbl_treinamento1.Text = ":OK";
      }
#endregion

#region Treinamento da Rede de Atuação Função 4.2
privatevoid btn_trede2_Click(object sender, EventArgs e)
{

          lbl_treinamento2.Text = ":";

int cic;
int NN = 6;

for (int i = 0; i <= 60; i++)
{
              Xj[i] = i * 0.1;
              Yj[i] = ((-2 * Xj[i]) + Math.Pow(Xj[i], 2)) * 0.005;
}
```

```
            alfa2 = Convert.ToDouble(txt_taxa2.Text);
            cic = Convert.ToInt32(txt_ciclos2.Text);

double[] DVij = newdouble[6];
double[] DWjk = newdouble[6];
double[] DVo = newdouble[6];
double DWo;
double[] Zin = newdouble[6];
double[] Zj = newdouble[6];
double dk = 0, Yin = 0, Yk = 0;
double[] dj = newdouble[6];
double ax = 0, axx = 0;

for (int s = 1; s <= cic; s++)
{
for (int n = 0; n <= 60; n++)
                {

for (int j = 0; j < NN; j++)
                    {
                        Zin[j] = 0;
                        Zj[j] = 0;
                        Yin = 0;
                        DVij[j] = 0;
                        DWjk[j] = 0;
                    }

for (int j = 0; j < NN; j++)
                    {
                        Zin[j] = Zin[j] + (Xj[n] * Vij2[j]) + Vo2[j];
```

```
                    Zj[j] = Zin[j];
                    f_sig(ref Zj[j]);
                    Yin = Yin + Zj[j] * Wjk2[j];
                }

                Yin = Yin + Wo2;
                Yk = Yin;
                ax = Yin;
                f_sig(ref Yk);
                f_dsig(ref ax);
                dk = (Yj[n] - Yk) * ax;

for (int j = 0; j < NN; j++)
                {
                    DWjk[j] = alfa2 * dk * Zj[j];
                    axx = Zin[j];
                    f_dsig(ref axx);
                    dj[j] = dk * Wjk2[j] * axx;
DVij[j] = alfa2 * dj[j] * Xj[n];
                    DVo[j] = alfa2 * dj[j];
                    Wjk2[j] = Wjk2[j] + DWjk[j];
                    Vij2[j] = Vij2[j] + DVij[j];
                    Vo2[j] = Vo2[j] + DVo[j];
}
                DWo = alfa2 * dk;
                Wo2 = Wo2 + DWo;
}
            }
            lbl_treinamento2.Text = ":OK";
        }
#endregion
```

APÊNDICE III – ALGORITMO DE TREINAMENTO DA REDE DE CLASSIFICAÇÃO COM OS VALORES DA VARIÁVEL CONTROLADA.

```
#region REDE DE CLASSIFICAÇÃO
privatevoid btn_class_Click(object sender, EventArgs e)
      {
          lbl_class.Text = ":";
int cic;
int NN = 6;
int cont = 0;
for (int j = 0; j < 60; j++) {
              Xc[j] = Xi[cont];
              Xc[j + 1] = Xj[cont];
              Xc[j + 2] = Xk[cont];
              Yc[j] = Yi[cont];
              Yc[j + 1] = Yj[cont];
              Yc[j + 2] = Yk[cont];
cont++;
              j += 2;
         }

         #region Treinamneto da Rede de Clasificação
for (int i = 0; i <60; i++)
              {
                 c_rede1[i] = 0.10;
                 c_rede1[i + 1] = 0.20;
                 c_rede1[i + 2] = 0.30;
i+=2;
              }

                    alfa4 = Convert.ToDouble(txt_taxac.Text);
```

```
cic = Convert.ToInt32(txt_ciclosc.Text);

double[] DVij3 = newdouble[6];
double[] DVij4 = newdouble[6];
double[] DWjk = newdouble[6];
double[] DVo = newdouble[6];
double DWo;
double[] Zin = newdouble[6];
double[] Zj = newdouble[6];
double dk = 0, Yin = 0, Ykk = 0;
double[] dj = newdouble[6];
double ax = 0, axx = 0;

for (int s = 1; s <= cic; s++)
                    {
for (int n = 0; n < 60; n++)
                        {
for (int j = 0; j < NN; j++)
{
                            Zin[j] = 0;
                            Zj[j] = 0;
                            Yin = 0;
                            DVij3[j] = 0;
                            DVij4[j] = 0;
                            DWjk[j] = 0;
                        }

for (int j = 0; j < NN; j++)
                            {
                            Zin[j] = Zin[j] + (Xc[n] * Vij4[j]) + (Yc[n] * (Vij5[j]) + Vo4[j]);
                            Zj[j] = Zin[j];
```

```
                    f_sig(ref Zj[j]);
                    Yin = Yin + Zj[j] * Wjk4[j];
                }

                Yin = Yin + Wo4;
                Ykk = Yin;
                ax = Yin;
                f_sig(ref Ykk);
f_dsig(ref ax);
                dk = (c_rede1[n] - Ykk) * ax;

for (int j = 0; j < NN; j++)
                {
                    DWjk[j] = alfa4 * dk * Zj[j];
                    axx = Zin[j];
                    f_dsig(ref axx);
                    dj[j] = dk * Wjk4[j] * axx;
DVij3[j] = alfa4 * dj[j] * Xc[n];
                    DVij4[j] = alfa4 * dj[j] * Yc[n];
                    DVo[j] = alfa4 * dj[j];
                    Wjk4[j] = Wjk4[j] + DWjk[j];
                    Vij4[j] = Vij4[j] + DVij3[j];
                    Vij5[j] = Vij5[j] + DVij4[j];
                    Vo4[j] = Vo4[j] + DVo[j];
}
                DWo = alfa4 * dk;
                Wo4 = Wo4 + DWo;
}
        }
#endregion
```

APÊNDICE IV – ALGORITMO DE TREINAMENTO DA REDE DE ATUAÇÃO COM OS VALORES DA VARIÁVEL CONTROLADA.

```
#region Treinamento da Rede 1
        privatevoid btn_trede_Click(object sender, EventArgs e)
              {

                 lbl_treinamento1.Text = ":";

        int cic;
        int NN = 6;

                 alfa1 = Convert.ToDouble(txt_taxa1.Text);
                 cic = Convert.ToInt32(txt_ciclos1.Text);



        double[] DVij = newdouble[6];
        double[] DWjk = newdouble[6];
        double[] DVo = newdouble[6];
        double DWo;
        double[] Zin = newdouble[6];
        double[] Zj = newdouble[6];
        double dk = 0, Yin = 0, Yk = 0;
        double[] dj = newdouble[6];
        double ax = 0, axx = 0;



        for (int s = 1; s <= cic; s++)
                 {
        for (int n = 0; n <= 45; n++)
                    {
```

```
for (int j = 0; j < NN; j++)
{
                    Zin[j] = 0;
                    Zj[j] = 0;
                    Yin = 0;
                    DVij[j] = 0;
                    DWjk[j] = 0;
                }

for (int j = 0; j < NN; j++)
                {
                    Zin[j] = Zin[j] + (Xi[n] * Vij1[j]) + Vo1[j];
                    Zj[j] = Zin[j];
                    f_sig(ref Zj[j]);
                    Yin = Yin + Zj[j] * Wjk1[j];
                }

                Yin = Yin + Wo1;
                Yk = Yin;
                ax = Yin;
                f_sig(ref Yk);
                f_dsig(ref ax);
                dk = (Yi[n] - Yk) * ax;

for (int j = 0; j < NN; j++)
                {
                    DWjk[j] = alfa1 * dk * Zj[j];
                    axx = Zin[j];
                    f_dsig(ref axx);
                    dj[j] = dk * Wjk1[j] * axx;
DVij[j] = alfa1 * dj[j] * Xi[n];
```

```
                    DVo[j] = alfa1 * dj[j];
                    Wjk1[j] = Wjk1[j] + DWjk[j];
                    Vij1[j] = Vij1[j] + DVij[j];
                    Vo1[j] = Vo1[j] + DVo[j];
}
                DWo = alfa1 * dk;
                Wo1 = Wo1 + DWo;
}
          }
          lbl_treinamento1.Text = ":OK";
      }
#endregion

#region Treinamento da Rede 2
privatevoid btn_trede2_Click(object sender, EventArgs e)
{

          lbl_treinamento2.Text = ":";

int cic;
int NN = 6;

          alfa2 = Convert.ToDouble(txt_taxa2.Text);
          cic = Convert.ToInt32(txt_ciclos2.Text);

double[] DVij = newdouble[6];
double[] DWjk = newdouble[6];
double[] DVo = newdouble[6];
double DWo;
double[] Zin = newdouble[6];
double[] Zj = newdouble[6];
```

```
double dk = 0, Yin = 0, Yk = 0;
double[] dj = newdouble[6];
double ax = 0, axx = 0;


for (int s = 1; s <= cic; s++)
        {
for (int n = 0; n <= 45; n++)
            {

for (int j = 0; j < NN; j++)
                {
                    Zin[j] = 0;
                    Zj[j] = 0;
                    Yin = 0;
                    DVij[j] = 0;
                    DWjk[j] = 0;
                }

for (int j = 0; j < NN; j++)
                {
                    Zin[j] = Zin[j] + (Xj[n] * Vij2[j]) + Vo2[j];
                    Zj[j] = Zin[j];
                    f_sig(ref Zj[j]);
                    Yin = Yin + Zj[j] * Wjk2[j];
                }

                Yin = Yin + Wo2;
                Yk = Yin;
                ax = Yin;
                f_sig(ref Yk);
```

```
                f_dsig(ref ax);
                dk = (Yj[n] - Yk) * ax;

for (int j = 0; j < NN; j++)
                {
                    DWjk[j] = alfa2 * dk * Zj[j];
                    axx = Zin[j];
                    f_dsig(ref axx);
                    dj[j] = dk * Wjk2[j] * axx;
DVij[j] = alfa2 * dj[j] * Xj[n];
                    DVo[j] = alfa2 * dj[j];
                    Wjk2[j] = Wjk2[j] + DWjk[j];
                    Vij2[j] = Vij2[j] + DVij[j];
                    Vo2[j] = Vo2[j] + DVo[j];
}
                DWo = alfa2 * dk;
                Wo2 = Wo2 + DWo;
}
            }
            lbl_treinamento2.Text = ":OK";
        }
#endregion

#region Treinamento da Rede 3
privatevoid btn_trede3_Click(object sender, EventArgs e)
{

            lbl_treinamento3.Text = ":";

int cic;
int NN = 6;
```

```
            alfa3 = Convert.ToDouble(txt_taxa3.Text);
            cic = Convert.ToInt32(txt_ciclos3.Text);

double[] DVij = newdouble[6];
double[] DWjk = newdouble[6];
double[] DVo = newdouble[6];
double DWo;
double[] Zin = newdouble[6];
double[] Zj = newdouble[6];
double dk = 0, Yin = 0, Ykk = 0;
double[] dj = newdouble[6];
double ax = 0, axx = 0;


for (int s = 1; s <= cic; s++)
            {
for (int n = 0; n <= 45; n++)
                {
for (int j = 0; j < NN; j++)
{
                    Zin[j] = 0;
                    Zj[j] = 0;
                    Yin = 0;
                    DVij[j] = 0;
                    DWjk[j] = 0;
                }

for (int j = 0; j < NN; j++)
                {
                    Zin[j] = Zin[j] + (Xk[n] * Vij3[j]) + Vo3[j];
```

```
                    Zj[j] = Zin[j];
                    f_sig(ref Zj[j]);
                    Yin = Yin + Zj[j] * Wjk3[j];
                }

                Yin = Yin + Wo3;
                Ykk = Yin;
                ax = Yin;
                f_sig(ref Ykk);
                f_dsig(ref ax);
                dk = (Yk[n] - Ykk) * ax;

for (int j = 0; j < NN; j++)
                {
                    DWjk[j] = alfa3 * dk * Zj[j];
                    axx = Zin[j];
                    f_dsig(ref axx);
                    dj[j] = dk * Wjk3[j] * axx;
DVij[j] = alfa3 * dj[j] * Xk[n];
                    DVo[j] = alfa3 * dj[j];
                    Wjk3[j] = Wjk3[j] + DWjk[j];
                    Vij3[j] = Vij3[j] + DVij[j];
                    Vo3[j] = Vo3[j] + DVo[j];
}
                DWo = alfa3 * dk;
                Wo3 = Wo3 + DWo;
}
            }
            lbl_treinamento3.Text = ":OK";
        }
#endregion
```

APÊNDICE V – ALGORITMO DO NEUROCONTROLADOR

```
//Declaração das variáveis para receber os dados pelo Serial Monitor
char Texto[20];
String StrDados;
int cont=0;
float convTensaoAtual=0,convVelocAtual=0, convRotDesejada=0, Sr1=0, Sr2=0, Sr3=0;

//Declaração dos pesos da Rede de Classificação
float Vij[]={6.60,0.42,7.05,4.45,1.38,5.99};
float Wij[]={1.64,0.00,-1.91,2.13,-0.00,-1.08};
float Vkj[]={-6.00,-0.21,-20.15,-7.37,-1.42,-3.63};
float V0[]={-3.81,-0.37,-1.86,-5.00,-1.09,-6.30};
float W0 = 1.53;

//Declaração dos pesos da Rede 1
float Vij1[]={8.09,9.80,0.12,2.35,5.26,1.07};
float Wij1[]={2.04,7.66,0.05,0.98,-3.85,0.58};
float V01[]={-11.21,-18.57,-0.14,-1.98,-7.88,-0.87};
float W01 = 6.23;

//Declaração dos pesos da Rede 2
float Vij2[]={0.23,0.67,0.24,3.19,0.25,1.10};
float Wij2[]={0.28,0.49,0.26,3.10,0.26,-0.44};
float V02[]={-0.02,-0.42,-0.05,-6.95,-0.09,-1.46};
float W02 = 3.13;

//Declaração dos pesos da Rede 3
float Vij3[]={0.50,0.28,1.65,2.94,0.15,0.58};
float Wij3[]={0.64,0.38,-0.69,3.17,0.21,0.73};
```

```
float V03[]={0.01,0.07,-2.29,-6.33,0.05,0.05};
float W03 = 2.71;

void setup(){
  Serial.begin(9600);
}

void loop(){
    convTensaoAtual = 0;
    convVelocAtual = 0;
    convRotDesejada = 0;
    Sr1 = 0;
    Sr2 = 0;
    Sr3 = 0;

  //Recebendo e convertendo os Dados do Serial Monitor
if (Serial.available() > 0){
    delay(100);
    int tamanho = Serial.available();
for (int i = 0; i < tamanho; i++){
        Texto[i] = Serial.read();
        Texto[i+1] = '\0';
      }

    StrDados = Texto;

    int tampalavra =  StrDados.length();
    //Tensao
    int v = StrDados.indexOf('V');
    //Rotacao Atual
    int r = StrDados.indexOf('R');
```

```
    //Rotacao Desejada
int d = StrDados.indexOf('D');

//Armazenando o valor da Tensao Atual
    cont=0;
    char tensaoAtual[r-v];
    while (cont < (r-1)){
      v++;
      tensaoAtual[cont] = StrDados[v];
      cont++;
   }

   //Armazenando o valor da Rotacao Atual
   cont=0;
   char velocAtual[d-r];
int c = d-r-1;
   while (cont < c){
r++;
      velocAtual[cont] = StrDados[r];
      cont++;
   }

   //Armazenando o valor da Rotacao Desejada
   cont=0;
   char rotacaoDesejada[tampalavra-r];
int f = tampalavra-d-1;
   while (cont < f){
d++;
      rotacaoDesejada[cont] = StrDados[d];
      cont++;
   }
```

```
    //Convertendo os valores para double e aplicando o fator de correção
    convTensaoAtual = atof(tensaoAtual)*0.1;
    convVelocAtual = atof(velocAtual)*0.001;
    convRotDesejada = atof(rotacaoDesejada)*0.001;

    Serial.print("TENSAO DE ATUAL: ");
    Serial.print(convTensaoAtual/0.1-0.01);
    Serial.print(" RPM ATUAL: ");
    Serial.println(convVelocAtual/0.001);

    //Classificador
float Yn = 0, Sn = 0, Sx=0, SCla=0;
    for (int j = 0; j < 6; j++){
      Yn = (convVelocAtual * Vij [j]) + (convTensaoAtual * Vkj [j]);
      Yn = Yn + V0[j];
      Sx = (2/(1+exp(-Yn)))-1;
      //f_sig(ref Sx);
      Sn = Sn + Sx * Wij[j];
      }
      Sn = Sn + W0;
      SCla = (2/(1+exp(-Sn)))-1;
Serial.print("SAIDA CLASSIFICADA: ");
      Serial.print(SCla);


    //Atuador - Rede 1
    if(SCla >= 0.06 && SCla <= 0.11){
Yn = 0, Sx = 0, Sn = 0;
      for (int j = 0; j < 6; j++){
Yn = (convRotDesejada * Vij1[j]);
```

```
        Yn = Yn + V01[j];
Sx = (2/(1+exp(-Yn)))-1;
        Sn = Sn + Sx * Wij1[j];
        }
        Sn = Sn + W01;
        Sr1 = ((2/(1+exp(-Sn)))-1)/0.1;

Serial.println(" (MOTOR-CARGA: VAZIO)");
        Serial.println("");
        Serial.print("RPM DESEJADA    : ");
        Serial.println(convRotDesejada/0.001);
        Serial.print("TENSAO DE AJUSTE: ");
        Serial.println(Sr1);
        }

    //Atuador - Rede 2
if(SCla >= 0.16   && SCla <= 0.22){
     Yn = 0, Sx = 0, Sn = 0;
for (int j = 0; j < 6; j++){
        Yn = (convRotDesejada * Vij2[j]);
Yn = Yn + V02[j];
        Sx = (2/(1+exp(-Yn)))-1;
        Sn = Sn + Sx * Wij2[j];
        }
        Sn = Sn + W02;
        Sr2 = ((2/(1+exp(-Sn)))-1)/0.1;

Serial.println(" (MOTOR-CARGA: 100g)");
        Serial.println("");
        Serial.print("RPM DESEJADA    : ");
        Serial.println(convRotDesejada/0.001);
```

```
        Serial.print("TENSAO DE AJUSTE: ");
        Serial.println(Sr2);
        }

    //Atuador - Rede 3
if(SCla >= 0.27 && SCla <= 0.32){
      Yn = 0, Sx = 0, Sn = 0;
for (int j = 0; j < 6; j++){
        Yn = (convRotDesejada * Vij3[j]);
Yn = Yn + V03[j];
        Sx = (2/(1+exp(-Yn)))-1;
        Sn = Sn + Sx * Wij3[j];
        }
        Sn = Sn + W03;
        Sr3 = ((2/(1+exp(-Sn)))-1)/0.1;

Serial.println(" (MOTOR-CARGA: 300g)");
        Serial.println("");
        Serial.print("RPM DESEJADA    : ");
        Serial.println(convRotDesejada/0.001);
        Serial.print("TENSAO DE AJUSTE: ");
        Serial.println(Sr3);
        }

    delay(5000);
    Serial.println("");
 }
}
```