Universidade Federal de Mato Grosso Do Sul
Faculdades de Engenharias, Arquitetura e Urbanismo e Geografia
Dissertação de Mestrado do Curso de Engenharia Elétrica

André D'Estefani Müller Ribeiro

Implementação em FPGA de Algoritmos para o Controle de Motores Trifásicos Baseados em Redes Neurais Artificiais com Função de Ativação Projetada Usando Algoritmos Genéticos

Campo Grande – MS
30 de Julho de 2020

Universidade Federal de Mato Grosso Do Sul
Faculdades de Engenharias, Arquitetura e Urbanismo e Geografia
Dissertação de Mestrado do Curso de Engenharia Elétrica

André D'Estefani Müller Ribeiro

Implementação em FPGA de Algoritmos para o Controle de Motores Trifásicos Baseados em Redes Neurais Artificiais com Função de Ativação Projetada Usando Algoritmos Genéticos.

Dissertação submetida à Banca Examinadora do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Mato Grosso do Sul para a obtenção do Grau de Mestre em Engenharia Elétrica.

Orientador: Professor Doutor Raymundo Cordero García

Campo Grande – MS,
30 de Julho de 2020

IMPLEMENTAÇÃO EM FPGA DE ALGORITMOS PARA O CONTROLE DE MOTORES TRIFÁSICOS BASEADOS EM REDES NEURAIS ARTIFICIAIS COM FUNÇÃO DE ATIVAÇÃO PROJETADA USANDO ALGORITMOS GENÉTICOS.

ANDRÉ D'ESTEFANI MULLER RIBEIRO

Esta Dissertação foi julgada adequada para obtenção do Título de Mestre em Engenharia Elétrica e aprovada em sua forma final pelo Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Mato Grosso do Sul.

_____

Raymundo Cordero García, Dr.
Orientador

_____

Edson Antônio Batista. Dr.
Coordenador do Programa de Pós-Graduação em Engenharia Elétrica

Banca Examinadora:

_____

Raymundo Cordero García, Dr.
Presidente

_____

Walter Issamu Suemitsu, Dr. Ing.

_____

João Onofre Pereira Pinto, Dr.

_____

Luigi Galotto Junior, Dr.

Dedico este trabalho aos meus avós maternos, “In Memorian”, fonte de inspiração na engenharia e na vida.

AGRADECIMENTOS

Agradeço primeiramente ao meu orientador, Prof. Dr. Raymundo Cordero García, pela paciência e orientação.

Ao Prof. Dr. João Onofre Pinto, pelo apoio e incentivo.

Ao Laboratório BATLAB e sua equipe, o qual forneceu o material necessário para a realização deste trabalho.

Aos meus colegas Felipe e Thyago os quais foram meus colegas no laboratório e ajudaram em momentos de necessidade.

RESUMO

Este trabalho de Dissertação de Mestrado visa a implementação em *FPGA* (*Field Programmable Gate Array*, em inglês) dos seguintes algoritmos usados no controle de motores trifásicos: A técnica de modulação por vetores espaciais *SVPWM* (*Space Vector Pulse Widht Modulation*, em inglês) e o algoritmo de leitura do sensor de posição resolver. Estes algoritmos serão implementados utilizando Redes Neurais Artificiais (RNA). A aplicação de RNAs executados em *FPGA* permite um processamento rápido, paralelo e não linear dos dados. Desta maneira, podem-se desenvolver sistemas de controle e estimação com um maior desempenho e robustez em relação ao uso de técnicas lineares. Um dos problemas mais destacáveis da implementação de RNAs em um *FPGA* é o cálculo da função de ativação não linear, usada geralmente nos neurônios da camada oculta de um perceptron multicamada (*Multi-Layer Perceptron, MLP,* em inglês). Entre as diferentes funções de ativação existentes, a função sigmoide é uma das mais utilizadas e estudadas. Na dissertação proposta, a função de ativação sigmoide será aproximada usando a técnica *SPLINE*, a qual apresenta compromisso entre a exatidão da aproximação e o custo computacional. Nesta técnica, a função a aproximar é dividida em segmentos, e cada segmento é aproximado através de um polinômio. A exatidão da aproximação depende desta segmentação. Um Algoritmo Genético (AG) foi aplicado para procurar uma segmentação ótima da função sigmoide que permita reduzir o erro de aproximação. Resultados de simulação e experimentais mostram que o uso de AG e *SPLINE*s permite uma eficiente implementação de RNAs aplicado à modulação *SVPWM* e para a estimação da posição angular usando o sensor resolver.

Palavras-chave: Algoritmo Genético, *FPGA*, Redes Neurais Artificiais, Sensor Resolver; *SPLINE, SVPWM*.

ABSTRACT

This master's thesis work aims to implement in *FPGA* (Field Programmable Gate Array) the following algorithms used to control three-phase motors: The *SVPWM* (Space Vector Pulse Widht Modulation) modulation technique and the resolver position sensor reading algorithm. These algorithms will be implemented using Artificial Neural Networks (*ANN*). The application of *ANN*s executed in *FPGA* allows a fast, parallel and non-linear processing of the data. In this way, control and estimation systems can be developed with greater performance and robustness in relation to the use of linear techniques. One of the most notable problems of implementing *ANN*s in *FPGA* is the calculation of the nonlinear activation function, generally used in neurons in the hidden layer of a Multi-Layer Perceptron (*MLP*). Among the different activation functions, the sigmoid function is one of the most used and studied. In the proposed dissertation, the sigmoid activation function will be approximated using the *SPLINE* technique, which presents a compromise between the accuracy of the approximation and the computational cost. In this technique, the function to be approximated is divided into segments, and each segment is approximated through a polynomial. The accuracy of the approximation depends on the segmentation. A Genetic Algorithm (*GA*) was applied to obtain a segmentation of the sigmoid function that reduces the approximation error. Simulation and experimental results show that the use of *GA* and *SPLINE*s allows an efficient implementation of *ANN*s applied to *SVPWM* modulation and to estimate the angular position using the resolver sensor.

Keywords: Artificial Neural Network; *FPGA*; Genetic Algorithm; Resolver Sensor; *SPLINE*; *SVPWM*.

LISTA DE FIGURAS

LISTA DE TABELAS

SUMÁRIO

1 INTRODUÇÃO

1.1 Contextualização do Problema

O uso de veículos elétricos (VE) visa melhorar a qualidade do ar em meios urbanos, diminuir o consumo de reservas petrolíferas e reduzir o impacto causado pelas mudanças climáticas. Não obstante, o desenvolvimento do sistema de controle dos motores elétricos utilizados na propulsão ainda representa um desafio técnico (SAIDI, TALEB, *et al.*, 2015). O sistema de controle deve ser robusto e reagir com a velocidade suficiente para atingir as especificações de controle, principalmente a rejeição de ruído e perturbações. Para atingir tal objetivo, os sistemas de leitura de velocidade, de instrumentação e o algoritmo de controle do motor devem ter boa resposta dinâmica e robustez.

Apesar do desenvolvimento de técnicas de controle de motores sem sensores, a indústria de fabricação de veículos elétricos ainda utiliza sensores de posição para implementar sistemas de controle de torque e/ou velocidade robustos, como no caso do Toyota *Prius* (KITAZAWA, 2006). Para a escolha adequada do sensor de posição, uma série de fatores deve ser levado em conta: ser absoluto ou incremental, magnético ou ótico, o tamanho, precisão, montagem, interface, dentre outros.

No sensor de posição encoder tipo incremental, o ângulo é obtido com base na contagem de eventos de acordo com o disco e o sensor aplicado, o qual pode ser ótico ou magnético. No encoder ótico, a luz é gerada por um LED que atravessa o disco feito de vidro, metal ou plástico, e detectada pelo foto-receptor a cada intervalo de ângulo. Como resultado, uma onda quadrada na saída é gerada para cada ocorrência do evento. Além disso, um ou mais sinais defasados podem ser gerados para determinar o sentido de rotação. Por outro lado, para o sensor encoder do tipo magnético, o disco apresenta vários polos, a variação senoidal do campo magnético em um sinal elétrico o qual dará origem a onda quadrada. No sensor encoder absoluto, o disco é codificado em um padrão binário que não se repete dentro da volta, possibilitando saber o ângulo exato (DYNAPAR, 2018).

O resolver é mais utilizado em veículos elétricos e híbridos (KAEWJINDA e KONGHIRUN, 2006). Sua principal diferença está em produzir sinais analógicos (no lugar de um sinal digital) e não apresentar circuito ótico. O sensor resolver pode ser modelado como transformador rotativo com dois enrolamentos secundários defasados em 90 graus mecânicos. Em decorrência de sua estrutura, o resolver apresenta maior capacidade de choque e vibração,

é mais robusto comparado aos encoders, pode ser usado em ambientes com alta radiação e sua resolução é teoricamente infinita (DYNAPAR, 2018). A tabela 1.1 apresenta a comparação entre os diferentes tipos de sensor de posição.

Tabela 1.1 - Especificações técnicas para diferentes sensores de posição.

Dispositivo	**Resolução**	**Melhor Precisão**	**IP**	**Choque**	**Vibração**
Encoder Incremental Ótico	10000 PPR	Até 2,5 arc-min	IP67	50g[1]	20g[1]
Encoder Incremental Magnético	2048 PPR	NA[2]	IP67	30g	18g
Encoder Absoluto Ótico	2^{22} CPR	36 arc-sec	IP67	100g	10g
Encoder Absoluto Magnético	4096 CPR	0,6º	IP68k	200g	20g
Resolver	∞	2 arc-min	IP65[3]	200g[3]	40g

Fonte: Dynapar, 2018.

[1] Baseado no Northstar HD25.
[2] Precisão do encoder incremental magnético depende da instalação.
[3] Instalado.
Todos os valores são baseados na linha de produtos de feedback Dynapar

Embora os motores de corrente contínua (CC) sejam mais simples de se controlar, os motores trifásicos são os mais utilizados nos veículos elétricos, devido a seu baixo volume e peso, alta velocidade e torque além de oferecer menos manutenção. A Tabela 1.2 mostra os motores trifásicos utilizados nos principais veículos elétricos e híbridos.

Tabela 1.2 - Modelos comerciais de veículos elétricos e hibrido-elétrico.

Tipo	**Veículo**	**Característica(s) do(s) Motor(es) no Veículo**
Veículo Hibrido-elétrico (VHE)	Honda Insight	*Brushless* CC, 9,7 kW @ 1500 RPM
	Ford Fusion Hybrid	MSIP[1], 79 kW @ 6500 RPM
	Toyota Prius	MSIP[1] 1, 30 kW @ 1800 RPM MSIP[1] 2, 50 kW @ 1200 RPM
	GM Volt	MSIP[1] 1 (motor) 111 kW MSIP[1] 2 (motor/gerador) 55kW
	Hyundai Sonata	MSIP[1], 30 kW
Veículo Elétrico (VE)	Mitsubishi iMiEV	MSIP[1], 47 kW
	Renault Fluence	MSIP[1], 7 kW
	BMW Active	MSIP[1], 125 kW
	Ford Focus BEV	MSIP[1], 107 kW
	Tesla Roadstar	Motor de indução, 215 kW

Fonte: GARCÍA, 2015.

[1]MSIP – Motor Síncrono de Imã Permanente.

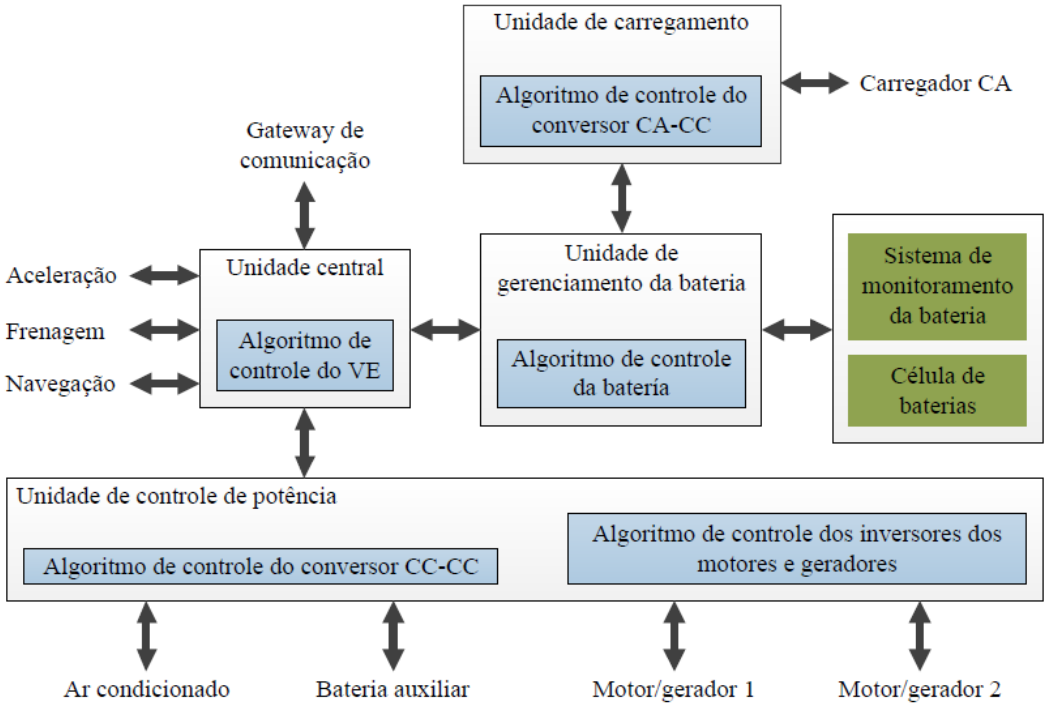
A fonte de energia elétrica utilizada em veículos elétricos geralmente é de corrente contínua (exemplo, baterias). Portanto, para alimentar os motores trifásicos, é necessário usar um inversor para fornecer energia em tensão alternada (CA) ao motor (SAIDI, *et al.*, 2015). Assim, uma técnica de modulação (*Pulse Width Modulation*, *PWM,* em inglês) é necessária para gerar os pulsos digitais permitam produzir as tensões de alimentação desejadas nas saídas do inversor.

Uma das técnicas de modulação mais populares é a modulação por vetores espaciais ou *SVPWM* (*Space Vector Pulse Width Modulation,* em inglês). Esta técnica está baseada na representação de tensões trifásicas através de um vetor espacial definido em um sistema de referência ortogonal. A técnica *SVPWM* possui melhor desempenho harmônico e apresenta resposta satisfatória para as regiões de sobremodulação comparado a outras técnicas *PWM*, em decorrência da liberdade de posicionamento do vetor de referência para o ciclo de chaveamento. (KUMAR e DAS, 2014). Além disso, a técnica *SVPWM* permite uma melhor utilização do barramento CC do inversor quando comparada com a modulação senoidal. Além disso, em *SVPWM*, as três tensões trifásicas são abordadas como um todo, ao em vez de trata-las independentemente (GARCÍA, 2015).

Microcontroladores, processadores digitais de sinais (*Digital Signal Processor, DSP*, em ingl) e *FPGA* (*Field Programmable Gate Array*, em inglês) são sistemas digitais utilizados na implementação de controladores de motores e outros dispositivos de eletrônica de potência. Entre os diferentes tipos de processadores digitais, o *FPGA* será utilizado nesta Dissertação pela sua maior velocidade de processamento em relação aos microcontroladores e *DSP*s, e pela sua arquitetura flexível e paralela (GARCÍA, 2015), (ALOUANE, *et al.*, 2018). A capacidade de processamento paralelo do *FPGA* permite o controle simultâneo dos diferentes sistemas que compõem um veículo elétrico, como será mostrado posteriormente.

Os veículos elétricos e híbridos requerem o controle simultâneo de mais de um motor: motor de propulsão, motor de arranque, etc. Ademais, devem-se controlar outros sistemas como o controle de consumo de combustível, sistema de frenagem, sistema de baterias, entre outros. No caso do controle de motores, procuram-se maiores taxas de aquisição de dados, maiores frequências de chaveamento, e melhores algoritmos de controle. Tudo isto exige o uso de processadores digitais com maior capacidade de cálculo. A Figura 1.1 mostra os diferentes sistemas que devem ser controlados em um veículo elétrico. Anteriormente, cada algoritmo era executado por um controlador digital diferente (GARCÍA, 2015).

Figura 1.1 - Principais sistemas de um veículo elétrico.



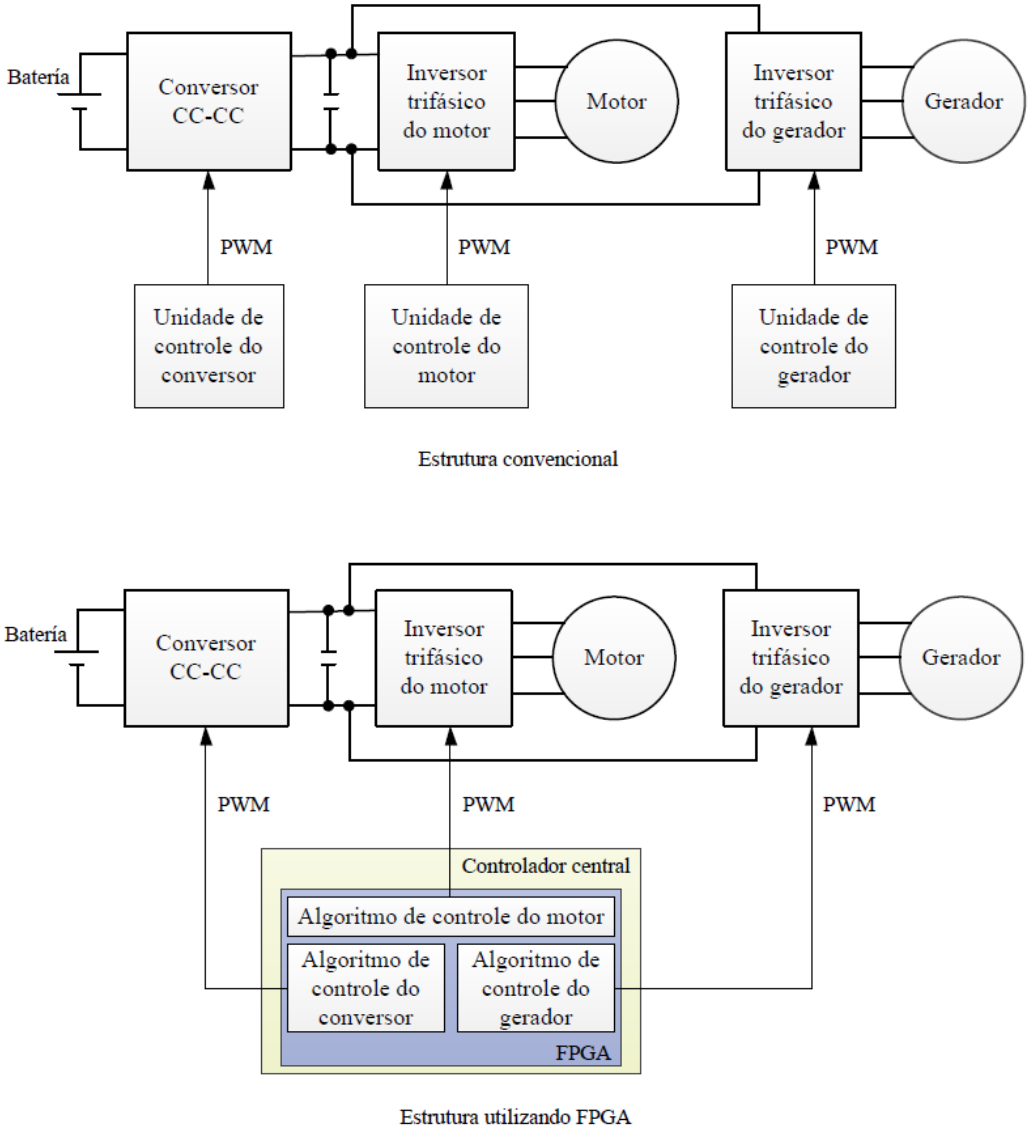
Fonte: GARCÍA, 2015

Neste sentido, o *FPGA* está ganhando aceitação no projeto de controladores de velocidade de motores elétricos pela sua velocidade de processamento, processamento paralelo, flexibilidade e pela existência de ferramentas que simplificam a programação e verificação do *FPGA* (ALTERA, 2019) (MONMASSON, IDKHAJINE, *et al.*, 2011). Entre os benefícios de utilizar um *FPGA* na fabricação de veículos elétricos e híbridos, têm-se:

- Simplificação da arquitetura de controle. Em decorrência da capacidade de processamento paralelo torna-se possível executar simultaneamente muitos algoritmos de controle e processamento de dados.
- Maior velocidade de processamento em comparação com dispositivos DSP e micro controladores.
- Uso de conversores de potência de alta frequência. Como no caso de inversores onde a alta velocidade de processamento permite a implementação de sistemas de controle em altas taxas de amostragem.

A Figura 1.2 mostra uma comparação do sistema de controle para uma arquitetura típica de um veículo elétrico que usa um conversor CC-CC, um motor e um gerador trifásico (ALTERA, 2013). Convencionalmente, cada um destes elementos é controlado por uma unidade de controle independente, enquanto o *FPGA* permite executar paralelamente muitas funções de controle no mesmo dispositivo. Isto reduz o número de controladores e interfaces de *hardware*. Além disso, é possível executar facilmente simulações em *software* e emulações para verificar o bom funcionamento do controlador.

Figura 1.2 - Estrutura de controle de um VE utilizando um FPGA.



Fonte: GARCÍA, 2015

Outro fator importante a considerar na escolha de processadores digitais para o controle de veículos elétricos e híbridos é a tendência de adicionar sistemas de segurança inteligentes nos veículos (ALTERA, 2013) (TAO, 2014): frenagem eletrônica automática, câmeras de vídeo com processamento de imagens para a detecção de pedestres e obstáculos, entre outros. Estes sistemas envolvem uma grande quantidade de operações matemáticas, cujos resultados devem ser disponibilizados em tempo real. O *FPGA* constitui uma plataforma ótima para a execução dos algoritmos necessários para tais sistemas adicionais.

Os motores trifásicos são sistemas não lineares. Portanto, reguladores lineares tipo PID não têm um bom desempenho para diferentes pontos de operação. Assim, sistemas de controle não lineares foram propostos na literatura. Entre as diferentes técnicas de controle não linear, as Redes Neurais Artificiais (RNAs) são muito utilizadas pela habilidade de processamento

paralelo, velocidade de resposta, capacidade de aprendizado, robustez e resposta não linear (KUMAR e DAS, 2014).

A estrutura de uma RNA permite o processamento paralelo de dados. Por outro lado, o *FPGA* tem uma arquitetura paralela flexível que permite a execução de algoritmos simultaneamente (processamento paralelo de informação). Portanto, o *FPGA* é um processador digital que permite o melhor aproveitamento das capacidades de processamento de uma RNA. Existem diversos trabalhos sobre a implementação de RNAs em *FPGA* (CULLEN, GERBETH e DOROJEVETS, 2018), (RAMIREZ, PROAÑO e DARWIN, 2015), (ZAPATÁN, LÚCERO, *et al.*, 2015).

Não obstante, a implementação de RNAs em *FPGA* requer a solução de determinados problemas não triviais. Um dos problemas mais difíceis é o cálculo da função de ativação (geralmente não linear) do neurônio artificial. Por exemplo, a função sigmoide, uma das funções de ativação mais utilizadas, requer o cálculo de exponenciais. Esta função possui alto custo computacional. Geralmente, técnicas de aproximação são utilizadas para calcular as funções de ativação em processadores digitais Entre tais técnicas, têm-se a *Look-Up Table* (*LUT*), séries de Taylor, *Piece-Wise Linear* (*PWL*) e *SPLINE*. A série de Taylor fornece alta exatidão de aproximação, mas requer muitas operações matemáticas. A *LUT* requer uma grande quantidade de memória (para armazenar valores discretos da função a aproximar) e de um algoritmo de interpolação. Na técnica *PWL*, a função a aproximar é dividida em uma grande quantidade de segmentos, e cada segmento é aproximado por uma função linear (ARMATO, FANUCCI, *et al.*, 2009). Esta técnica é simples, mas requer muitos intervalos e o erro de aproximação é relativamente grande.

Na dissertação proposta foi escolhido utilizar aproximação usando *SPLINE*. Esta técnica consiste na segmentação da função de ativação sigmoide em intervalos delimitados por nós (segmentação), e cada intervalo da função é aproximado usando um polinômio de grau 2 ou superior. O *SPLINE* fornece um bom compromisso entre exatidão de aproximação e custo computacional: tem um custo computacional menor que a Série de Taylor, e o erro de aproximação e número de segmentos é menor comparado ao *PWL*. A exatidão do *SPLINE* depende da ordem do polinômio e da maneira como a função foi segmentada. Convencionalmente, a segmentação é efetuada arbitrariamente (SOARES, 2006).

1.2 Justificativa do Tema de Estudo

O uso de *FPGA* possibilita a centralização do controle em um único dispositivo com processamento em paralelo e arquitetura flexível. Além disso, o *FPGA* possui uma maior velocidade de processamento que os *DSPs* e microcontroladores. Da mesma forma, controladores baseados em RNAs possuem estrutura de processamento em paralelo, capacidade de interpolação/extrapolação, robustez, predição e processamento não linear (GARCÍA, 2015). Portanto, o *FPGA* permite ter acesso a todas as vantagens que as RNAs oferecem no desenvolvimento de controladores e estimadores para veículos elétricos.

Realizar a implementação de redes neurais artificiais em sistemas embarcados *FPGA* é um trabalho não trivial. Técnicas convencionais como uso de *LUT* e para a representação de funções de ativação demandam uso de memória interna. Outro desafio é a minimização do tempo de execução, tendo em vista a manutenção da característica de processamento paralelo de uma RNA e a escolha adequada do tamanho das variáveis binárias na arquitetura do processo.

A motivação deste trabalho visa superar estas barreiras ao propor o uso da aproximação para funções de ativação por meio da técnica *SPLINE*. Refinada com auxílio de Algoritmos Genéticos para obtenção de uma segmentação otimizada a reduzir o erro de estimação.

A aplicação de Algoritmos Genéticos se justifica em manter um número pequeno de segmentos na técnica *SPLINE* mantendo um erro de aproximação satisfatório por poupar tempo de processamento. Porém, o grau dos polinômios adotados é um fator crucial na implementação, pois embora ocorra a redução no erro, as constantes demandam variáveis com maior quantidade de bits para serem representadas e, consequentemente, maior uso de memória interna.

Conforme era realizado a implementação em *FPGA*, problemas foram surgindo, como a quantidade de bits usada nas variáveis que impactam no resultado. Embora as simulações ajudem a determinar o tamanho binário requerido, ajustes manuais ainda se fizeram necessários para a resolução das inconsistências.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo geral da presente Dissertação de mestrado é o desenvolvimento em *FPGA* dos algoritmos de modulação *SVPWM* e o algoritmo de leitura para sensor de posição angular tipo resolver, usando RNAs. A função de ativação sigmoide será aproximada por meio da técnica *SPLINE*, sendo que um Algoritmo Genético será aplicado para efetuar uma segmentação da função a qual minimize o erro de aproximação.

1.3.2 Objetivos Específicos

1. Compreensão sobre algoritmos de modulação por vetores espaciais e o sensor resolver.
2. Aproximação e otimização de função de ativação sigmoide com uso de *SPLINE* e algoritmo genético.
3. Implementação da aproximação para a função sigmoide.
4. Implementação de Redes Neurais em *FPGA.*
5. Implementação da técnica *SVPWM* em *FPGA* usando RNAs.
6. Implementação de algoritmo de leitura do sensor resolver usando RNAs

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Redes Neurais Artificiais

Redes Neurais Artificiais (RNAs) é uma área de estudo inspirada na neurologia, onde unidades simples chamadas de neurônios se conectam para formar uma rede complexa. Uma das motivações no desenvolvimento de RNA é mimetizar a alta capacidade de processamento paralelo presente em sistemas neurais biológicos (MITCHELL, 1997).

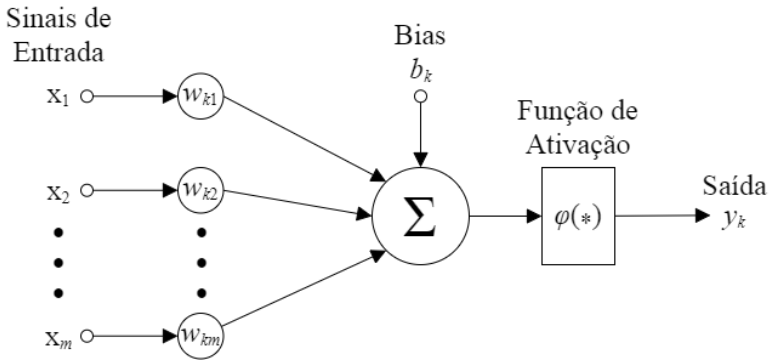
O cérebro biológico possui a capacidade de criar suas próprias regras por meio de experiências desenvolvidas ao longo dos anos. De forma análoga, em redes neurais o processo de aprendizagem ocorre através de algoritmos, cuja função é alterar o peso das sinapses da rede com o objetivo de atingir o resultado desejado. (HAYKIN, 2009), ALEKSANDER e MORTON, 1990, definem as Redes Neurais Artificiais da seguinte forma:

> Uma Rede Neural é um processador paralelo distribuído massivamente constituído por unidades de processamento simples, que tem uma propensão natural para armazenar conhecimento experimental e disponibiliza-lo para uso. Assemelha-se ao cérebro em dois aspectos:
>
> 1. Conhecimento é adquirido pela rede a partir de seu ambiente em um processo de aprendizagem.
> 2. As forças de interconexão neural, conhecidas como pesos sinápticos, são usadas para armazenar o conhecimento adquirido.

Redes Neurais Artificiais possuem a habilidade de generalização, que consiste em reproduzir resultados não utilizados no processo de aprendizagem. Uma RNA não possui a capacidade de obter resultados trabalhando individualmente, devendo estar integrada em um sistema consistente que condiz com suas capacidades inertes (HAYKIN, 2009) .

A Figura 2.1 apresenta o modelo de um neurônio de uma rede neural. Os elementos observados são: um conjunto de pesos sinápticos w_{kj}; um *bias* b_k; um somador que combina os valores de sinapse modificados pelos pesos e o *bias*; uma função de ativação $\varphi(*)$ cujo objetivo é normalizar a resposta y_k do neurônio em torno de determinada faixa de valores.

Figura 2.1 - Modelo de um neurônio.



Fonte: Autor

O neurônio pode ser representado matematicamente, Onde u_k corresponde ao somatório das entradas x_1, x_2 ... x_m multiplicada pelos pesos w_{k1}, w_{k2} ... w_{km} do k-esimo neurônio (2.1). v_k é obtido com a adição do *bias* em u_k (2.2) o qual é normalizado através da função de ativação resultando na saída do neurônio y_k (2.3). (HAYKIN, 2009).

$$u_k = \sum_{j=1}^{m} w_{kj} x_j \tag{2.1}$$

$$v_k = (u_k + b_k) \tag{2.2}$$

$$y_k = \varphi(v_k) \tag{2.3}$$

Segundo HAYKIN, 2009, redes neurais artificiais em geral possuem três classes de arquitetura:

1. Redes de camada única: Cada saída é constituída por um único neurônio.
2. Redes com múltiplas camadas: Apresentam uma ou mais camadas ocultas, fornecendo à rede maior capacidade de análise de dados.
3. Redes recorrentes: Apresentam realimentação de valores prévios da saída, geralmente usadas quando há necessidade de representar evolução temporal.

2.1.1 Função de Ativação

Os tipos básicos de função de ativação normalizados entre 0 e 1 são: a função degrau (2.4); À função linear (2.5); e a função sigmoide (2.6), a qual é a mais usada na construção de arquiteturas de Redes Neurais Artificiais. Na equação (2.6) a constante α é usada para ajustar a curva da função sigmoide. No desenvolvimento de técnicas da aproximação da função sigmoide, considera-se α = 1, sendo que a entrada da função de ativação depende dos pesos sinápticos

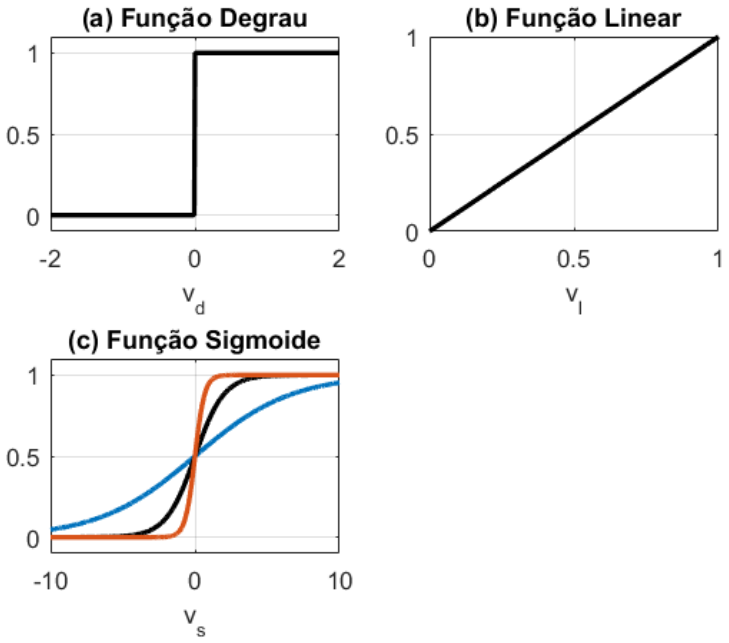
que atuam como ganhos. A Figura 2.2 apresenta os gráficos para os diferentes tipos de função de ativação.

$$\text{Função degrau: } \varphi(v_d) = \begin{cases} 0 \; se \; v_d < 0 \\ 1 \; se \; v_d \geq 0 \end{cases} \tag{2.4}$$

$$\text{Função linear: } \varphi(v_l) = v_l \tag{2.5}$$

$$\text{Função sigmoide: } \varphi(v_s) = \frac{1}{1+\exp(-\alpha v_s)} \tag{2.6}$$

Figura 2.2 - Funções de ativação: (a) Degrau; (b) Linear; (c) Sigmoide.



Fonte: Autor.

Para normalizar a resposta entre –1 e 1, a simetria da função é deslocada para a origem. Na função degrau a relação é ajustada conforme (2.7). Por outro lado, a função tangente hiperbólica, descrita em (2.8), pode substituir a função sigmoide (HAYKIN, 2009).

$$\varphi(v_d) = \begin{cases} -1 \; se \; v_d < 0 \\ 1 \; se \; v_d \geq 0 \end{cases} \tag{2.7}$$

$$\varphi(v_h) = \tanh(v_h) \tag{2.8}$$

2.1.2 Redes Neurais de Múltiplas Camadas

Redes neurais com múltiplas camadas possuem a habilidade de expressar uma rica variedade de respostas não lineares. Um exemplo é o reconhecimento e sintetização de fonemas e sílabas, ao representar a entrada e saída de forma numérica por meio da análise espectral do som. Para isso é necessária uma função onde a saída seja uma combinação não linear diferenciável das suas entradas, como ocorre ao utilizar a sigmoide como função de ativação no neurônio (MITCHELL, 1997).

HAYKIN, 2009, delimita três características para uma rede neural perceptron multicamada (*Multi-Layer Perceptron, MLP*, em inglês):

1. Cada neurônio presente nas camadas ocultas da rede inclui uma função de ativação não linear e diferenciável em todos os seus pontos.
2. A rede possui uma ou mais camadas ocultas que não fazem parte da entrada ou saída do sistema.
3. Uma mudança na conectividade da rede requer uma alteração na população de pesos sinápticos.

O uso de camadas ocultas torna o processo de aprendizagem da rede complexa de ser visualizado, enquanto que a presença da distribuição de não linearidade e alta conectividade torna a análise teórica de uma rede multicamada difícil de ser interpretada.

Para o treinamento da rede multicamada, considere o erro absoluto para o treinamento para o neurônio *k* na interação *n*, conforme a equação (2.9):

$$e_k(n) = t_{k(n)} - y_k(n) \tag{2.9}$$

Onde:

- $t_{k(n)}$:valor objetivo (*target*) do treinamento do neurônio *k* na interação *n*.
- $y_k(n)$:resposta do neurônio *k* na interação *n*.

Estendendo para todos os neurônios na camada de saída da rede "C", a equação (2.9) assume a seguinte forma:

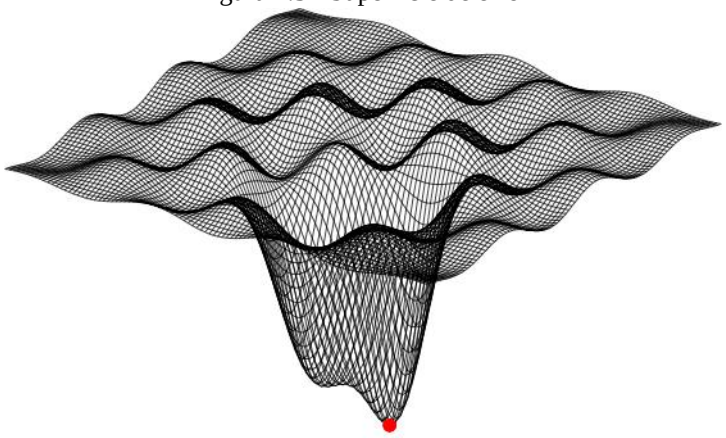
$$E(n) = \frac{1}{2}\sum_{k \in C} e_k^2(n) \tag{2.10}$$

Seja *M* o número total de configurações possíveis contidos no conjunto de treinamento. A função custo é definida de acordo com a equação (2.11).

$$E_{med} = \frac{1}{M}\sum_{n=1}^{M} E(n) \tag{2.11}$$

O objetivo do treinamento consiste em minimizar a função custo ajustando os parâmetros da rede neural (HAYKIN, 2009). A função custo pode ser representada como uma superfície de erro (MITCHELL, 1997), conforme mostrado pela Figura 2.3.

Figura 2.3 - Superfície de erro



Fonte: Bahl, 2012.

2.1.3 Algoritmo de treinamento *Backpropagation*

O problema enfrentado durante o aprendizado de uma rede neural de múltiplas camadas consiste em um grande campo de busca para a obtenção dos pesos. O gradiente de erro também representa uma dificuldade. Isto implica na possibilidade de existência de múltiplos mínimos locais os quais, não necessariamente, representam o menor erro global. Apesar dos obstáculos, o *Backpropagation* apresenta bons resultados em aplicações reais. (HAYKIN, 2009)

O aprendizado por *Backpropagation* consiste em duas fases: o passo para frente (*forward pass*) e o passo para trás (*backward pass*). Na primeira fase (*forward pass*), os pesos da rede são fixados e um conjunto de valores de treinamento são aplicados resultando em uma saída. Na segunda fase, é calculado o erro e os pesos são ajustados partindo da camada de saída em direção à camada de entrada. (MITCHELL, 1997).

De forma resumida, MITCHEL, 1997 descreve os passos do algoritmo de treinamento *Backpropagation* da seguinte forma:

1. Inicialização: É escolhido um conjunto de pesos iniciais randômicos e normalizada a entrada da rede.
2. Apresentação do exemplo de treinamento: A rede neural é alimentada com o conjunto de valores de treinamento.
3. Computação para frente: Os valores de treinamento introduzidos na rede navegam camada por camada onde é computado a resposta dos neurônios até a obtenção do sinal de saída e cálculo do erro.
4. Computação para trás: Computa o gradiente do erro para a rede de acordo com as equações (2.12). Os pesos sinápticos então são ajustados de acordo com a equação (2.13) onde η é o parâmetro de aprendizado e α é a constante de momento.

$$\delta_k^l(n) = e_k^L(n)\varphi_k'\left(v_k^{(L)}(n)\right),\ para\ o\ neurônio\ k\ na\ camada\ de\ saída\ L \quad (2.12.a)$$

$$\delta_k^l(n) = \varphi_k'\left(v_k^{(l)}(n)\right)\sum_j \delta_j^{(l+1)}(n)w_{jk}^{(l+1)}(n),\ para\ o\ neurônio\ k\ na\ camada\ oculta\ l \quad (2.12.b)$$

$$w_{kj}^{(l)}(n+1) = w_{kj}^{(l)}(n) + \alpha\left[w_{kj}^{(l)}(n-1)\right] + \eta\delta_k^{(l)}(n)y_i^{(l-1)}(n) \quad (2.13)$$

5. Interação: Os passos 3 e 4 são repetidos a cada nova época com novos valores de treinamento até que os critérios de parada sejam atendidos.

2.2 Estimação da Função Sigmoide usando SPLINE

A função de ativação sigmoide não pode ser implementada diretamente em *FPGA* porque é necessário calcular uma função exponencial, tal como é indicado na equação 2.6. Portanto, é necessário o uso de alguma técnica de aproximação para implementar a função sigmoide no *FPGA*, como por exemplo, *Look-Up Table*, Series de Taylor e *SPLINE*.

Look-Up Table (*LUT*) é uma técnica que substitui operações matemáticas por tabelas pré-calculadas armazenadas em memória. Segundo McNamee, 1998, resgatar valores diretamente da memória em muitos casos requer menos custo computacional do que executar a operação matemática diretamente.

Na Série de Taylor, a função é representada por uma soma infinita de termos calculados pela derivada da função em um ponto, conforme (2.14). Para uma aproximação de ordem *n*, a Série de Taylor resulta na aproximação da função (GREENBERG, 1998).

$$f(x) \approx f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots + \frac{f^{(n-1)}(a)}{(n-1)!}(x-a)^{n-1} \qquad (2.14)$$

A aproximação por polinômios *SPLINE* consiste na segmentação da função a ser aproximada em nós de aproximação. Polinômios são utilizados em cada seguimento para representar a função naquele intervalo. Em comparação a série de Taylor, a aproximação usando *SPLINE* é calculada por polinomiais de menor ordem. Por outro lado, a técnica de aproximação *SPLINE* oferece um bom compromisso entre exatidão de estimação e custo computacional.

Para a função $u(x)$ onde $u \in [a,b]$, o qual $a\ e\ b$ são números reais e n, j números naturais, teremos nós de interpolação do tipo $x_o = a, ; x_{j+1} = x_j + h_j;\ x_n = b$. Supondo $j = [0,1,\dots,n]$, $\tilde{u}(x)$ é a aproximação para $u(x)$ nos intervalos $\left[x_j, x_{j+1}\right] \subset [a,b]$, de acordo com a seguinte resolução (BUROVA e VARTANOVA, 2017):

$$\tilde{u}(x) = u(x_j)w_{j,0}(x) + u(x_{j+1})w_{j+1,0}(x) + u'(x_j)w_{j,1}(x) + \cdots$$
$$\cdots + u1(x_{j+1})w_{j+1,1}(x) + \int_{x_j}^{x_{j+1}} u(\xi)d\xi w_j^{<0>}(x) \qquad (2.15)$$

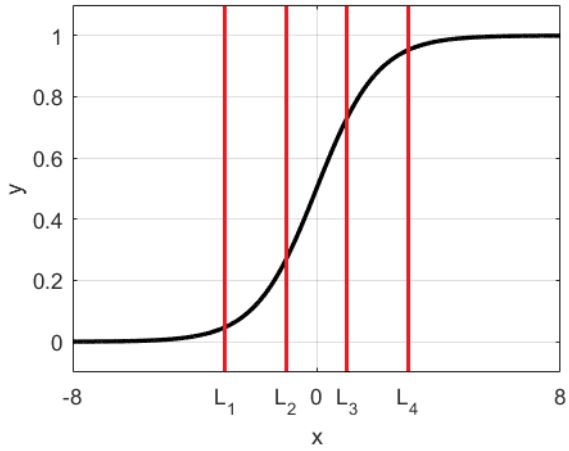
Onde as *SPLINEs* básicas do sistema são $w_{j,0}(x), w_{j+1,0}(x), w_{j,1}(x), w_{j+1,1}(x), w_j^{<0>}(x)$.

Assumindo que os valores de integral e derivada são conhecidos e não possuem erro de arredondamento ou de leitura, a equação (2.15) pode ser reescrita para um intervalo pertencente a função:

$$P_j = C_n x^n + C_{n-1}x^{n-1} + \cdots + C_1 x + C_0 \qquad (2.16)$$

As constantes C_n são obtidas por meio de regressão polinomial com referência na curva original para intervalos de estimação entre os nós. Este trabalho realiza a aproximação da função sigmoide de acordo com a Figura 2.4, com sete intervalos de estimação definidos pelos nós $-8, L_1, L_2, L_3, L_4, 8$.

Figura 2.4 - Intervalos de aproximação para a função Sigmoide.



Fonte: Autor.

Nos seguimentos $[-\infty, -8]$ e $[8, \infty]$ a função sigmoide é práticamente constante (acontece uma saturação). Assim a função pode ser simplificada conforme descrito abaixo (SOARES, 2006):

$$\tilde{u}(x) = \begin{cases} 0, & x = [-\infty, -8] \\ 1, & x = [8, \infty] \end{cases} \tag{2.17}$$

A consideração proposta em (2.17) reduz a estimação para polinômios nos intervalos entre $[-8, 8]$. Seja *x* um número positivo. Como a função sigmoide é simétrica em relação ao ponto (0; 0,5), o valor de $u(-x)$ pode ser obtido a partir de $u(x)$:

$$u(-x) = 1 - u(x), \ x \geq 0 \tag{2.18}$$

Substituindo o valor da função pela aproximação, tem-se:

$$\tilde{u}(-x) = 1 - \tilde{u}(x), \ x \geq 0 \tag{2.19}$$

A equação (2.19) indica que só é necessário obter as aproximações para os valores de entrada positivos da função, o que reduz a quantidade de parâmetros a armazenar quando a técnica *SPLINE* é utilizada para aproximar a sigmoide.

2.3 Algoritmo Genético

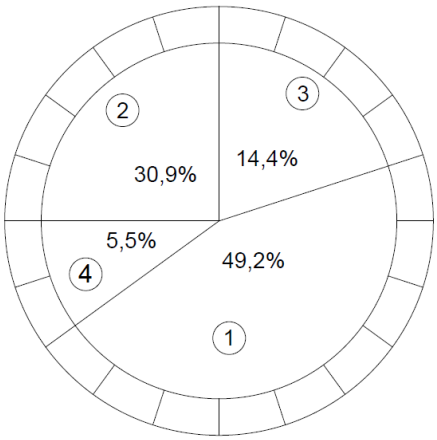
Algoritmo genético (AG) consiste em uma técnica de otimização inspirada na evolução natural. Esta técnica é usada para procurar valores ótimos para um sistema, por exemplo, na determinação de parâmetros de uma rede neural ou na sua topologia. AG é uma técnica populacional: Usa-se uma população composta por indivíduos, sendo que cada indivíduo é uma possível solução. Sua estrutura inicia com uma população inicial que dará origem à próxima geração por meio de mutações e cruzamentos aleatórios (MITCHELL, 1997). O objetivo de um AG é procurar uma solução que minimize uma função de custo que caracteriza o desempenho da solução (*fitness*, em inglês).

Para cada geração, a informação dos indivíduos mais aptos pertencentes a população é passada adiante (GOLDBERG, 1989). O tamanho assim como o significado dos indivíduos pertencentes à população é atribuído pelo pesquisador, a exemplo dos pesos, *bias* e estrutura de uma Rede Neural. (BANZHAF, NORDIN, *et al.*, 1998)

A composição de um algoritmo genético é dividida entre seleção, cruzamento e mutação. No processo de seleção, a população é reordenada de acordo com a função objetivo, também chamada de *fitness*. Indivíduos com melhores valores de *fitness* possuem prioridade com relação aos demais para se tornar as bases (“pais”) da seguinte geração de soluções. Esta etapa representa, de forma artificial, a seleção natural do mais apto.

A seleção dos indivíduos como pais pode ser efetuada de diversas formas. Uma delas consiste em a técnica da roleta: quanto melhor (menor) é o valor do *fitness* do indivíduo, maior é sua possibilidade de ser escolhido como pai. A técnica da roleta é ilustrada na Figura 2.5 (GOLDBERG, 1989).

Figura 2.5 - Exemplo de roleta para seleção de pares.

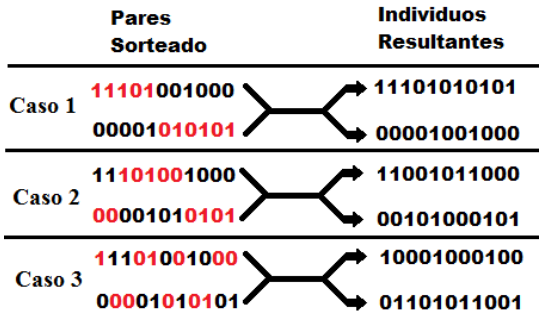


Fonte: Autor.

No cruzamento (*crossover*, em inglês), os pais são agrupados em pares são selecionados aleatoriamente de forma probabilística com base na taxa de cruzamento definida. Assim dois novos indivíduos (filhos) são gerados, combinando informações de cada pai.

A Figura 2.6 apresenta formas distintas de se realizar o *crossover*. No primeiro caso a sequência de *bits* inicial do primeiro pai é combinada com a sequência final de bits do segundo pai a partir de um ponto escolhido aleatoriamente. Para o segundo caso, a combinação ocorre para uma sequência intermediária de *bits* definidos por dois pontos aleatórios. No último caso os *bits* responsáveis por gerar os indivíduos resultantes são totalmente aleatórios. (MITCHELL, 1996).

Figura 2.6 - Operações comuns em pares sorteados.



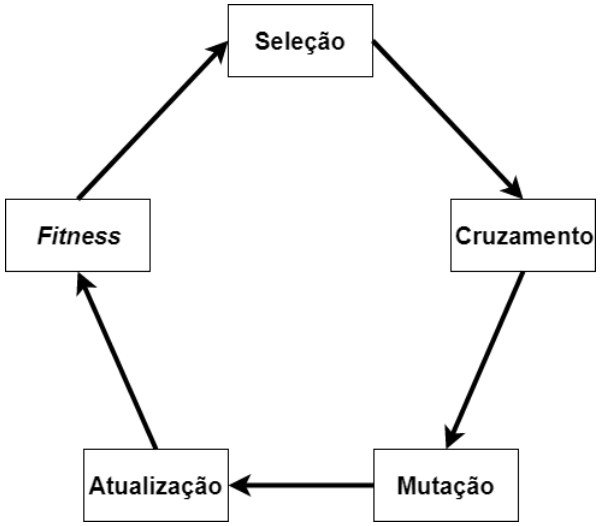
Fonte: Autor.

A mutação normalmente ocorre após o cruzamento. Nesta operação, os bits dos filhos são escolhidos aleatoriamente, segundo uma taxa de mutação, e seu valor é alterado (BANZHAF, NORDIN, *et al.*, 1998). A mutação tem como finalidade prover variabilidade genética a cada nova geração, evitando saturação prematura (GOLDBERG, 1989).

Segue abaixo, de forma resumida, o roteiro de operação de um algoritmo genético, simplificado através do fluxograma da figura 2.6:

1. Seleção: Reordena a população de acordo com seu *fitness*, para escolher os pais que produzirão os novos indivíduos (filhos) da seguinte geração.
2. *Crossover*: Agrupa os pais em pares de forma probabilística com base no valor de *fitness* e realiza o cruzamento entre os pares, gerando novos indivíduos (filhos).
3. Mutação: Altera aleatoriamente com uma baixa probabilidade valores presentes nos novos indivíduos (filhos) da população.
4. Atualização na população, após passar pelas operações anteriores
5. Computa o *fitness* para a população atualizada.
6. Recomeça o ciclo na geração seguinte.

Figura 2.7 - Fluxograma de operação para algoritmo genético.

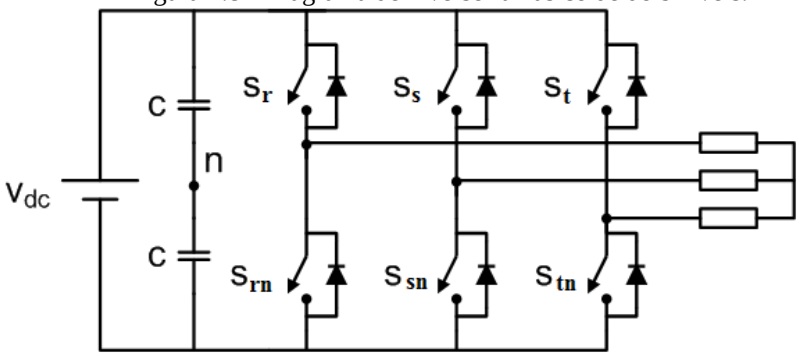


Fonte: Autor.

2.4 *SVPWM*

A Figura 2.8 apresenta o modelo para um inversor trifásico de dois níveis, cada chave de comutação oferece valores de tensão usando o ponto neutro como referência, conforme a equação (2.20) (RASHID, 2001).

Figura 2.8 - Diagrama de inversor trifásico de dois níveis.



Fonte: Autor.

$$V_X = \begin{cases} \frac{V_{dc}}{2},\ S_X = 1\ (ligado) \\ -\frac{V_{dc}}{2},\ S_X = 0\ (desligado) \end{cases} ; X = r, s, t \qquad (2.20)$$

Uma técnica usada para encontrar os estados de chaveamento S_r, S_s e S_t da Figura 2.6 consiste na modulação por largura de pulso *SVPWM* (*Space Vector Pulse Width Modulation*). Nesta técnica, as tensões V_r, V_s e V_t são representadas por meio de um vetor espacial V definido em um plano complexo conjugado, conforme (2.21) (WU, *et al.*, 2008) (IQBAL, *et al.*, 2006).

$$V = \begin{bmatrix} V_{\propto} \\ V_{\beta} \end{bmatrix} = \begin{bmatrix} \frac{2V_r - V_s - V_t}{3} \\ \frac{V_s - V_t}{\sqrt{3}} \end{bmatrix} \tag{2.21}$$

A Tabela 2.1 apresenta os vetores básicos para as possíveis combinações de chaveamento para o inversor de dois níveis apresentado pela Figura 2.7. Os valores $V_{\propto}$ e V_{β} são obtidos com a equação (2.21), conforme o estado de comutação das chaves S_r, S_s e S_t. Observa-se na Tabela 2.1 que V_0 e V_7 são vetores nulos (magnitude zero).

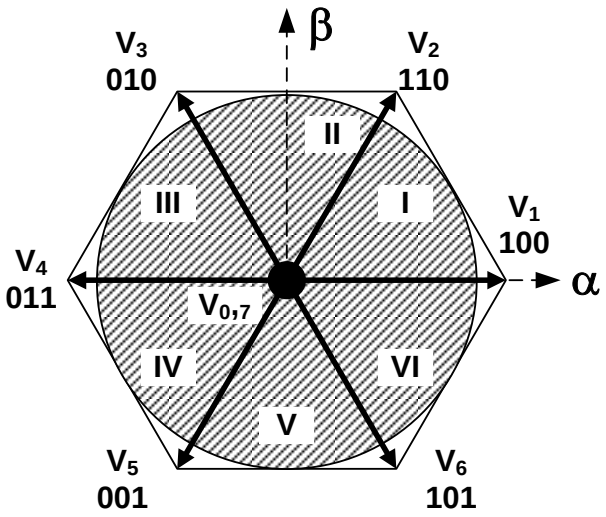
Tabela 2.1 - Vetores espaciais para inversor de dois níveis

$\boldsymbol{V}$	$\boldsymbol{V_{\propto}}$	$\boldsymbol{V_{\beta}}$	$\boldsymbol{S_r}$	$\boldsymbol{S_s}$	$\boldsymbol{S_t}$
$\boldsymbol{V_0}$	0	0	0	0	0
$\boldsymbol{V_1}$	$\frac{2V_{cc}}{3}$	0	1	0	0
$\boldsymbol{V_2}$	$\frac{V_{cc}}{3}$	$\frac{V_{cc}}{\sqrt{3}}$	1	1	0
$\boldsymbol{V_3}$	$-\frac{V_{cc}}{3}$	$\frac{V_{cc}}{\sqrt{3}}$	0	1	0
$\boldsymbol{V_4}$	$-\frac{2V_{cc}}{3}$	0	0	1	1
$\boldsymbol{V_5}$	$-\frac{V_{cc}}{3}$	$-\frac{V_{cc}}{\sqrt{3}}$	0	0	1
$\boldsymbol{V_6}$	$\frac{V_{cc}}{3}$	$-\frac{V_{cc}}{\sqrt{3}}$	1	0	1
$\boldsymbol{V_7}$	0	0	1	1	1

Fonte: (GARCÍA, 2015)

Os vetores básicos dividem o plano complexo em 6 setores de trabalho, conforme a Figura 2.9 (SHARMA, *et al.*, 2017).

Figura 2.9 - Setores de trabalho definidos pelos vetores espaciais não nulos.



Fonte: (GARCÍA, 2015)

Seja V_r o vetor espacial de referência que representa as tensões trifásicas. Este vetor espacial está incluso em um dos setores de trabalho gerados através da disposição dos vetores básicos. O objetivo da técnica *SVPWM* consiste em aproximar V_r mediante a combinação de comutações de estados correspondente aos vetores básicos (SHARMA, BHAT e AHMAD, 2017). A equação (2.22) define como deve ser obtido o vetor de referência para um pequeno período de modulação t_{pwm}.

$$V_r t_{pwm} = V_a t_a + V_b t_b \tag{2.22}$$

As variáveis t_a e t_b são os intervalos de tempo os quais os estados de comutação correspondentes aos vetores básicos V_a e V_b são mantidos no inversor, desta forma tem-se que:

$$t_{pwm} \geq t_a + t_b \tag{2.23}$$

Com o intuito de reduzir o conteúdo harmônico do sinal de tensão, os vetores nulos devem ser utilizados por um período de tempo t_{nulo} de acordo com a equação (2.24).

$$t_{pwm} = t_a + t_b + t_{nulo} \tag{2.24}$$

A partir das equações (2.22) e (2.24), tem-se que:

$$V_r t_{pwm} = V_a t_a + V_b t_b + V_{nulo} t_{nulo} \quad (2.25)$$

Para obter os valores de t_a e t_b a equação (2.24) pode ser reescrita matricialmente em função das componentes reais e imaginárias, conforme a seguinte solução (LIU, *et al.*, 2008):

$$\begin{bmatrix} t_a \\ t_b \end{bmatrix} = t_{pwm}[V_a \quad V_b]^{-1} V_r = t_{pwm} \begin{bmatrix} V_{a\_\propto} & V_{b\_\propto} \\ V_{a\_\beta} & V_{b\_\beta} \end{bmatrix}^{-1} \begin{bmatrix} V_{ref\_\alpha} \\ V_{ref\_\beta} \end{bmatrix} \quad (2.26)$$

A partir da equação (2.26), foram obtidas as fórmulas para os intervalos de tempo t_a e t_b em cada setor de trabalho, de acordo com a Tabela 2.2.

Tabela 2.2 - Intervalos de tempo de comutação.

Setor	$\boldsymbol{t_a}$	$\boldsymbol{t_b}$
I	$\frac{t_{pwm}(3V_{r\_\propto} - \sqrt{3V_{r\_\beta}})}{2V_{cc}}$	$\frac{t_{pwm}\sqrt{3V_{r\_\beta}}}{V_{cc}}$
II	$\frac{t_{pwm}(3V_{r_\propto} + \sqrt{3V_{r\_\beta}})}{2V_{cc}}$	$-\frac{t_{pwm}(3V_{r\_\propto} - \sqrt{3V_{r\_\beta}})}{2V_{cc}}$
III	$\frac{t_{pwm}\sqrt{3V_{r\_\beta}}}{V_{cc}}$	$-\frac{t_{pwm}(3V_{r_\propto} + \sqrt{3V_{r\_\beta}})}{2V_{cc}}$
IV	$-\frac{t_{pwm}(3V_{r\_\propto} - \sqrt{3V_{r\_\beta}})}{2V_{cc}}$	$-\frac{t_{pwm}\sqrt{3V_{r\_\beta}}}{V_{cc}}$
V	$-\frac{t_{pwm}(3V_{r_\propto} + \sqrt{3V_{r\_\beta}})}{2V_{cc}}$	$\frac{t_{pwm}(3V_{r\_\propto} - \sqrt{3V_{r\_\beta}})}{2V_{cc}}$
VI	$-\frac{t_{pwm}\sqrt{3V_{r\_\beta}}}{V_{cc}}$	$\frac{t_{pwm}(3V_{r_\propto} + \sqrt{3V_{r\_\beta}})}{2V_{cc}}$

Fonte: (GARCÍA, 2015)

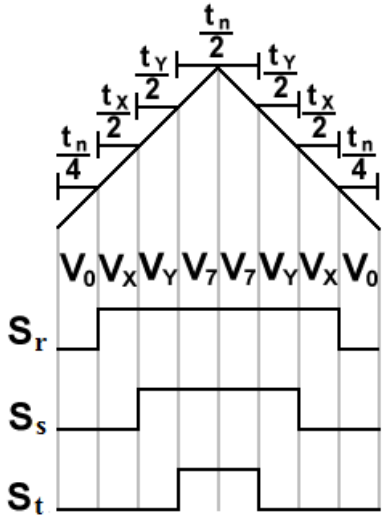
Os vetores V_X e V_Y são obtidos de acordo com a posição no setor *m* do vetor de referência V_r, de acordo com as equações (2.27) e (2.28).

$$V_x = V_m \quad (2.27)$$

$$V_Y = \begin{cases} V_{m+1}, & m = [1,2,3,4,5] \\ V_1, & m = 6 \end{cases} \quad (2.28)$$

A partir das equações (2.25), (2.27) e (2.28) é obtida a disposição dos vetores no tempo, conforme a Figura 2.10.

Figura 2.10 - Disposição dos vetores no tempo.



Fonte: Autor.

Onde t_X e t_Y são obtidos de acordo com as equações (2.29) e (2.30).

$$t_X = \begin{cases} t_a, & m = [1,3,5] \\ t_b, & m = [2,4,6] \end{cases} \tag{2.29}$$

$$t_Y = \begin{cases} t_a, & m = [2,4,6] \\ t_b, & m = [1,3,5] \end{cases} \tag{2.30}$$

A Tabela 2.3 apresenta as condições para ser determinado o setor de trabalho m onde se encontra o vetor de referência, por meio da relação entre $V_\propto$ e V_β .

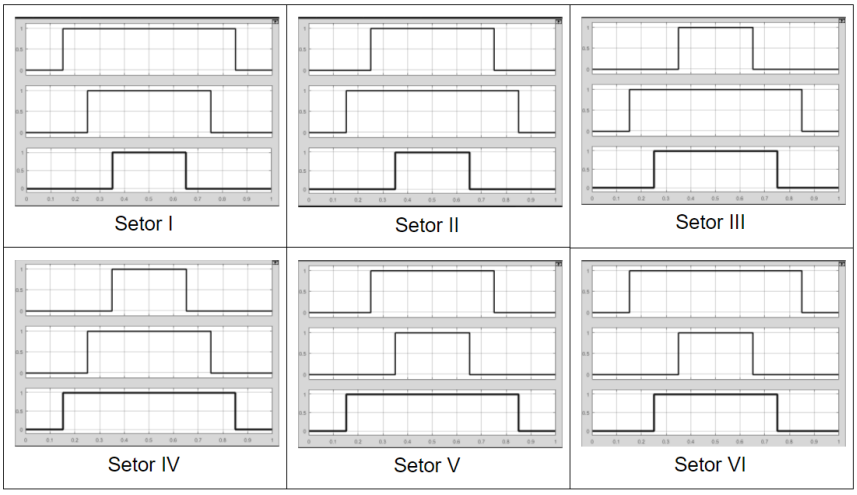
Tabela 2.3 - Critérios de identificação do setor.

Setor (m)	**Ângulo do Vetor**	**Condição**	
		$V_\propto$	V_β
I	[0°, 60°]	$\sqrt{3}V_\propto > V_\beta$	$V_\beta > 0$
II	[60°, 120°]	$\sqrt{3}V_\propto \leq V_\beta$ ou $-\sqrt{3}V_\propto < V_\beta$	$V_\beta > 0$
III	[120°, 180°]	$-\sqrt{3}V_\propto \geq V_\beta$	$V_\beta > 0$
IV	[180°, 240°]	$\sqrt{3}V_\propto > V_\beta$	$V_\beta \leq 0$
V	[240°, 300°]	$\sqrt{3}V_\propto \geq V_\beta$ ou $-\sqrt{3}V_\propto > V_\beta$	$V_\beta \leq 0$
VI	[300°, 360°]	$-\sqrt{3}V_\propto \leq V_\beta$	$V_\beta \leq 0$

Fonte: GARCÍA, 2015

Existem seis padrões de chaveamento de $S_r, S_s \; e \; S_t$ para a Figura 2.11, de acordo com o setor onde encontra-se o vetor de referência.

Figura 2.11 - Padrão de chaveamento.



Fonte: Autor.

O tempo de comutação pode ser simplificado calculando diretamente o período o qual cada chave permanece ligada de acordo com as equações 2.31 a 2.33:

$$T_{R-on} = \begin{cases} \frac{t_{pwm}}{2}\left(1 + f_c \frac{3}{2V_{cc}}\left[-V_\alpha - \frac{V_\beta}{\sqrt{3}}\right]\right), & m = [1,4] \\ \frac{t_{pwm}}{2}\left(1 + f_c \frac{3}{2V_{cc}}[-2V_\alpha]\right), & m = [2,5] \\ \frac{t_{pwm}}{2}\left(1 + f_c \frac{3}{2V_{cc}}\left[-V_\alpha + \frac{V_\beta}{\sqrt{3}}\right]\right), & m = [3,6] \end{cases} \quad (2.31)$$

$$T_{S-on} = \begin{cases} \frac{t_{pwm}}{2}\left(1 + f_c \frac{3}{2V_{cc}}\left[V_\alpha - \sqrt{3}V_\beta\right]\right), & m = [1,4] \\ \frac{t_{pwm}}{2}\left(1 + f_c \frac{3}{2V_{cc}}\left[-\frac{2V_\beta}{\sqrt{3}}\right]\right), & m = [2,5] \\ \frac{t_{pwm}}{2}\left(1 + f_c \frac{3}{2V_{cc}}\left[V_\alpha - \frac{V_\beta}{\sqrt{3}}\right]\right), & m = [3,6] \end{cases} \quad (2.32)$$

$$T_{T-on} = \begin{cases} \frac{t_{pwm}}{2}\left(1 + f_c \frac{3}{2V_{cc}}\left[V_\alpha + \frac{V_\beta}{\sqrt{3}}\right]\right), & m = [1,4] \\ \frac{t_{pwm}}{2}\left(1 + f_c \frac{3}{2V_{cc}}\left[\frac{2V_\beta}{\sqrt{3}}\right]\right), & m = [2,5] \\ \frac{t_{pwm}}{2}\left(1 + f_c \frac{3}{2V_{cc}}\left[V_\alpha + \sqrt{3}V_\beta\right]\right), & m = [3,6] \end{cases} \quad (2.33)$$

Para obter o período em que a chave permanece desligada, é utilizada a relação 2.34

$$T_{off} = T_{on} - t_{pwm} \quad (2.34)$$

O fator de correção f_c é unitário para a região de operação linear, representado pelo hexágono da Figura 2.8, para a região de sobremodulação seu valor depende do ângulo e da amplitude de compensação. Foi observado nas equações (2.31), (2.32) e (2.33) um termo em comum que se repete, portanto a equações foram generalizadas conforme descrito pela equação (2.35).

$$T_{on} = \frac{t_{pwm}}{2}\left(1 + f_c \frac{3}{2V_{cc}}\left[f(V_\propto), f(V_\beta)\right]\right) \quad (2.35)$$

A equação (2.36) é utilizada para determinar o índice de modulação (IM) do *SVPWM* que varia entre 0 e 1. A região linear de operação ocorre para IM < 0,91. Quando 0,91 < IM < 0,95, *SVPWM* opera no modo de sobremodulação I, enquanto que o *SVPWM* opera no modo de sobremodulação II quando o índice de modulação é maior a 0,95. (KUMAR e DAS, 2014)

$$IM = \frac{V_{ref}}{V_{fund}} \quad (2.36)$$

$$V_{fund} = \frac{2V_{dc}}{\pi} \quad (2.37)$$

2.5 Transformada Z

Para a compreensão dos tópicos na sequência, é necessário o entendimento sobre a Transformada Z, o qual é o equivalente à transformada de Laplace no domínio discreto. Através da Transformada de Laplace, equações diferencias definidas no tempo contínuo são transformadas em equações algébricas. De forma análoga, a Transformada Z converte equações de diferenças definidas no tempo discreto em equações algébricas (LATHI, 1992). A transformada Z permite definir funções de transferência em tempo discreto: a relação entre as entradas e saídas de um sistema linear definido no tempo discreto. À diferença das equações diferenciais em tempo contínuo, as equações de diferenças podem ser facilmente implementadas em um processador digital. Assim, após definir a função de transferência de um sistema (por exemplo, um controlador ou observador) é possível implementar este sistema em um processador digital (*FPGA* nesta Dissertação).

A Transformada Z unilateral pode ser expressa de acordo com a equação (2.38), enquanto sua transformada inversa é dada de acordo com a equação (2.39). Para estender a equação (2.38) à Transformada Z bilateral, os limites são definidos entre $-\infty\ e\ \infty$, em vez de 0 e ∞ (LATHI, 1992).

$$X(z) = \sum_{n=0}^{\infty} x(n)z^{-n} \tag{2.38}$$

$$x(n) = \frac{1}{2\pi j}\oint X(z)\, z^{n-1}dz \tag{2.39}$$

Para aplicações em engenharia, a Transformada Z, assim como sua inversa, são representadas por meio de tabelas, possibilitando a operação de transformação sem a utilização das formulas. (LATHI, 1992) A transformada Z existe quando a somatória dos valores infinitos na equação (2.38) possui convergência, conforme a restrição em (2.40). (HAYKIN e VEEN, 2003)

$$\sum_{n=-\infty}^{\infty}|x(n)r^{-n}| < \infty \tag{2.40}$$

Onde a variação para r em que o valor da somatória na equação (2.40) tenha um valor finito é chamada de região de convergência, (HAYKIN e VEEN, 2003).

A tabela 2.4 apresenta, de forma resumida, as principais propriedades presentes na Transformada Z.

Tabela 2.4 - Propriedades da transformada Z

Operação	$\boldsymbol{x(n)}$	$\boldsymbol{X(z)}$
Adição	$x_1(n) + x_2(n)$	$X_1(z) + X_2(z)$
Multiplicação Escalar	$\alpha x(n)$	$\alpha X(z)$
Atraso	$x(n-1)$ $x(n-2)$ $x(n-m)$	$z^{-1}X(z)$ $z^{-2}X(z)$ $z^{-m}X(z)$
Avanço	$x(n+1)$ $x(n+2)$ $x(n+m)$	$zX(z)$ $z^2X(z)$ $z^mX(z)$
Multiplicação por Expoente	$\alpha^m x(n)$	$X\left(\frac{z}{\alpha}\right)$
Multiplicação por n	$nx(n)$	$-z\frac{d}{dz}X(z)$
Convolução	$x_1(n) * x_2(n)$	$X_1(z)X_2(z)$
Valor Inicial	$x(0)$	$\lim_{z\to\infty} X(z)$
Valor Final	$\lim_{n\to\infty} x(N)$	$\lim_{z\to 1}(z-1)X(z)^1$

1 – Polos de $(z-1)X(z)$ dentro do círculo unitário

Fonte: LATHI, 1992.

2.5.1 Função de Transferência Discreta

Para obtenção da função de transferência, parte-se de um sistema onde a saída $y(n)$ é expressa em termos de resposta ao impulso $h(n)$ e da entrada $x(n)$ por meio da convolução:

$$y(n) = h(n) * x(n) \tag{2.41}$$

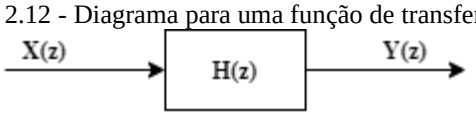
Aplicando a transformada Z em ambos os lados conforme as propriedades na tabela 2.4 resulta em:

$$Y(z) = H(z)X(z) \tag{2.42}$$

A função de transferência discreta $H(z)$ é definida como a relação entre a saída e a entreda do sistema modelado em tempo discreto (HAYKIN e VEEN, 2003), tal como indicado na equação (2.3). A Figura 2.12 representa o conceito de função de transferência discreta.

$$H(z) = \frac{Y(z)}{X(z)} \tag{2.43}$$

Figura 2.12 - Diagrama para uma função de transferência.



Fonte: LATHI, 1992.

A equação (2.43) pode ser reescrito com auxílio da equação (2.38) e das propriedades de transformada Z da seguinte forma (condições inicias nulas):

$$H(z) = \frac{\sum_{n=0}^{k} b_k z^{-k}}{\sum_{n=0}^{m} a_n z^{-n}} = \frac{b_k z^k + b_{k-1} z^{k-1} + \cdots b_1 z + b_0}{z^n + a_{n-1} z^{n-1} + \cdots a_1 z + a_0} \quad (2.44)$$

2.5.2 Equação das Diferenças

Considerando as equações (2.43) e (2.44):

$$\frac{b_k z^k + b_{k-1} z^{k-1} + \cdots b_1 z + b_0}{z^n + a_{n-1} z^{n-1} + \cdots a_1 z + a_0} = \frac{Y(z)}{X(z)} \quad (2.45)$$

Desenvolvendo a equação (2.45) obtém-se:

$$Y(z)(z^n + a_{n-1} z^{n-1} + \cdots a_1 z + a_0) = X(z)(b_k z^k + b_{k-1} z^{k-1} + \cdots b_1 z + b_0) \quad (2.46)$$

Conforme a Tabela 2.4, $z^{-m}X(z) = x(n-m)$, assim a equação (2.46) pode ser representada por meio de atrasos da seguinte forma:

$$y_{(n)} + a_{n-1} y_{(n-1)} + \cdots a_1 y_{(0)} + a_0 = b_k x_{(k)} + b_{k-1} x_{(k-1)} + \cdots b_1 x_{(0)} + b_0 \quad (2.47)$$

Isolando $y_{(n)}$ da equação (2.47) obtém-se:

$$y_{(n)} = b_k x_{(k)} + b_{k-1} x_{(k-1)} + \cdots b_1 x_{(0)} + b_0 \; + a_{n-1} y_{(n-1)} + \cdots a_1 y_{(0)} + a_0 \quad (2.48)$$

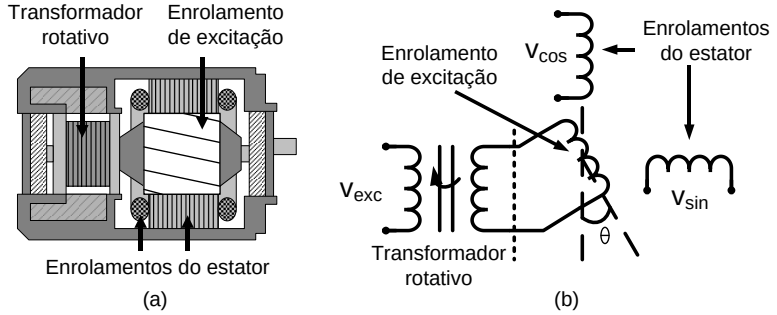
A equação (2.48) é chamada de equação das diferenças, e permite a representação da função de transferência por meio do atraso da entrada e da saída. Conhecendo-se as condições iniciais, o termo $b_1 x_{(0)} + b_0 + a_1 y_{(0)} + a_0$ é reduzido a uma constante.

O uso da equação das diferenças permite a implementação de sistemas discretos em sistemas *FPGA*.

2.6 Sensor de Posição Resolver

Sensores de posição tipo *resolver* se diferenciam de *encoders* por fornecer a informação de posição em formato analógico e com maior precisão. Sua estrutura consiste em um enrolamento rotórico de excitação e dois enrolamentos estacionários de saída defasados em 90º, conforme a Figura 2.13.

Figura 2.13 - Sensor resolver: (a) diagrama físico; (b) circuito esquemático.



Fonte: (GARCÍA, 2015)

A onda senoidal de excitação V_{exc} chega ao enrolamento do rotor conforme (2.45) através de um transformador rotativo, com frequência entre 1 kHz e 20 kHz.

$$V_{exc} = a_{exc} sen(\omega_{exc} t) \tag{2.45}$$

Os enrolamentos de saída possuem tensões induzidas pela excitação do motor de acordo com a equação (2.46) e (2.47).

$$V_{sen} = k a_{exc} \left[sen(\theta) sen(\omega_{exc} t) + \frac{d\theta}{dt} \frac{\cos(\theta) sen(\omega_{exc} t)}{\omega_{exc}} \right] \tag{2.46}$$

$$V_{cos} = k a_{exc} \left[cos(\theta) sen(\omega_{exc} t) + \frac{d\theta}{dt} \frac{\cos(\theta) sen(\omega_{exc} t)}{\omega_{exc}} \right] \tag{2.47}$$

Sendo:

ω_{exc} : Frequência do sinal de excitação;

k : Razão entre os enrolamentos do estator e o rotor do sensor;

θ : Ângulo mecânico;

V_{exc} : Sinal de excitação de entrada;

a_{exc} : Amplitude do sinal de excitação;

$V_{sen,}\, V_{cos}$: Saídas do sensor.

As equações (2.46) e (2.47) podem ser simplificadas caso a velocidade mecânica seja menor que o sinal de excitação, conforme as equações (2.48) e (2.49):

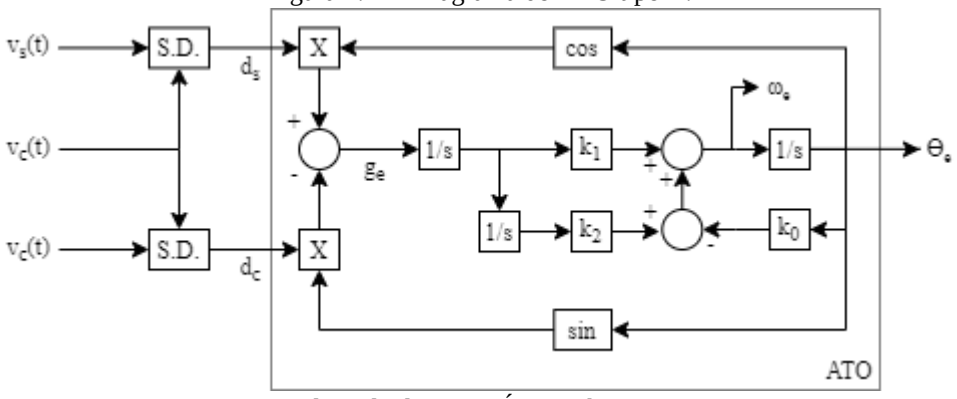
$$V_{sen} = ka_{exc}sen(\theta)sen(\omega_{exc}t) = kV_{exc}sen(\theta) \quad (2.48)$$

$$V_{cos} = ka_{exc}cos(\theta)sen(\omega_{exc}t) = kV_{exc}cos(\theta) \quad (2.49)$$

Por meio de (2.48) e (2.49) pode ser observado que os sinais de saída do sensor resolver são modulados em amplitude (GARCÍA, 2015).

A Figura 2.14 apresenta uma solução para a leitura das informações fornecidas pelo sensor de posição resolver proposto por (GARCÍA, *et al.*, 2018). Neste sistema, um demodulador síncrono permite obter os envelopes d_s e d_c dos sinais de saída do sensor. Os sinais demodulados d_s e d_c são usados para estimar o ângulo por meio do observador em malha fechada *ATO* (*Angle Tracking Observer, ATO em inglês*).

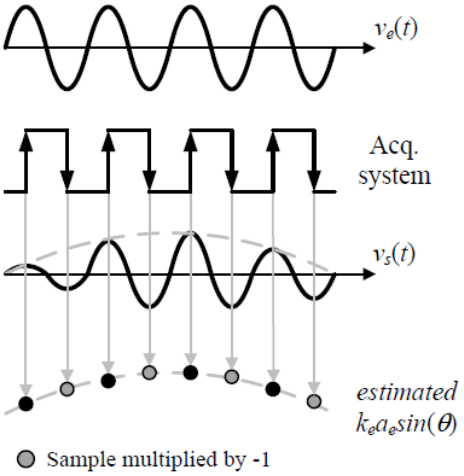
Figura 2.14 - Diagrama do ATO tipo II.



Adaptado de GARCÍA, *et al.*, 2018.

No demodulador síncrono, o sinal de excitação é sincronizado com uma onda quadrada conforme a Figura 2.15. As transições positivas e negativas da onda quadrada acontecem quando o sinal de excitação do resolver atinge seus valores extremos.

Figura 2.15 - Demodulador síncrono.



Fonte: Adaptado de GARCÍA, *et al.*, 2018

A onda quadrada é utilizada para demodular as informações de saída do sensor resolver: são utilizadas as amostras dos sinais de saída do sensor que correspondem às transições positiva e negativa da onda quadrada. Naqueles instantes de tempo, $sen(\omega_{exc}t) = 1$. Como resultado os sinais demodulados d_s e d_c são obtidos:

$$d_s = ka_{exc}sen(\theta) \tag{2.50}$$

$$d_c = ka_{exc}cos(\theta) \tag{2.51}$$

Em decorrência do uso dos picos e dos vales do sinal de excitação, a frequência de amostragem será o dobro da frequência da onda de excitação. Com as saídas do resolver demoduladas, é realizado a multiplicação pelo seno ou cosseno do ângulo estimado. Assim, o sinal g_e é obtido conforme a solução abaixo:

$$u_s = d_s \cos(\theta_e) = ka_{exc}sen(\theta)\cos(\theta_e) \tag{2.52}$$

$$u_c = d_c \,\mathrm{sen}(\theta_e) = ka_{exc}cos(\theta)\,\mathrm{sen}(\theta_e) \tag{2.53}$$

$$g_e = u_s - u_c$$

$$g_e = ka_{exc}sen(\theta)\cos(\theta_e) - ka_{exc}\cos(\theta)\,\mathrm{sen}(\theta_e)$$

$$g_e = ka_{exc}[sen(\theta)\cos(\theta_e) - \cos(\theta)\,\mathrm{sen}(\theta_e)]$$

$$g_e = ka_{exc}sen(\theta - \theta_e) \approx ka_{exc}(\theta - \theta_e) \tag{2.54}$$

O *ATO* pode ser modelado em espaço de estados usando como referência o sistema proposto em GARCÍA, *et al.*, 2018, de acordo com as equações (2.55) e (2.56). Na modelagem

proposta, o vetor E é o vetor de erro dos estados, enquanto que os ganhos k_0, k_1, e k_2 devem ser projetados para que a operação do ATO tenha boa resposta dinâmica e robustez a ruído. É possível usar técnicas como a fórmula de Ackermann para sintonizar os ganhos.

$$\dot{E} = (A - BK)E \tag{2.55}$$

$$A = \begin{bmatrix} 0 & 0 & 0 \\ -1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix}; \quad B = \begin{bmatrix} ka_{exc} \\ 0 \\ 0 \end{bmatrix}$$

$$K = [k_0 \quad -k_1 \quad -k_2] \tag{2.56}$$

2.7 *FPGA-In the Loop*

O *FPGA* (*Field Programmable Gate Array*, em inglês) utilizam blocos lógicos programáveis, que permitem a implementação de modelos matemáticos via *linguagem de programação em Hardware*, o qual determina a forma como os componentes internos irão ser conectados entre si. Os *FPGA*s são reconfiguráveis, permitindo a alteração e recopilação de novos modelos matemáticos. Ademais, os *FPGAs* podem ser configurados para operarem com processamento serial ou paralelo (INTEL, 2020). Nesta dissertação, os algoritmos *SVPWM* e da leitura do sensor resolver foram implementados no Kit de Desenvolvimento DE2-115 da ALTERA, baseado no *FPGA EP4CE115F29C7N*, conforme ilustrado pela Figura 2.16.

Figura 2.16 - Kit FPGA DE12-115.

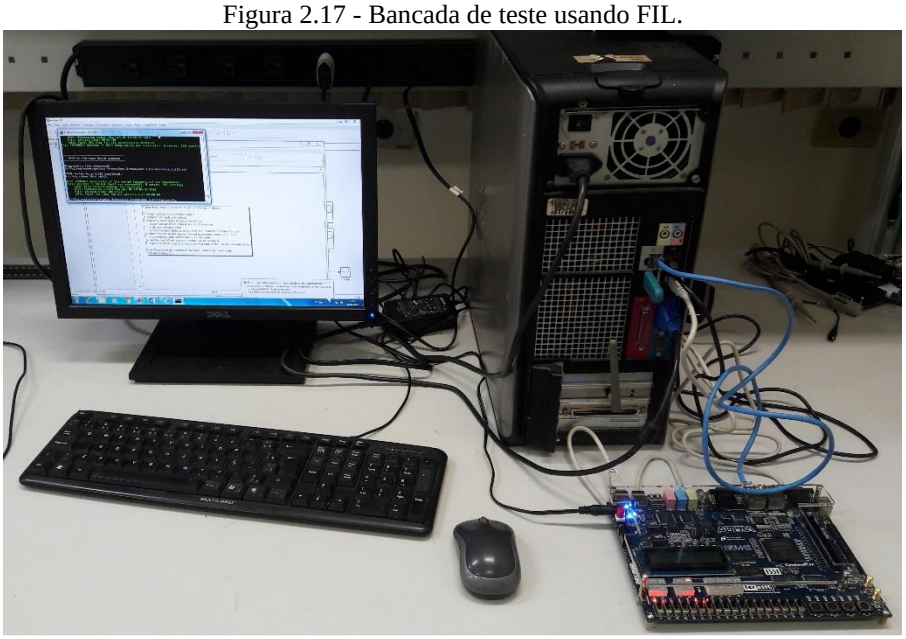
Fonte: Autor.

O *FIL* consiste em uma técnica de testes onde algoritmo a ser testado é implementado no *FPGA.* Esta técnica oferece as vantagens de teste de *software* em conjunto com os testes de *hardware*, pois permite a implementação do sistema de controle em *hardware* com a planta simulada em *software* (tal como seria implementado na prática) (MATHWORKS, 2020). Nesta

dissertação, os sistemas de testes foram implementados em *SIMULINK*. Um cabo Ethernet permite a comunicação de dados entre o Kit do *FPGA* e o computador onde o algoritmo de teste será executado.

O *FIL* permite avaliar algoritmos desenvolvidos em *FPGA* considerando efeitos como os atrasos de processamento e o número limitado de bits nas operações aritméticas, enquanto é possível usar diferentes valores de sinais de testes. Esta técnica é muito útil no teste do algoritmo de leitura do sensor resolver, sendo que o verdadeiro valor da posição angular é conhecido no ensaio. Um sensor de posição de alta exatidão (de alto custo e difícil de obter) seria necessário para verificar a exatidão do valor de posição estimado quando o algoritmo é testado com um resolver físico. A Figura 2.17 mostra a implementação do sistema *FIL* usado nos ensaios experimentais efetuados.

Figura 2.17 - Bancada de teste usando FIL.



Fonte: Autor

3 MODELAGEM MATEMÁTICA E COMPUTACIONAL

3.1 Aproximação da Função Sigmoide usando *SPLINE* e Algoritmo Genético

3.1.1 População Inicial

A população inicial de indivíduos em um algoritmo genético é escolhida aleatoriamente. No entanto, é possível definir critérios na escolha de forma a resultar em uma distribuição uniforme ou reduzir o esforço computacional.

De acordo com (2.17), (2.18) e (2.19), presentes no capítulo 2.2, os trechos dos polinômios *SPLINE* são reduzidos a, respectivamente, $[0, L_3]$, $[L_3, L_4]$ e $[L_4, 8]$.

Cada segmento é aproximado por um polinômio de grau *n*, de acordo com (2.16), presente no capítulo 2.2 o qual foi incluído ao gene dos indivíduos da população inicial, resultando no cromossomo de acordo com (3.1).

L_3	L_4	n_{P_3}	n_{P_4}	n_{P_5}

(3.1)

Onde n_{P_3} , n_{P_4} e n_{P_5} correspondem respectivamente ao grau do polinômio nos intervalos 3, 4 e 5.

Em vista de obter melhor distribuição dos valores correspondentes para os indivíduos que compõe a população inicial, assegurar que o nó L_4 tenha magnitude superior a L_3, , a geração aleatória é realizada de acordo com as seguintes restrições, onde a constante d determina a distância mínima entre as variáveis:

$$L_4 - L_3 > d \tag{3.2}$$

$$1 \leq [n_{P_3}, n_{P_4}, n_{P_5}] \leq 4 \tag{3.3}$$

A população aleatória inicial é gerada conforme (3.4) e (3.5) a partir de um valor randômico $rand$ entre 0 e 1, respeitando as restrições em (3.2) e (3.3); $round$ é utilizado para arredondar o valor, uma vez que não existe grau de polinômio fracionário.

$$[L_3, L_4] = 8 * rand \tag{3.4}$$

$$[n_{P_3}, n_{P_4}, n_{P_5}] = round[(3 * rand) + 1] \tag{3.5}$$

A Tabela 3.1 tem um exemplo comparativo da população inicial gerada aletoriamente e com uso da restrição proposta em (3.2).

Tabela 3.1 - Comparação na geração da população inicial.

Geração Puramente Aleatória				**Geração com Restrição**			
$rand$	L_3	$rand$	L_4	$rand$	L_3	$rand$	L_4
0,04	0,29	0,74	5,95	0,27	2,17	0,67	5,38
0,85	6,79	0,39	3,14	0,09	0,68	0,66	5,26
0,93	7,47	0,66	5,24	0,21	1,66	0,39	3,10
0,68	5,43	0,17	1,37	0,30	2,43	0,94	7,55
0,76	6,06	0,71	5,65	0,63	5,07	0,77	6,13

Fonte: Autor.

É possível observar a ocorrência de valores maiores em L_3 com relação a L_4 para a geração puramente aleatória, assim como magnitudes semelhantes, situações os quais não ocorrem com o uso de (3.2).

Outras restrições foram testadas, apesar dos resultados na geração da população inicial para as condições adicionais, não houve impacto sobre a otimização do Algoritmo Genético. Portanto a restrição proposta é suficiente para gerar a população inicial.

3.1.2 Função Objetivo

O *fitness* foi definido como sendo o menor modulo dentre os máximos erros absolutos entre a função sigmoide $f(x)$ e sua aproximação por polinômios *SPLINE* $f_a(x)$, conforme a equação (3.2)

$$função\_custo = |f(x) - f_a(x)| \tag{3.2}$$

Para análise complementar, foi inserido o erro quadrático médio (*mean square error*, *MSE*), conforme equação (3.3).

$$MSE = \frac{1}{n}\sum_{i=1}^{n}[f(x) - f_a(x)]^2 \tag{3.3}$$

A escolha do erro absoluto como parâmetro de otimização no lugar do *MSE*, ocorreu pois o erro quadrático médio realiza a média de todos os valores aproximados no intervalo do

polinômio. Com isso os picos de erro pontual típicos na transição entre polinômios decorrente da técnica *SPLINE* não são adequadamente representados pelo *MSE*.

A função fitness deve considerar as restrições na implementação da função de aproximação. Uma das mais importantes é o número de bits que serão utilizados nos cálculos aritméticos no *FPGA*. Neste trabalho, considera-se uma representação numérica de ponto fixo com 16 bits para representar a parte fracionárias dos números.

3.1.3 Ajuste dos Parâmetros

A tabela 3.2 apresenta os parâmetros de configuração do algoritmo genético adotados com base nos testes realizados.

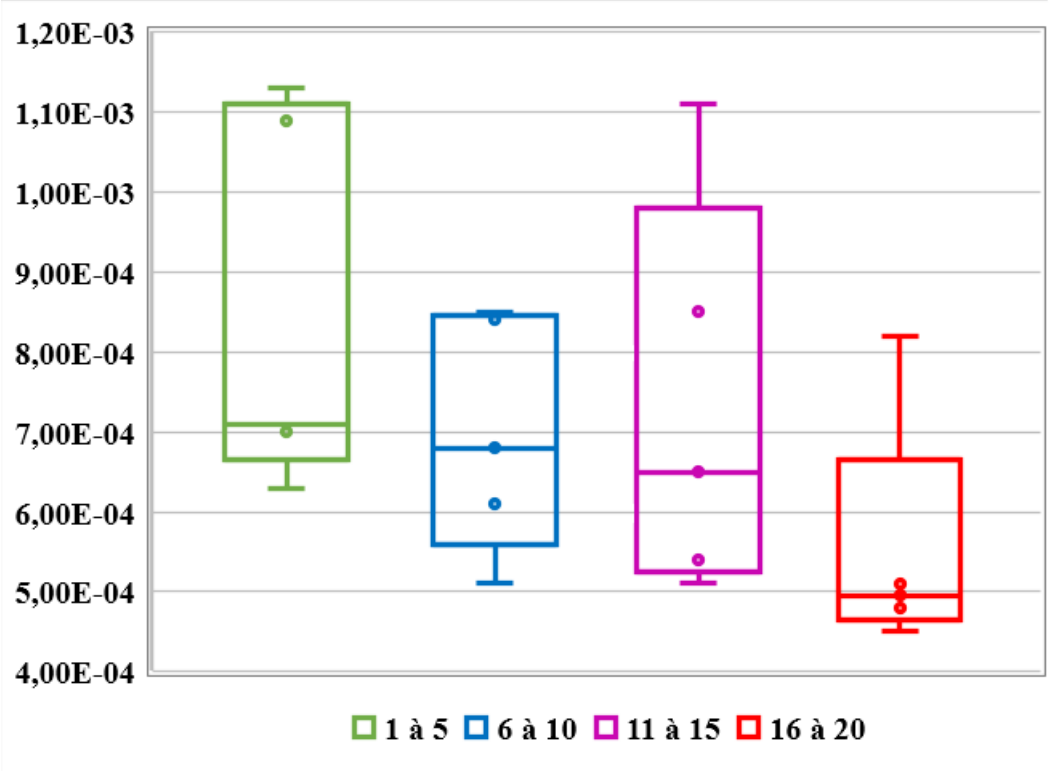
Tabela 3.2 - Parâmetros usados no algoritmo genético.

População Inicial	100
Taxa de *Crossover*	80%
Taxa de Mutação	5%
Número de Gerações	80
Grau do Polinômio.	4

Fonte: Autor

Os parâmetros foram escolhidos por meio de análise gráfica de acordo com a Figura 3.1. Onde foi realizado testes sob os parâmetros da tabela 3.3 e analisada sua influência no *fitness* do resultado final, assim como a convergência a cada geração.

Figura 3.1 - Testes de parâmetros do algoritmo genético.



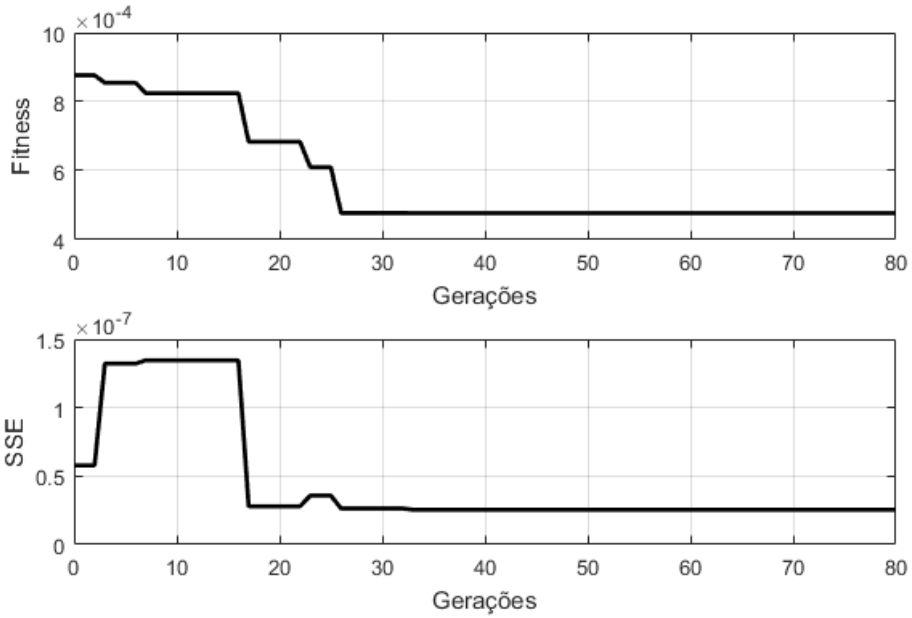
Fonte: Autor.

Tabela 3.3 - Parâmetros usados nos testes do algoritmo genético.

	Testes			
	1 a 5	**6 a 10**	**11 a 15**	**16 a 20**
Número de Indivíduos	50	100	100	100
Taxa de *Crossover*	80%	80%	60%	80%
Taxa de Mutação	1%	1%	1%	5%
Gerações	80	80	80	80

Fonte: Autor.

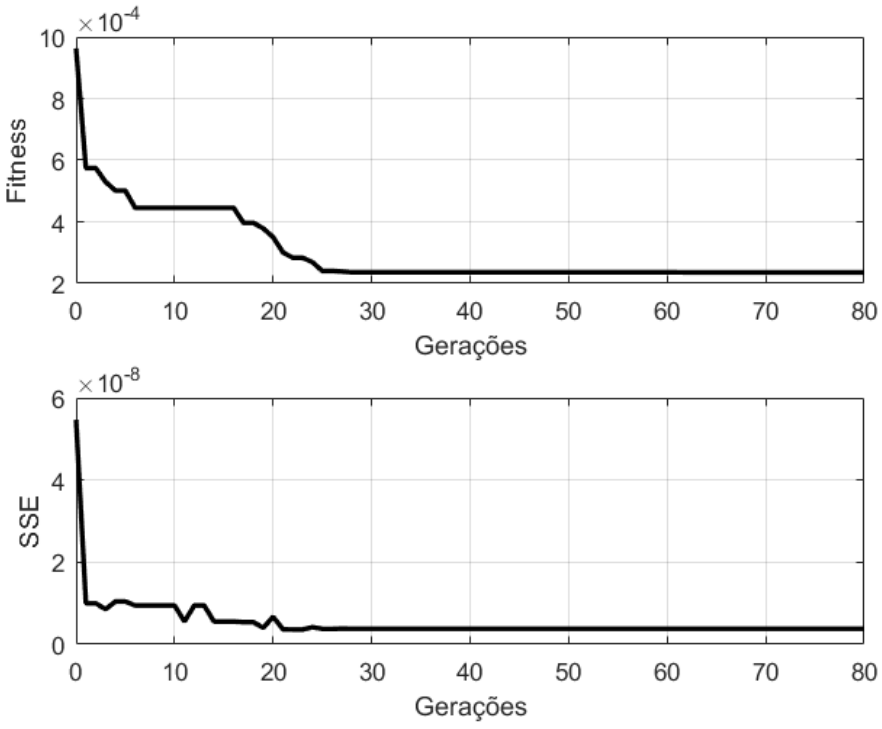
Figura 3.2 - Convergência usando representação numérica de ponto fixo.



Fonte: Autor.

Através da Figura 3.2 observa-se que o erro estabiliza após 30 gerações utilizando ponto fixo para a função objetivo com um bit de sinal, um bit inteiro e 16 bits fracionários. A Figura 3.3 apresenta um caso onde é utilizado uma representação numérica ponto flutuante de 64 bits (formato *double*).

Figura 3.3 - Convergência considerando representação numérica de ponto flutuante.



Fonte: Autor.

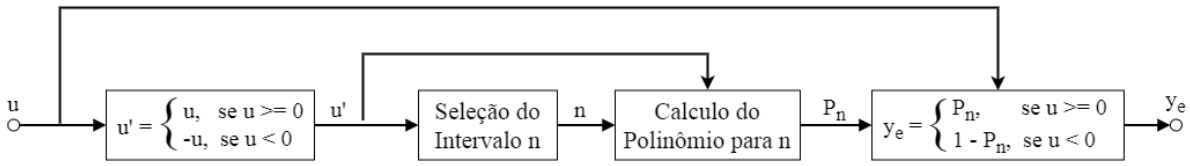
Observa-se uma diferença na forma como a função objetivo converge, isso ocorre pois na representação em ponto fixo o valor do erro está limitado a quantidade de bits adotados. O resultado gráfico sofre variações de acordo com a população aleatória inicial.

Em decorrência das características do *FPGA*, foi considerada a situação o qual faz uso de ponto fixo na escolha do número de gerações, embora seja possível reduzi-lo, optou-se em utilizar 80 gerações para garantir uma convergência adequada.

3.2 Modelagem do Neurônio

Com o intuito de verificar a aplicação da sigmoide aproximada, foram simulados dois neurônios isolados, um usando a função original e o segundo com a aproximação, conforme o modelo matemático da Figura 2.1 estabelecido no capítulo 2.1. A função de ativação aproximada foi modelada de acordo com a Figura 3.4.

Figura 3.4 - Diagrama da sigmoide aproximada.



Fonte: Autor.

De acordo com o capítulo 2.2, um polinômio realiza a aproximação para cada intervalo definida entre nós pertencente a função. A equação (3.4) é utilizada para modelar a seleção de intervalo.

$$\mathrm{n} = \begin{cases} 3, \ u' = [0, \ L_3] \\ 4, \ u' = [L_3, \ L_4] \\ 5, \ u' = [L_4, \ 8] \\ 6, \ u' = [8, \ \infty] \end{cases} \tag{3.4}$$

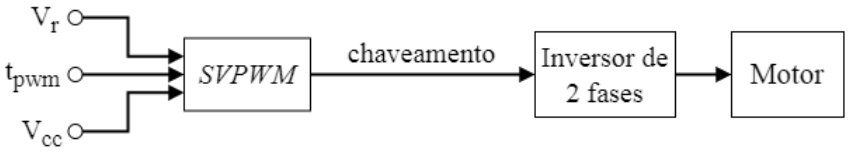
Note que o intervalo 6 refere-se à saturação em 1, dado o sistema discreto. Selecionado o polinômio, sua modelagem é realizada usando a equação (2.10), presente no capítulo 2.2. As constantes foram obtidas por regressão para os nós otimizados pelo algoritmo genético desenvolvido conforme os capítulos 2.2, 2.3 e 3.1.

3.3 Modulação por Vetores Espaciais *SVPWM*

3.3.1 Modelagem da região linear de operação

Foi realizada a simulação para um sistema típico de inversor com o algoritmo *SVPWM*, conforme o diagrama Figura 3.5.

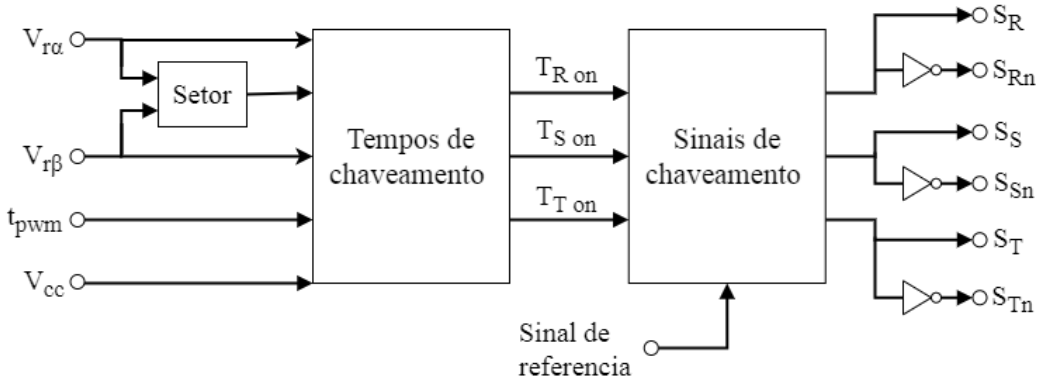
Figura 3.5 - Diagrama do motor com inversor controlado.



Fonte: Autor.

O sistema *SVPWM* modelado, de acordo com a Figura 3.6, usado na simulação utiliza as componentes do vetor de referência para obter-se os parâmetros de chaveamento S_r, S_s, S_t, S_{rn}, S_{sn} e S_{tn}.

Figura 3.6 - Diagrama do sistema *SVPWM*.

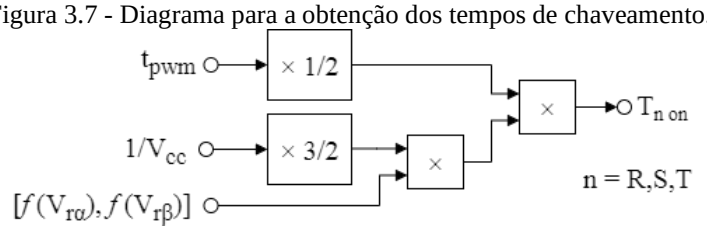


Fonte: Autor.

O setor de trabalho é identificado de acordo com a relação apresentada na Tabela 2.3, presente no capítulo 2.3.

A partir da equação (2.35), foi obtido o modelo para a parte generalizada das equações referentes aos tempos de chaveamento, conforme o diagrama da Figura 3.7.

Figura 3.7 - Diagrama para a obtenção dos tempos de chaveamento.



Fonte: Autor

Este modelo se repete na obtenção individual dos tempos $T_{R\,ON}$, $T_{S\,ON}$ e $T_{T\,ON}$ modelado conforme as equações (2.31), (2.32) e (2.33).

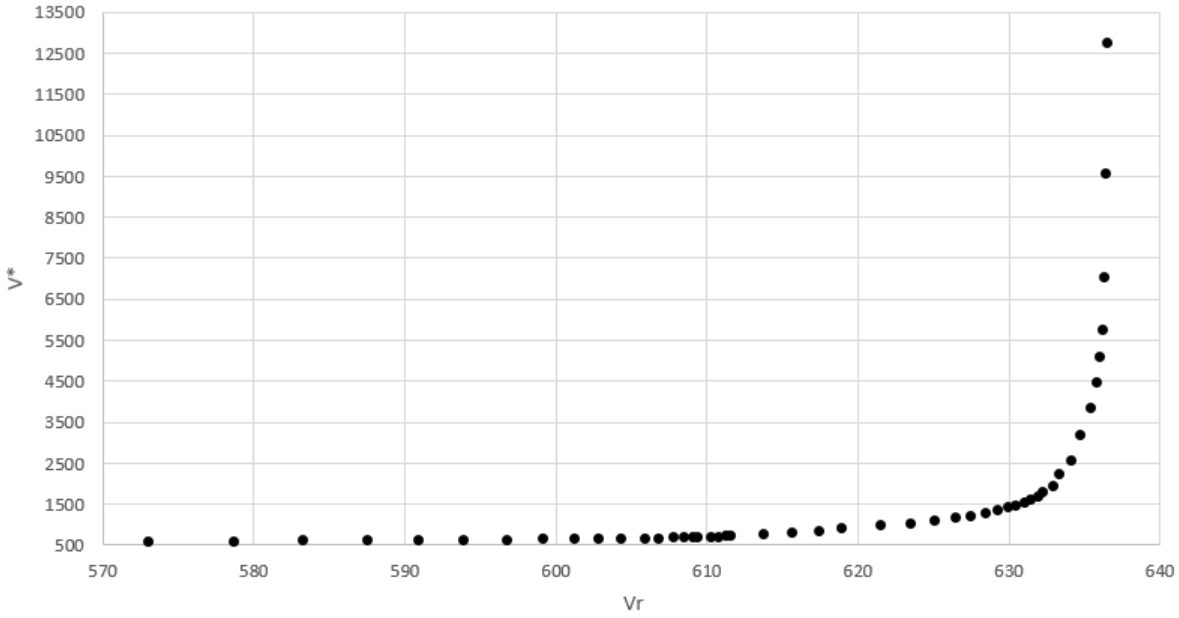
Com os tempos de comutação foi encontrado os padrões de chaveamento ao comparar os tempos a um sinal triangular de referência com amplitude e frequência igual a t_{pwm}.

3.3.2 Modelagem da região de sobremodulação

O índice de modulação utilizado para identificar a região de operação foi modelado conforme as equações (2.36) e (2.37).

O modelo usado na região linear de operação também é valido para a sobremodulação, contanto que seja feita uma compensação, o qual optou-se por aplicar diretamente sobre o vetor de referência, obtida experimentalmente conforme gráfico da Figura 3.8.

Figura 3.8 - Gráfico pontual para compensação de V_r.



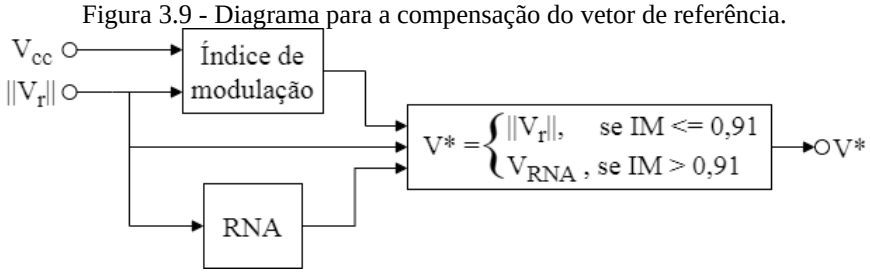
Fonte: Autor.

Como pode ser observado pelo gráfico da Figura 3.16, a variação na compensação do vetor de referência para índices de modulação próximo a um é considerável.

Foi projetada uma rede neural, com o intuito de realizar a compensação do vetor de referência. A rede possui uma camada oculta contendo cinco neurônio com função de ativação sigmoide, e função linear na camada de saída. Para o treinamento da rede foi utilizado os dados experimentais obtidos.

A Figura 3.9 apresenta o diagrama usado para realizar a compensação da sobremodulação através do uso de rede neural.

Figura 3.9 - Diagrama para a compensação do vetor de referência.

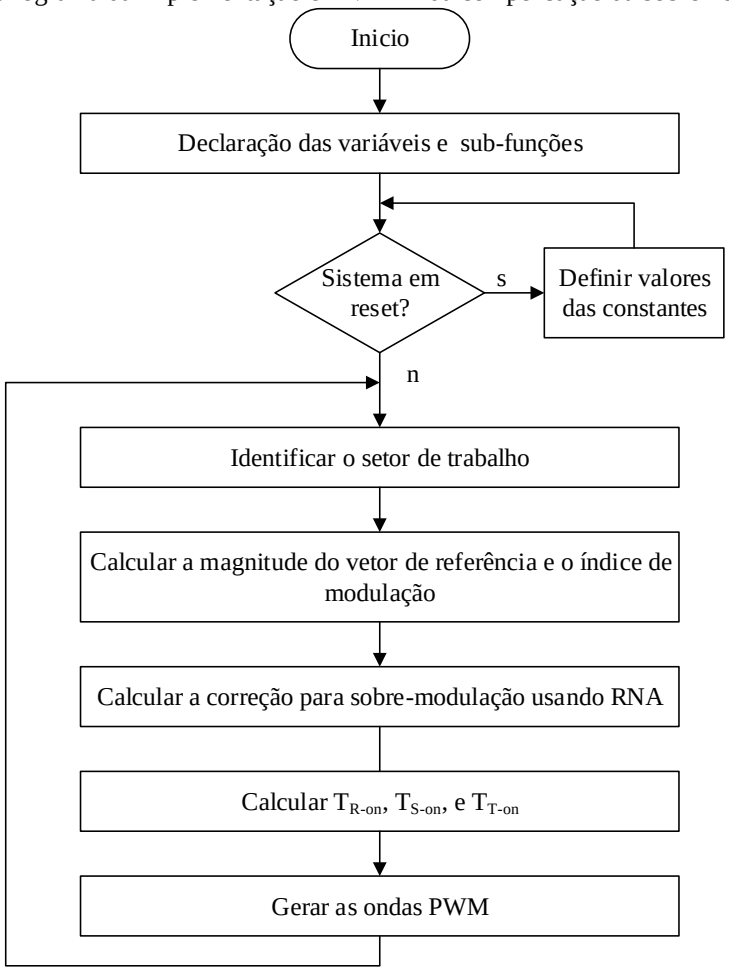


Fonte: Autor

3.3.3 Implementação através de *FPGA in the Loop*

Obtido a rede neural responsável por realizar a estimação da sobremodulação, foi realizado a implementação em *VDHL*, conforme diagrama da Figura 3.10 e a simulação para o *SVPWM*, utilizando-se da ferramenta *FIL* (*FPGA-in-the-loop*) presente no *SIMULINK*.

Figura 3.10 - Fluxograma da implementação em VHDL da compensação da sobremodulação.



Fonte: Autor.

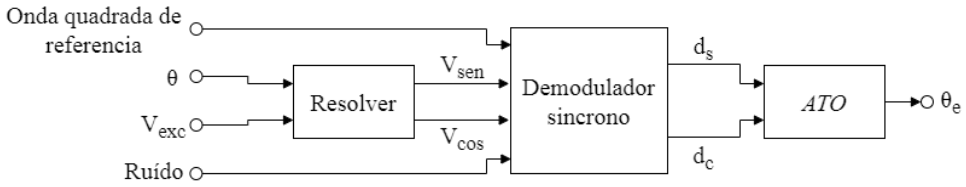
O modelo do *SVPWM* foi utilizado como resultado esperado, o qual teve de passar pelo *FPGA* para evitar que seja computado o erro inerte ao *hardware* ao ser realizado a diferença absoluta entre valores esperados e obtidos. Foram feitas três simulações para a região linear, sobremodulação tipo 1 e tipo 2.

3.4 Leitura de Posição

3.4.1 Modelagem e Simulação

Foi simulado o sistema de leitura de posição conforme diagrama da Figura 3.11, para um sinal de excitação de 5 kHz com amplitude unitária; um gerador de ruído aleatório e um ângulo variante no tempo.

Figura 3.11 - Diagrama para a leitura e posição.



Fonte: Autor.

O sensor resolver usado na leitura de posição, conforme descrito no capítulo 2.6, foi modelado de acordo com a as equações (2.45), (2.46) e (2.47). A partir do ângulo e do sinal de excitação, as saídas do sensor resolver foram calculadas.

No modelo responsável por demodular o sinal gerado pelo sensor resolver, utiliza a tensão de excitação e uma onda quadrada como referência. Neste demodulador, as bordas de subida e descida do sinal de referência são utilizadas para realizar a amostragem, resultando em uma frequência de amostragem de 10khz, o dobro em relação ao sinal de excitação.

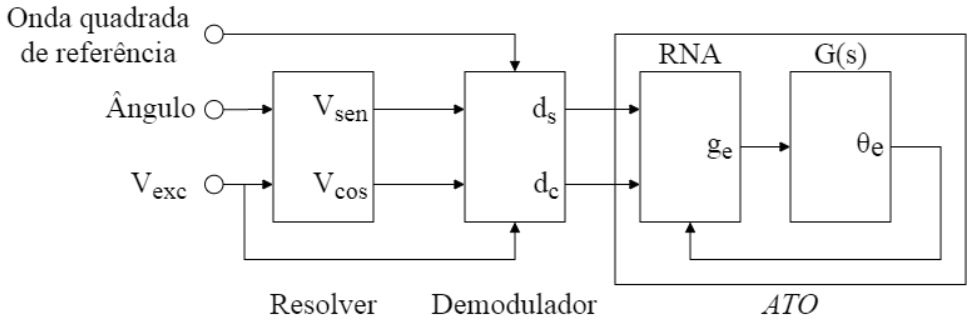
Com os sinais demodulados, foi obtido o ângulo estimado utilizando o observador *ATO* tipo II modelado de acordo com o diagrama da Figura 2.14.

Os integradores $1/s$ foram modelados no domínio discreto conforme proposto por (GARCÍA, *et al.*, 2018), de acordo com solução abaixo:

$$\frac{1}{s} = \frac{t_s z}{z-1}; \quad t_s = \frac{1}{f_s} \tag{3.4}$$

Para a implementação em *FPGA*, o sistema foi subdividido de acordo com a Figura 3.12, onde uma rede neural realiza a aproximação da diferença dos sinais demodulados multiplicados pelo seno e cosseno do ângulo estimado.

Figura 3.12 - Diagrama de implementação da leitura de posição.



Fonte: Autor.

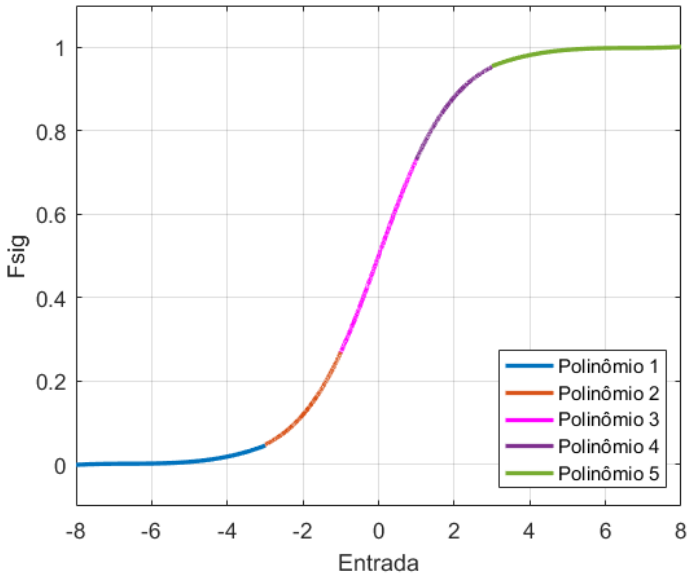
Foi utilizado uma rede neural artificial com seis neurônios na camada oculta, treinado com dados obtidos da simulação do modelo de leitura de posição. Um atraso foi incluído para solucionar problemas de loop algébricos, oriundos da simulação para sistemas discretos.

4 RESULTADOS

4.1 Estimação da Sigmoide.

Para testar a exatidaõ do algoritmo de aproximação proposto, foi utilizado como algoritmo de comparação a técnica descrita em SOARES, 2006, onde a segmentação da função foi efeutada arbitrariamente, conforme a Figura 4.1. O grau do polinômio foi ajustado para quatro.

Figura 4.1 -Sigmoide aproximada com nós arbitrários.



Fonte: Autor.

Onde:

$L_1 = -3$

$L_2 = -1$

$L_3 = 1$

$L_4 = 3$

Realizando o cálculo do erro quadrático médio conforme a equação 3.3 para cada polinômio e para o erro global, obteve-se a tabela 4.1.

Tabela 4.1 - MSE para aproximação arbitrária.

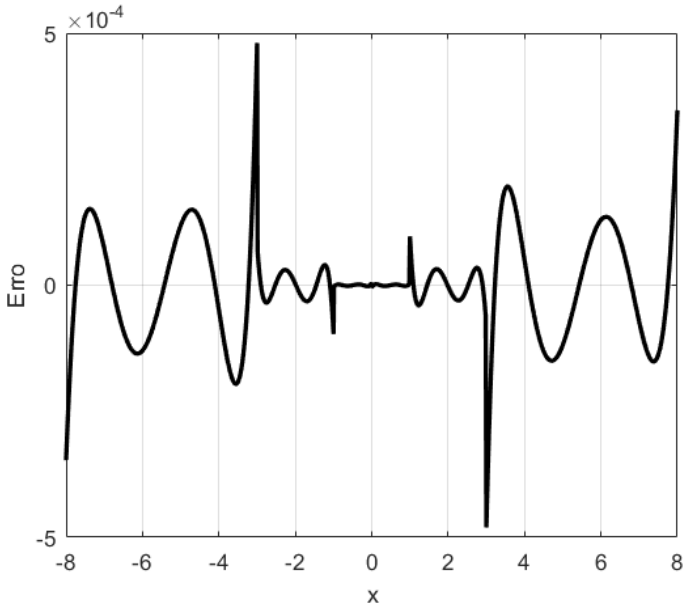
Seguimento	**MSE**
1	$1{,}56 \times 10^{-6}$
2	$7{,}29 \times 10^{-4}$
3	$1{,}78 \times 10^{-2}$
4	$7{,}29 \times 10^{-4}$
5	$1{,}56 \times 10^{-6}$
Global	$9{,}93 \times 10^{-5}$

Fonte: Autor.

Com erro pontual segundo equação (4.1) foi obtido o gráfico da Figura 4.2, onde o ponto de maior erro possui magnitude de $4{,}8 \times 10^{-4}$

$$erro = f(x) - f_a(x) \tag{4.1}$$

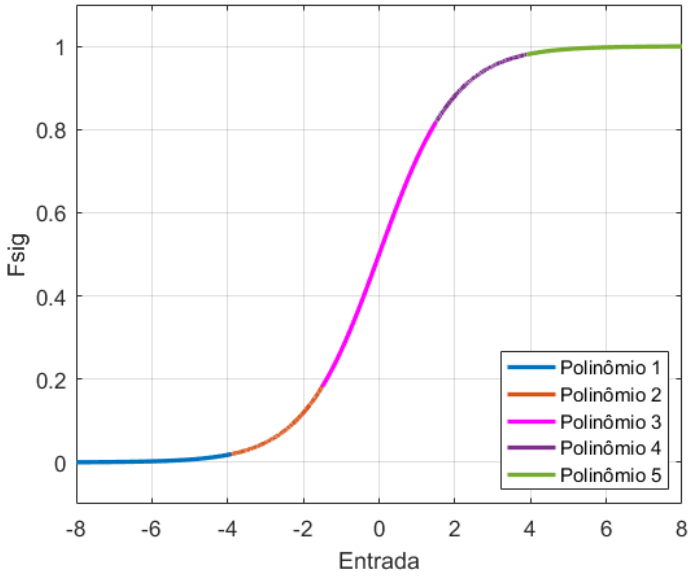
Figura 4.2 - Erro pontual, aproximação arbitrária.



Fonte: Autor.

Utilizando-se de algoritmos genéticos para obter os nós da estimação *SPLINE* por polinômios e quarta ordem resultou no gráfico da Figura 4.3.

Figura 4.3 - Sigmoide aproximada com uso de algoritmos genéticos.



Fonte: Autor.

Sendo:

$L_1 \approx -3.8958933527845554$

$L_2 \approx -1{,}498031040981520$

$L_3 \approx 1{,}498031040981520$

$L_4 \approx 3.8958933527845554$

Para este resultado, o algoritmo genético otimizou o grau dos polinômios utilizados nas aproximações para quatro em todos os casos. A Tabela 4.2 apresenta os erros MSE para cada segmento e o erro global da função aproximada com uso de algoritmos genéticos.

Tabela 4.2 - MSE para aproximação com uso de algoritmos genéticos.

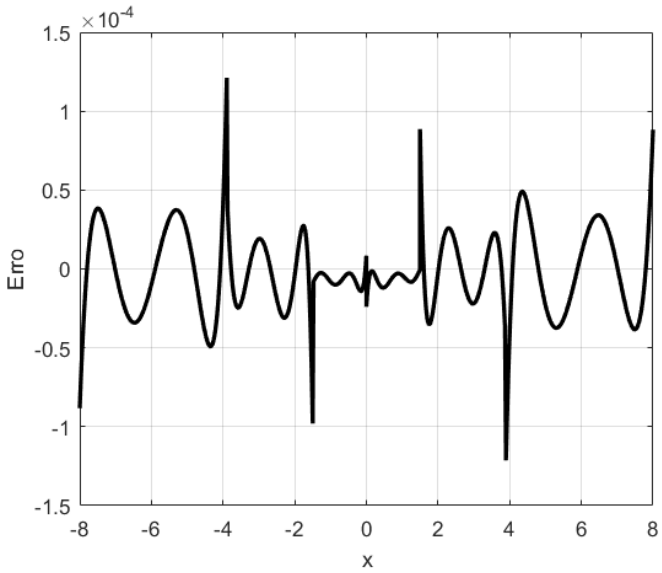
Seguimento	**MSE**
1	$9{,}85 \times 10^{-8}$
2	$4{,}68 \times 10^{-8}$
3	$5{,}61 \times 10^{-7}$
4	$4{,}44 \times 10^{-8}$
5	$9{,}85 \times 10^{-8}$
Global	$6{,}53 \times 10^{-8}$

Fonte: Autor.

Em comparação com os resultados presentes na Tabela 4.1, o erro é melhor distribuído entre cada seguimento que compõe a aproximação e o erro global é

aproximadamente dez vezes maior. O gráfico da Figura 4.4 apresenta o erro pontual ao longo da função.

Figura 4.4 - Erro pontual, aproximação com auxílio de algoritmos genéticos.



Fonte: Autor.

Comparando com a Figura 4.2 o maior erro possui magnitude de $1{,}21x10^{-4}$, cerca de quatro vezes menor. Comportamento similar a ocorrência na análise do *MSE*, onde o erro está distribuído de forma mais uniforme ao longo da função aproximada. Foi observado que os maiores erros absolutos se concentram nas proximidades dos nós, onde ocorre a transição entre polinômios aproximares.

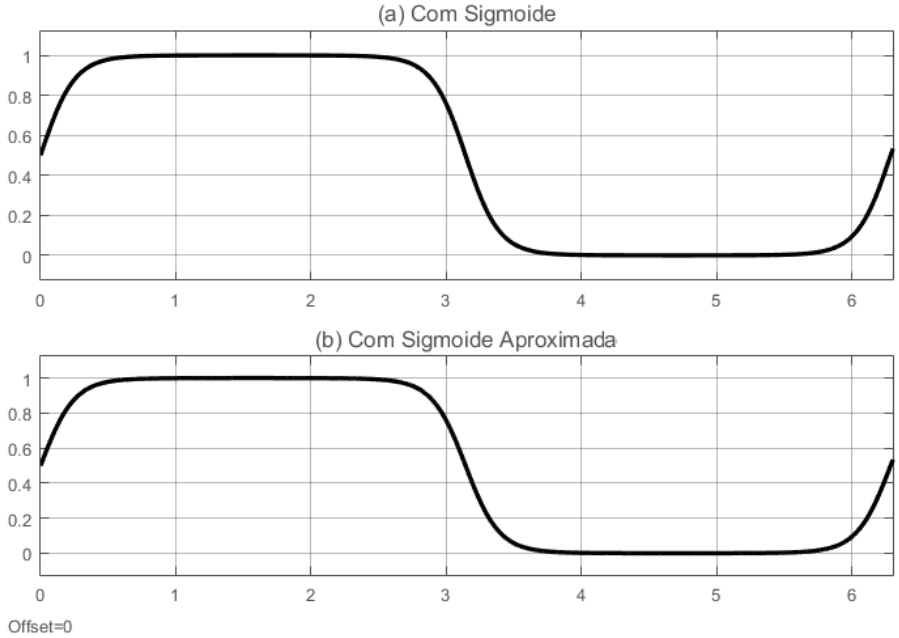
Foram realizadas pesquisas bibliográficas acerca do uso prático da técnica *SPLINE*. No entanto não foram encontradas referências do uso de algoritmos genéticos na otimização da *SPLINE*, até a data de elaboração desta pesquisa de dissertação. Como pode ser observado, a utilização de AG em conjunto com *SPLINE* pode trazer melhora significativa na aproximação realizada pelos polinômios.

4.2 Resultados das Simulações

4.2.1 Simulação do Neurônio

A Figura 4.5 apresenta o resultado da simulação dos neurônios modelados de acordo com o capítulo 3.2, onde 4.5.a corresponde ao neurônio com função de ativação sigmoide, e 4.5.b com função aproximada.

Figura 4.5 - Simulação: (a) Função sigmoide; (b) Função aproximada.



Fonte: Autor.

Onde:

$x_1 = 8sen(t)$

$x_2 = 8sen(0{,}5t)$

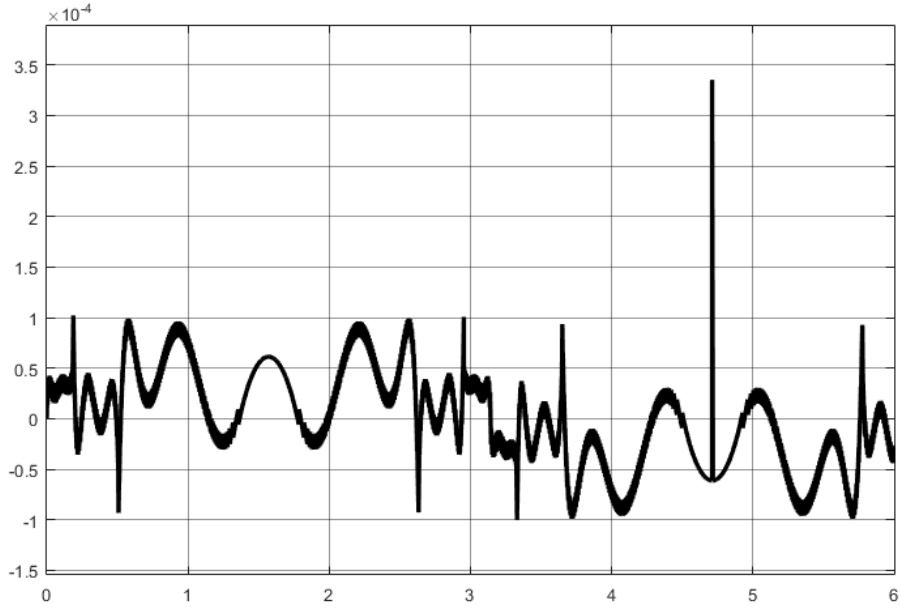
$w_{1,1} = 1$

$w_{1,2} = 0$

$b_1 = 0$

Os parâmetros foram escolhidos propositalmente para que a resposta na entrada da função de ativação variasse no intervalo onde se encontra a aproximação. Graficamente não é possível diferenciar os dois neurônios, para isso foi utilizado o erro apresentado na Figura 4.6.

Figura 4.6 - Erro entre os neurônios.



Fonte: Autor.

O erro é similar ao gráfico do erro absoluto pontual obtido no item 4.1, com magnitude similar. A simulação foi repetida utilizando os parâmetros abaixo:

Onde:

$x_1 = sen(t)$

$x_2 = sen(0{,}5t)$

$w_{1,1} = 1{,}8$

$w_{1,2} = -0{,}2$

$b_1 = -1$

Figura 4.7 - Simulação: (a) Função sigmoide; (b) Função aproximada.
Fonte: Própria.

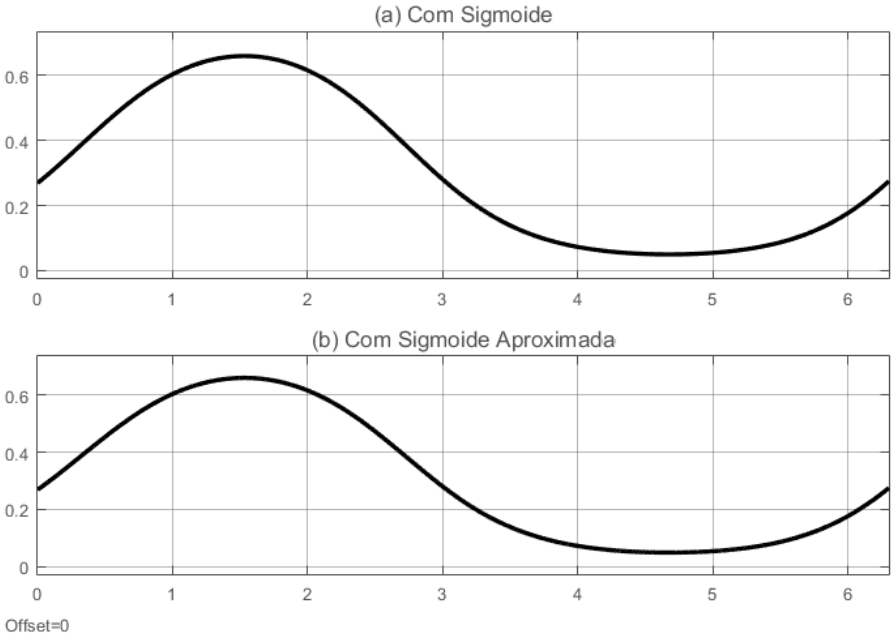
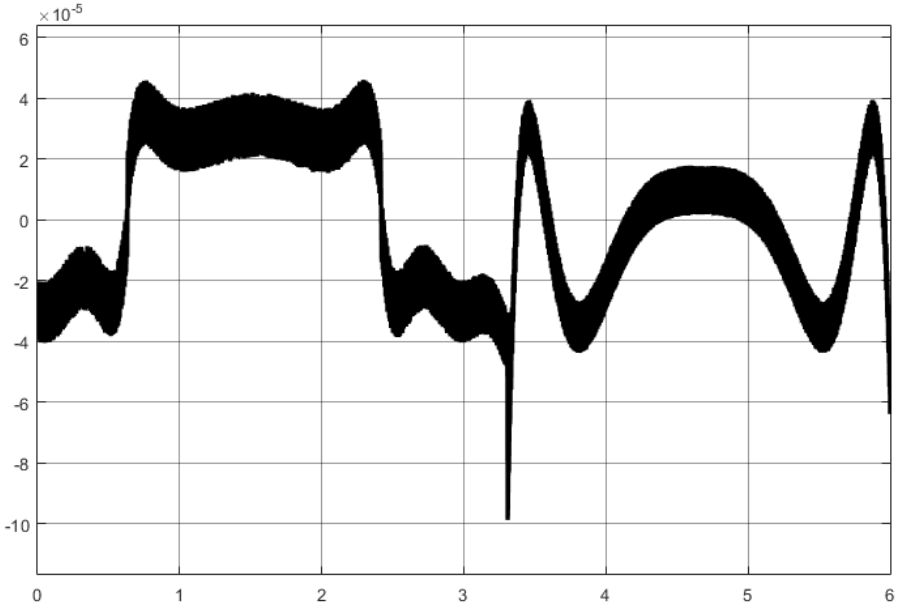


Figura 4.8 - Erro entre os neurônios.

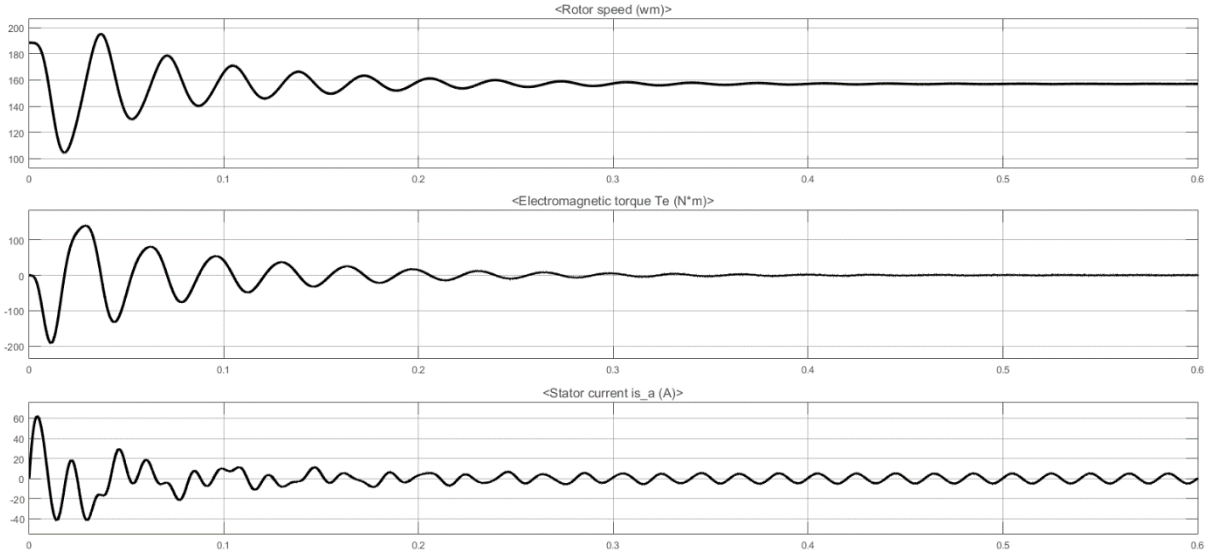


Fonte: Autor.

4.2.2 Simulação do Sistema *SV-PWM*

Simulando o modelo proposto em 3.3 onde a tensão V_{dc} foi definida como 1000 V, a frequência f_s em 5 kHz e vetor de referencia V_r de 500 V aplicado sobre o inversor de um motor simulado de 5 HP, 575 V e 1750 RPM, obteve-se os resultados a seguir:

Figura 4.9 - Resposta simulada do motor ao controle SV-PWM.

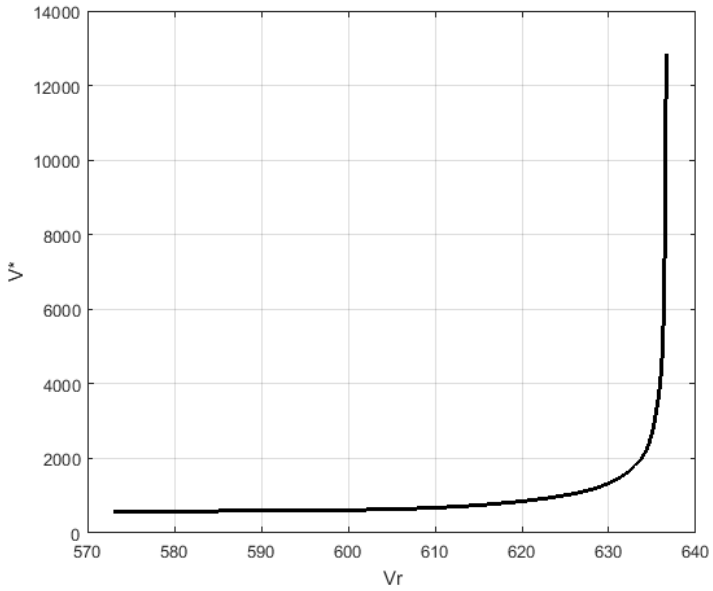


Fonte: Autor.

Observa-se pela resposta do motor que o chaveamento do inversor está sendo controlado pelo sistema *SV-PWM* em um sistema estável.

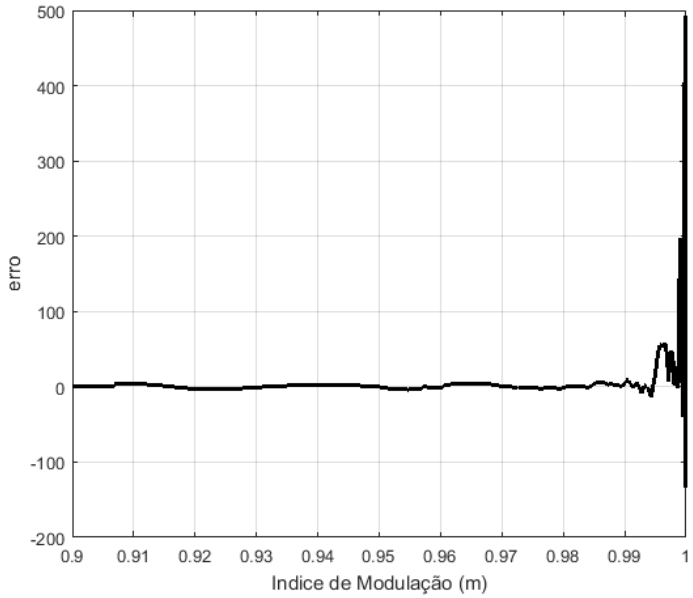
As Figuras 4.10 apresenta o resultado para a simulação do vetor de referência compensado utilizando redes neurais e a Figura e 4.11 o erro em comparação com o uso de *Look-Up Table*, respectivamente.

Figura 4.10 - Gráfico da compensação de V_r.



Fonte: Autor.

Figura 4.11 - Erro para a compensação entre LUT e RNA.



Fonte: Autor.

Observa-se que a rede neural foi capaz de aprender o comportamento da compensação para o vetor de referência. Conforme o índice de modulação se aproxima de um, o erro cresce significativamente, o que é esperado uma vez que o fator de compensação tende ao infinito.

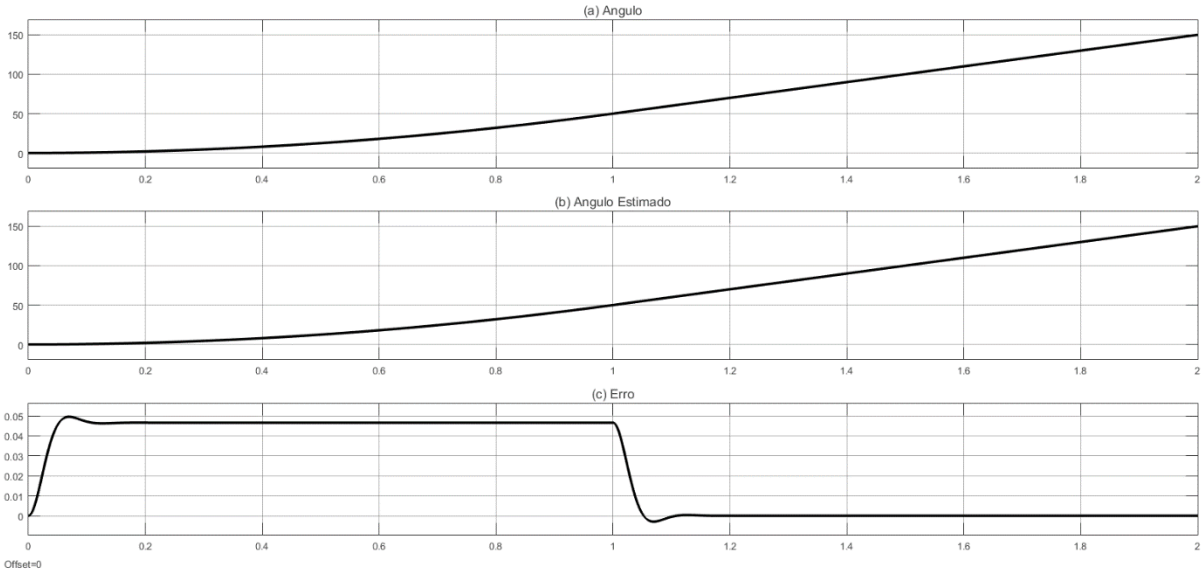
Uma das características das redes neurais, é a capacidade de aprendizado do comportamento dos dados usados no treinamento, que é aplicado aos pontos não usado no treinamento. Com isso, as RNAs se sobressaem a técnicas como *LUT* as quais fazem

aproximações lineares para valores intermediários não carregados na tabela. Desta forma, em aplicações como a compensação do vetor de referência, o uso de redes neurais pode proporcionar uma melhor aproximação para o comportamento desejado.

4.2.3 Simulação da Leitura de Posição

Executando a simulação para o sistema modelado de acordo com o capítulo 3.4, obteve-se a seguinte estimação para o ângulo o rotor:

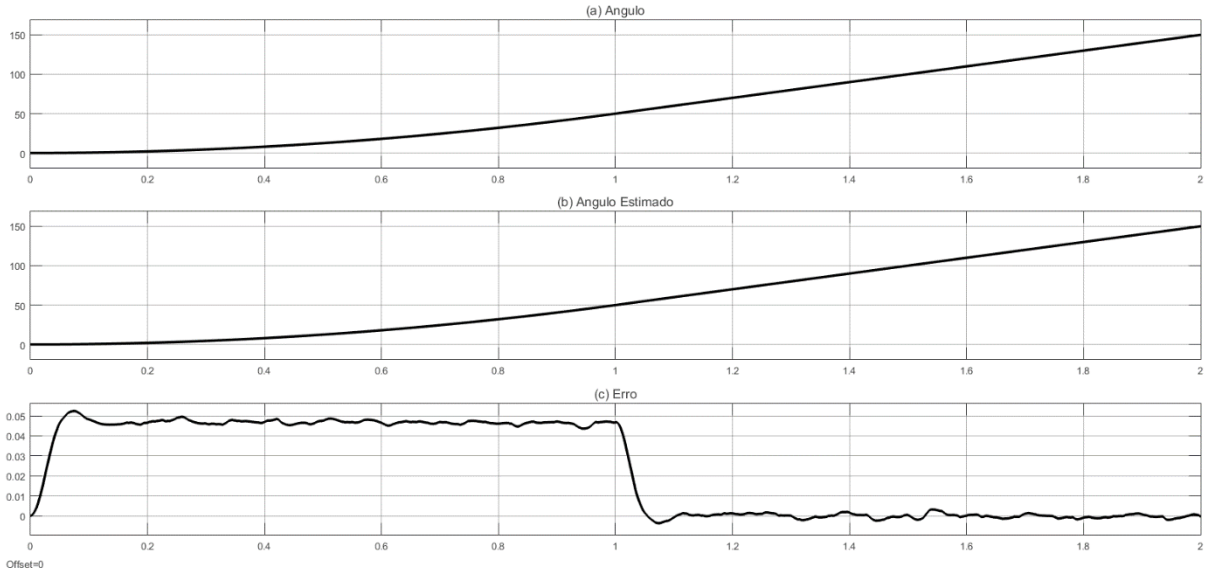
Figura 4.12 - Resultados de simulação para teste sem ruído (a) Ângulo real do rotor; (b) Ângulo estimado pelo ATO; (c) Erro.



Fonte: Autor.

Pode-se observar um erro de regime permanente durante a aceleração (de 0 até 1s). Aquele erro é característico de um sistema tipo-II: erro de rastreamento constante quando a referência (a posição angular neste caso) é uma parábola (que corresponde à posição angular quando a aceleração é constante). Porém, o erro em regime permanente é nulo durante a operação de velocidade constante (a partir de 1 segundo). Ao acrescentar ruído ao sistema, a resposta fica de acordo com a Figura 4.13. As oscilações no erro quando a velocidade angular é constante é produzido pelo ruído, porém, aquelas oscilações possuem uma amplitude menores a 0,004 rad, o que é desprezível.

Figura 4.13 - Resultados de simulação para testes com ruído: (a) Ângulo real do rotor; (b) Ângulo estimado pelo ATO; (c) Erro.



Fonte: Autor.

Os coeficientes utilizados no modelo discretizado do *ATO* utilizado nesta Dissertação foram obtidos usando os ganhos indicados na tabela 4.3. Estes ganhos foram calculados aplicando a fórmula de Ackermann na equação (2.55), para que os polos em malha fechada sejam -75 e -38 ± j54. Estes polos podem mudar dependendo das características desejadas para o *ATO*.

Tabela 4.3 - Coeficientes usados no ATO tipo II

$\mathbf{k_0}$	150
$\mathbf{k_1}$	10025
$\mathbf{k_2}$	322000
$\mathbf{T_s}$	0,0001

Fonte: Autor.

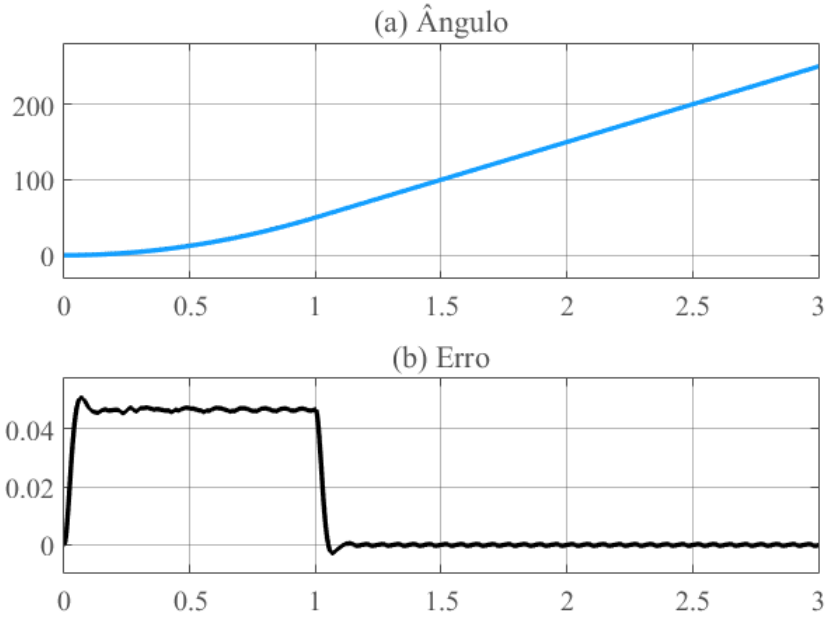
Observando os gráficos conclui-se que o *ATO* é robusto a presença de ruído e capaz de estimar o ângulo do rotor com pico de erro próximo a 0,05 para o sistema simulado.

Foi realizado a simulação do modelo proposto que combina redes neurais com o *ATO*, visando implementação em *FPGA* em trabalhos futuros.

A rede neural projetada utilizou um conjunto de 24001 dados de treinamento cuja as entradas foram d_s, d_c e $\theta_{e(k)}$, o objetivo de treinamento como sendo g_e. Sua estrutura consiste em seis neurônios na camada oculta com função de ativação sigmoide aproximada, e um neurônio na camada de saída com função de ativação linear. Foi normalizado o valor de entrada para o ângulo estimado, com o objetivo de manter a magnitude do valor dos pesos e bias razoável para futuras implementações em sistemas embarcados.

Simulando o sistema com a aproximação proposta, obtém-se a resposta de acordo com a Figura 4.14, observa-se que apesar da adição da rede neural, o comportamento é similar ao obtido pelo *ATO* verificado na Figura 4.12.

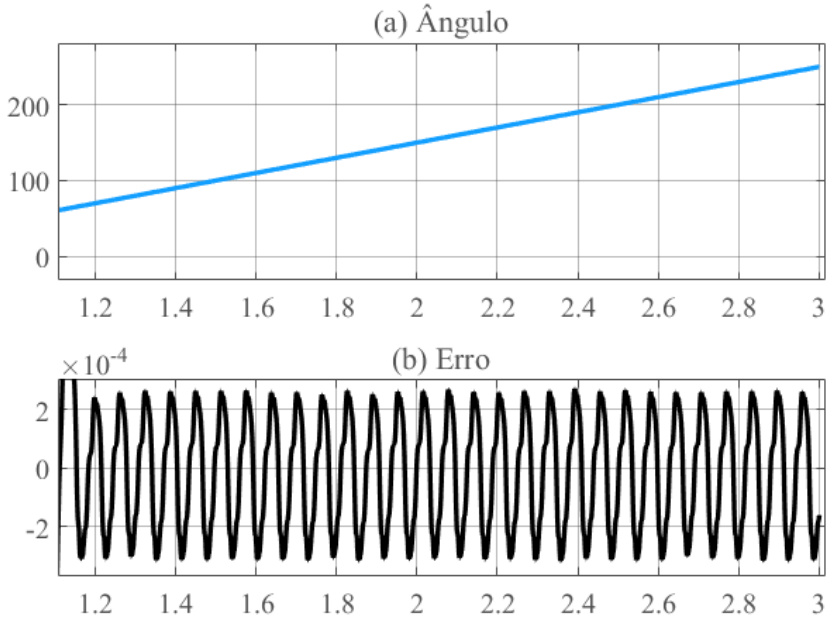
Figura 4.14 - Ângulo estimado, com aproximação por redes neurais.



Fonte: Autor.

Todavia na região linear a resposta está levemente deslocada com relação ao zero e possui uma ondulação com pico de, aproximadamente, $2{,}5 \times 10^{-4}$, conforme a Figura 4.15.

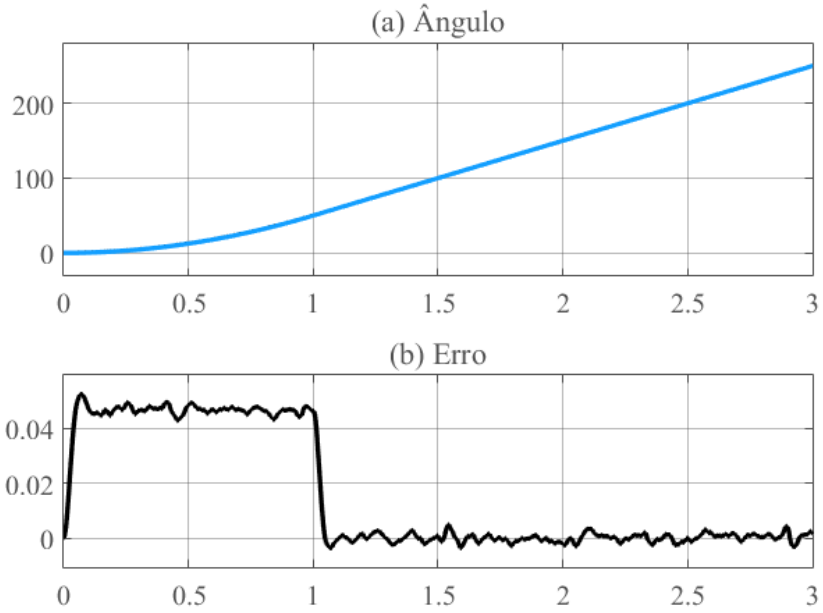
Figura 4.15 - Ângulo estimado, com aproximação por redes neurais, região linear.



Fonte: Autor.

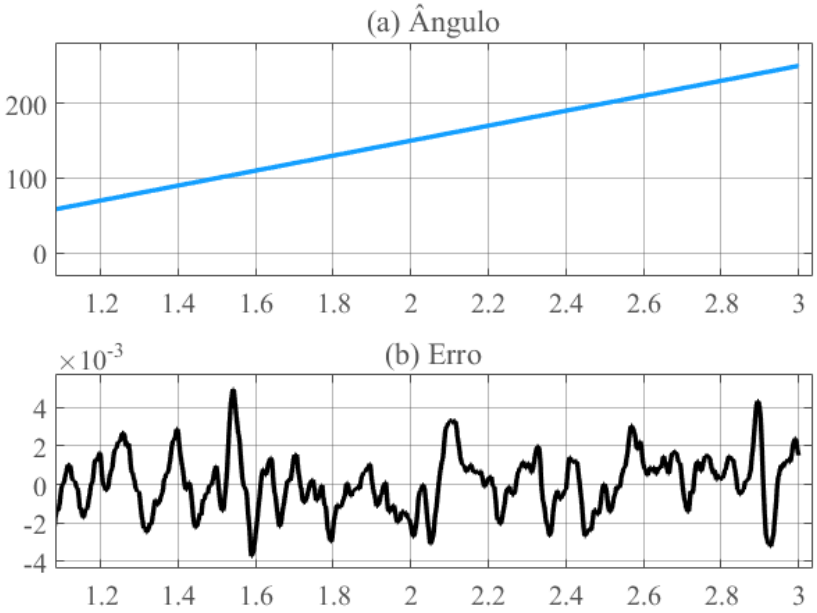
O deslocamento na curva do erro pode ser explicado em decorrência do erro de aproximação da rede neural. A ondulação representa uma limitação da rede ao aprender o comportamento de um sinal dente de serra, como o caso do ângulo estimado. As Figuras 4.16 e 4.17 apresentam a simulação com a inclusão de ruído no sistema.

Figura 4.16 - Ângulo estimado, com aproximação por redes neurais e ruído.



Fonte: Autor.

Figura 4.17 - Ângulo estimado, com aproximação por redes neurais e ruído, região linear.



Fonte: Autor.

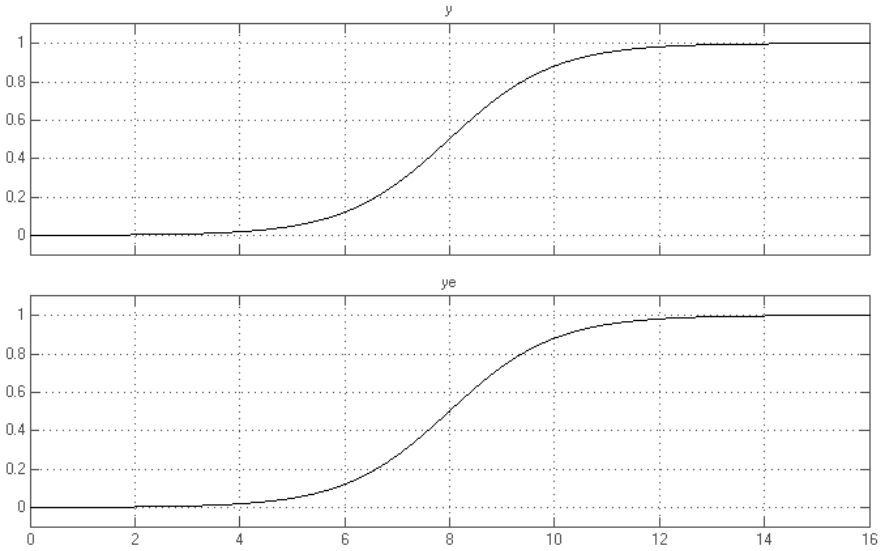
4.3 Implementação em *FPGA*

4.3.1 Implementação da Sigmoide

Com a finalidade de observar o desempenho da função sigmoide aproximada em sistemas embarcados *FPGA*, foi realizado a implementação da aproximação por polinômios *SPLINE* utilizando linguagem *VHDL*. Com auxílio de uma biblioteca de ponto fixo permitindo a implementação de valores fracionários.

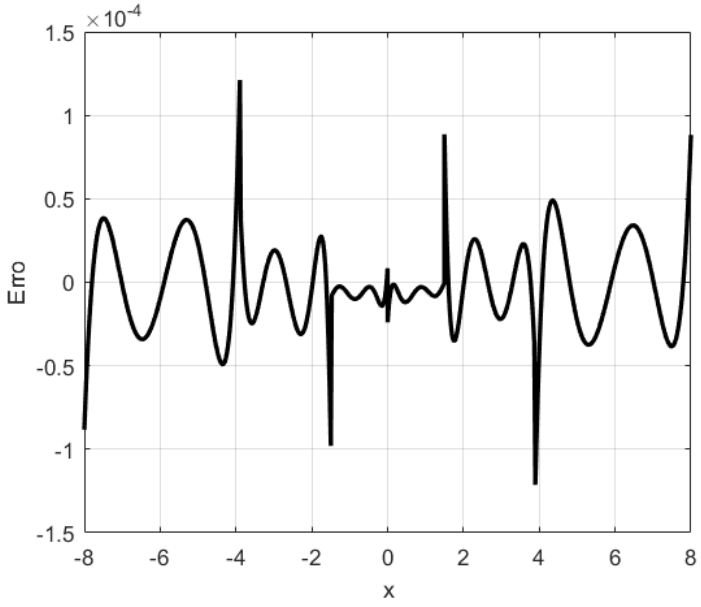
A implementação foi realizada com o auxílio do *SIMULINK* por meio do *FIL* (*FPGA in the Loop*), onde foram simulados sinais de entrada que são carregados no *FPGA* e processados pelo algoritmo *VHDL* o qual realiza a aproximação. Essa ferramenta suprime a necessidade da montagem de circuitos integrados para realizar os testes acerca do código fonte implementado em *FPGA*, poupando assim tempo no desenvolvimento. O resultado é enviado ao *software* e a resposta plotada, conforme as Figuras 4.18 e 4.19.

Figura 4.18 - Aproximação da Sigmoide em FPGA.



Fonte: Autor.

Figura 4.19 - Erro para a aproximação no FPGA.



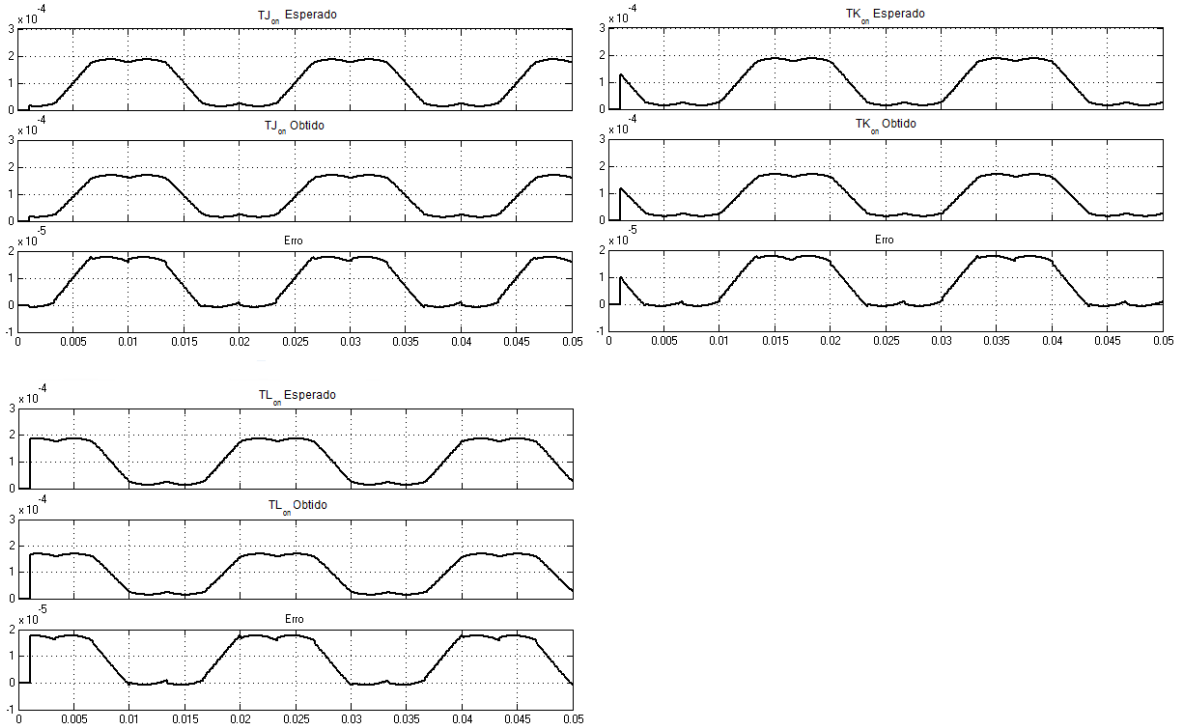
Fonte: Autor.

Observa-se que o erro possui comportamento semelhante ao observado nas simulações das Figuras 4.4 e 4.6, onde o pico ocorre em aproximadamente em $1{,}2 \times 10^{-4}$. Fatores como taxa de amostragem, representação binaria e a forma como o *FPGA* processa as operações matemáticas podem ser responsáveis por diferenciações na resolução da resposta pertinente.

4.3.2 Implementação do *SVPWM*

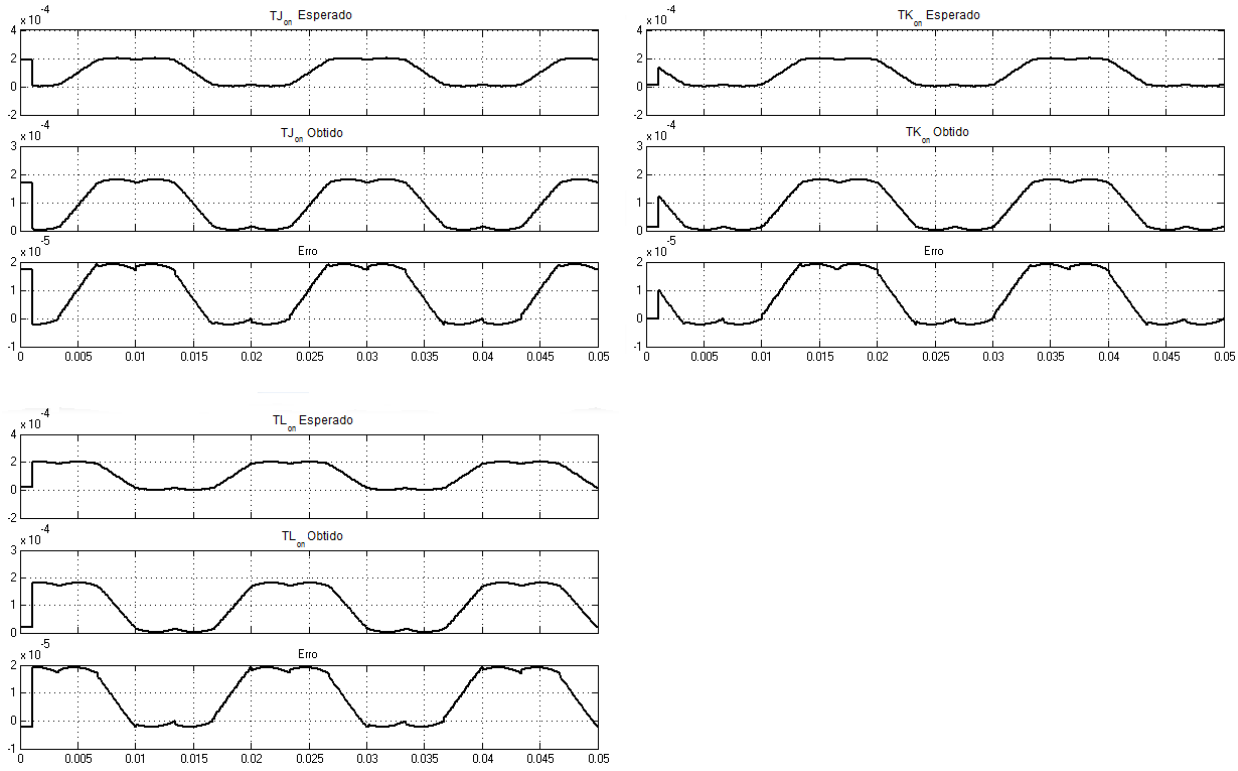
Foi realizada a implementação do *SVPWM* através de *FPGA in the Loop*. Os tempos T_{on} para cada região de operação foram obtidos e comparados ao resultado da simulação, conforme apresentado pelas Figuras 4.20 à 4.22. A compensação para o vetor de referência foi realizado utilizando a aproximação por redes neurais artificiais.

Figura 4.20 - Tempos de comutação para a região linear de operação.



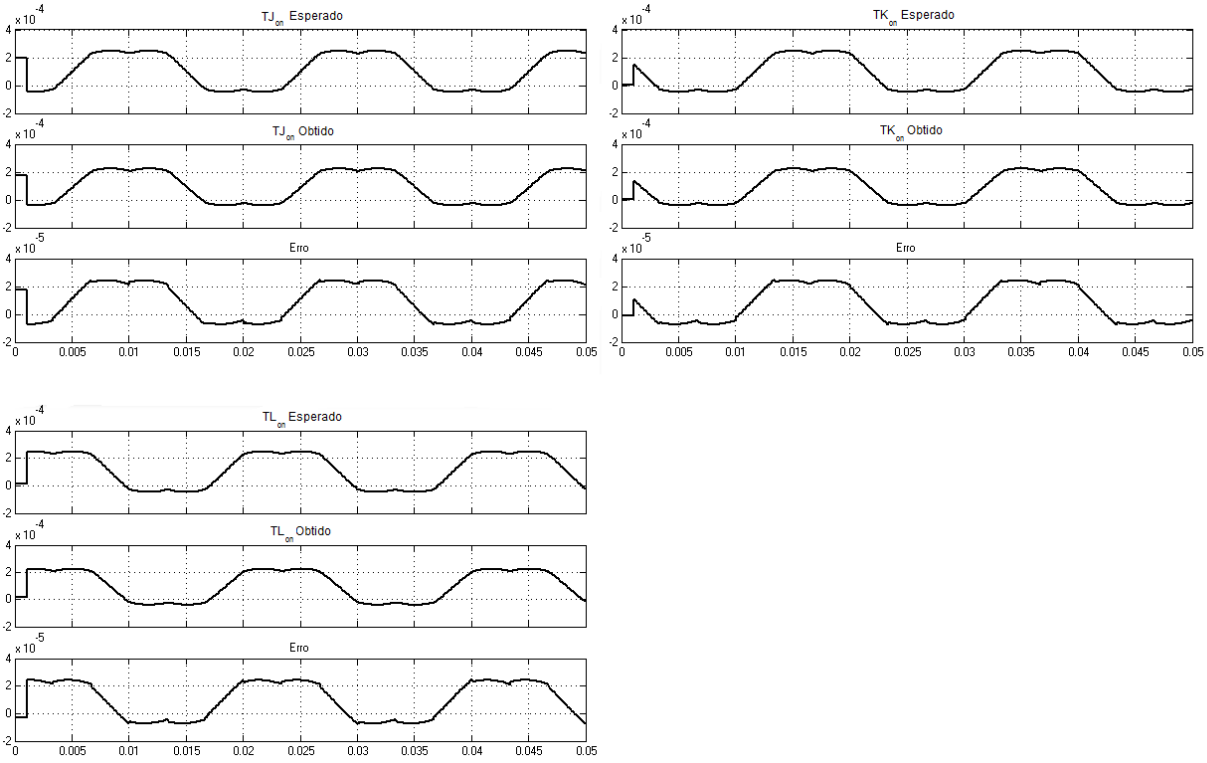
Fonte: Autor.

Figura 4.21 - Tempos de comutação para a região de sobremodulação modo 1.



Fonte: Autor.

Figura 4.22 - Tempos de comutação para a região de sobremodulação modo 2.



Fonte: Autor.

Para este caso, não houve alteração na estrutura do *SVPWM*. Portanto, os erros absolutos obtidos refletem aos truncamentos realizados pela implementação em *FPGA* para as operações intermediárias. Por meio da análise gráfica foi obtido a maior magnitude do erro para os temos de chaveamento R, S e T, nas respectivas regiões: Linear, sobremodulação modo 1 e modo 2, conforme a tabela 4.4.

Tabela 4.4 - Erro absoluto para a implementação em FPGA do SVPWM.

Região de Operação	$T_{R\,on}$	$T_{S\,on}$	$T_{T\,on}$
Linear (m = 0,76)	$1{,}77 \times 10^{-5}$	$1{,}78 \times 10^{-5}$	$1{,}77 \times 10^{-5}$
Sobremodulação modo 1 (m = 0,91)	$1{,}91 \times 10^{-5}$	$1{,}93 \times 10^{-5}$	$1{,}91 \times 10^{-5}$
Sobremodulação modo 2 (m = 0,97)	$2{,}42 \times 10^{-5}$	$2{,}45 \times 10^{-5}$	$2{,}42 \times 10^{-5}$

Fonte: Autor.

Foi observado que conforme se aproxima do índice de modulação ao valor unitário, maior o erro absoluto, o que é esperado tendo a tendência ao infinito apresentada pela compensação do vetor de referência. Deve ainda ser levado em conta que foi estipulado 32 bits para representar os tempos de chaveamento, portanto à medida que o sistema tende ao infinito, a resposta no *FPGA* irá tender a saturação.

5 CONCLUSÕES

Esta dissertação apresentou a implementação em *FPGA* para o *SVPWM* e a simulação para o *ATO* com uso de redes neurais artificiais. Os resultados experimentais demonstram a exatidão e a robustez dos modelos propostos.

O objetivo deste trabalho de dissertação não somente é a implementação destes sistemas em *FPGA, m*as também o uso de aproximações não convencionais como RNA e *SPLINE* com o intuito de manter baixo custo computacional. Tendo em vista que *hardwares* mais robustos apresentam custo econômico maior.

Ao utilizar Algoritmos Genéticos para realizar o ajuste dos parâmetros usados pela técnica *SPLINE,* o erro de estimação foi reduzido significativamente em comparação ao uso de parâmetros escolhidos arbitrariamente. Desta forma, sistemas que necessitam maior exatidão com o uso de redes neurais, podem aproximar a função de ativação sigmoide com menor grau dos polinômios *SPLINE* ao ajustar os parâmetros usando redes neurais. É possível obter uma estimação otimizada da função sigmoide para uma estrutura de hardware definida (graus do polinômio, número de bits utilizados nos cálculos aritméticos). O uso de Algoritmos Genéticos no cálculo da aproximação sigmoide é um dos principais aportes deste trabalho.

Deve ser levado em consideração, que para grau do polinômio cinco ou superior usado na aproximação da sigmoide, resulta em constantes com valor fracionário que exigem mais de 16 bits para serem implementados em *FPGA*, resultando em maior custo computacional.

Durante o desenvolvimento da implementação do *SVPWM*, observou-se que o uso de *LUT* na compensação do vetor de referência acarreta na interpolação linear para valores não tabelados. Uma rede neural artificial, no entanto, tem como característica o aprendizado do comportamento da curva por meio dos dados de treinamento. Desta forma, para os valores não tabelados, espera-se que a RNA desempenhe melhor comportamento em comparação a *LUT*.

Para a correta implementação do *ATO* tipo II proposto, a quantidade de bits fracionários armazenado nas variáveis devem estar corretamente ajustadas. De forma a garantir a precisão e robustez exigido na leitura do ângulo. As variáveis que armazenam os ângulos (presente e passado) usados pelo sistema discreto devem apresentar no mínimo 40 bits fracionários para a leitura ser correta, enquanto as constantes da equação das diferenças devem possuir no mínimo 24 bits fracionários. Para demais variáveis 16 bits fracionários devem ser suficientes para representar seus valores.

O grau dos polinômios usados na sigmoide aproximada por *SPLINE* podem gerar ondulação indesejada de magnitude significativa na resposta do *ATO*. Por ouro lado aumentar o grau dos polinômios acarreta em aumento de custo computacional, tanto pela adição de operações matemáticas, quanto pela quantidade de bits necessárias para armazenas as constantes dos polinômios. Nesta dissertação o objetivo foi alcançado ao aproximar a sigmoide com polinômios de grau quatro, com uso de 16 bits fracionários durante a simulação.

A taxa de *clock* usado no *FPGA*, assim como a taxa de aquisição de dados, podem influenciar na leitura do ângulo realizada pelo *ATO* em sistemas embarcados, tornando o sistema instável.

5.1 Trabalhos Futuros

Como trabalho a ser desenvolvido está o teste dos algoritmos *SVPWM* e de leitura de posição do sensor resolver propostos usando uma motor e sensor físicos. Nesta dissertação os sinais gerados pelo motor elétrico foram enviados ao *FPGA* usando a técnica *FIL*. Neste caso, o problema principal seria a implementação puramente em *hardware* e a solução de eventuais problemas relacionados aos sinais gerados pelo motor elétrico.

Outro trabalho a ser desenvolvido, corresponde a otimização de redes neurais implementadas em *FPGA*. Os parâmetros usados pela rede exigem armazenamento no dispositivo, de forma que o custo computacional escalona de acordo com a complexidade da rede implementada.

REFERÊNCIA BIBLIOGRÁFICA

ALEKSANDER, I.; MORTON, H. **An introduction to neural computing**. 1. ed.: Chapman and Hall, v. 1, 1990. ISBN 0412377802.

ALOUANE, A. et al. Efficient FPGA-based real-time implementation of an SVPWM algorithm for a delta inverter. **IET Power Electronics**, 11, 9 Agosto 2018. 1611-1619.

ALTERA. FPGA-based control for electric vehicle and hybrid electric vehicle power electronics., 2013. Disponivel em: <http://www.altera.com/literature/wp/wp-01210-electric-vehicles.pdf>. Acesso em: 17 Junho 2019.

ALTERA. Eletric Veichles, 2019. Disponivel em: <https://www.intel.com/content/www/us/en/automotive/products/programmable/electric-vehicles.html>. Acesso em: 28 Outubro 2019.

ARMATO, A. et al. Low-error approximation of artificial neuron sigmoid function and its derivative. **Eletronic Letters**, v. 45, p. 1082-1084, 8 Outubro 2009. ISSN 0013-5194.

BAHL, A. German Neuroinformatics Node. **Evolutionary Multi-Objective Optimization (EMOO)**, 2012. Disponivel em: <https://projects.g-node.org/emoo/>. Acesso em: 10 Fevereiro 2020.

BANZHAF, W. et al. **Genetic Programming - An Introduction**. 1. ed. São Francisco: Morgan Kaufmann Publishers, Inc, v. 1, 1998. ISBN 1-55860-510-X.

BUROVA, I. G.; VARTANOVA, A. A. **Interval Estimation Of Polynomial Splines of the Fifth Order**. 2017 Fourth International Conference on Mathematics and Computers in Sciences and in Industry (MCSI). Corfu, Greece: IEEE. 2017. p. 293 - 297.

CULLEN, J.; GERBETH, A.; DOROJEVETS, M. **FPGA-based Satisfiability Filters for Deep Packet Inspection**. 2018 IEEE Long Island Systems, Applications and Technology Conference (LISAT). Farmingdale:. 2018. p. 1-4.

DYNAPAR. Dynapar: Encoders Incrementais, Absolutos e Resolvers: como escolher a melhor opção? **Dynapar**, 2018. Disponivel em: <https://www.dynaparencoders.com.br/blog/encoder-incremental-absoluto-resolver/>. Acesso em: 08 Janeiro 2020.

GARCÍA, R. C. **Controle de Velocidade de Motor Síncrono de Ímã Permaneente Utilizando Redes Neurais Artificiais e Multiplexação em Frequência**. Rio de Janeeiro: [s.n.], 2015.

GARCÍA, R. C. et al. **Simplification of the Acquisition System for Sensored Vector Control using Resolver Sensor based on FDM and Current Synchronous Sampling.** 2018 IEEE 4th Southern Power Electronics Conference (SPEC). 2018.

GOLDBERG, D. E. **Genetic Algorithms in Search, Optimization and Machine Learning**. 1. ed.: Addison-Wesley Publishing Company , Inc, v. 1, 1989. ISBN 0-201-15767-5.

GREENBERG, M. D. **Advanced Engineering Mathematics**. 2ª. ed. Upper Saddle River: Prentice Hall, 1998. ISBN ISBN 0-13-321431-1.

HAYKIN, S. **Neural Networks:** A Comprehensive Foundation. 2. ed.: Prentice Hall, Inc, v. 1, 2009. ISBN 81-7808-300-0.

HAYKIN, S.; VEEN, B. V. **Signals and Systems**. [S.l.]: John Wiley and sons, inc, 2003.

INTEL. What is FPGA? Programming and FPGA basics, 2020. Disponivel em: <https://www.intel.com.br/content/www/br/pt/products/programmable/fpga/new-to-fpgas/resource-center/overview.html>. Acesso em: 12 Julho 2020.

IQBAL, A. et al. **Matlab/Simulink Model of Space Vector PWM for Three-Phase Voltage Source Inverter**. Proceedings of the 41st International Universities Power Engineering Conference. Newcastle-upon-Tyne:. 2006. p. 1096-1100.

KAEWJINDA, W.; KONGHIRUN,. **A DSP – Based Vector Control of PMSM Servo Drive Using Resolver Sensor.** TENCON 2006 - 2006 IEEE Region 10 Conference. Hong Kong: 2006. p. 1-4.

KITAZAWA, K. Development and commercialization of VR-type resolver system for onboard mouniting in hybrid vehicles, 2006. Disponivel em: <http://www.jmf.or.jp/monodzukuri/english/world/11.html>. Acesso em: 17 Junho 2019.

KUMAR, R.; DAS, S. **A modified approach to both conventional and ANN based SVPWM controllers for voltage fed inverter in sensorless vector control IM drive**. IEEE International Conference on Power Electronics, Drives and Energy Systems. Mumbai, India: IEE. 2014. p. 1 - 6.

LATHI, B. P. **Linear Systems and Signals**: Berkleley-Cambridge Press, 1992.

LIU, J. et al. **A new algorithm research and simulation for permanent magnet synchronous motor ac servo system**. 2008 IEEE Conference on Robotics, Automation and Mechatronics. Chengdu, China: 2008.

MATHWORKS. **FPGA-in-the-Loop Simulation**, 2020. Disponivel em: <https://www.mathworks.com/help/hdlverifier/ug/fpga-in-the-loop-fil-simulation.html>. Acesso em: 12 Julho 2020.

MCNAMEE, P. Automated Memoization in C++. **Paul McNamee's Homepage**, 21 Agosto 1998. Disponivel em: <http://pmcnamee.net/c++-memoization.html>. Acesso em: 23 Agosto 2019.

MITCHELL, M. **An Introduction to Genetic Algorithms**. Cambridge: MIT Press, 1996.

MITCHELL, T. M. **Machine Learning**. 1. ed.: McGraw-Hill Science, v. 1, 1997. ISBN 0070428077.

MONMASSON, E. et al. FPGAs in Industrial Control Applications. **IEEE Transactions on Industrial Informatics**, 7, Maio 2011. pp 224-243.

RAMIREZ, J.; PROAÑO, V.; DARWIN, A. **Basic Programmable Logic controller with FPGA and Artificial Neural Networks.** IEEE Colombian Conference on Communication and Computing (IEEE COLCOM 2015). Popayan: 2015. p. 1-4.

RASHID, M. H. **Power Eletronics Handbook**. San Diego: Academic Press, 2001.

SAIDI, H. et al. **Three phase inverter using SVPWM method for solar electric vehicle.** IREC2015 The Sixth International Renewable Energy Congress. Sousse, Tunisia: IEEE. 2015. p. 1- 5.

SHARMA, D.; BHAT, A. H.; AHMAD, A. **ANN Based SVPWM for Three-Phase Improved Power Quality Converter Under Disturbed AC Mains**. 2017 6th International Conference on Computer Applications In Electrical Engineering-Recent Advances (CERA). Roorkee, India: 2017.

SOARES, A. M. **Implementação Do Controle Vetorial Do Motor De Indução Via Redes Neurais Artificiais**, Campo Grande, 2006.

TAO, C. **Functional safety concept design of hybrid electric vehicle following ISO 26262**. IEEE Transportation Electrification Conference and Expo. Beijing: Agosto. 2014. p. pp. 1-6.

WU, F.; WAN, S.-M.; HUANG, S.-H. **Unity power factor control for PMSM position sensorless drive**. 2008 International Conference on Electrical Machines and Systems. Wuhan, China: 2008. p. 1618-1620.

ZAPATÁN, R. A. et al. **Implementation of a virtual velocity sensor for a DC motor through artificial neural networks in a FPGA system**. 2015 CHILEAN Conference on Electrical, Electronics Engineering, Information and Communication Technologies (CHILECON). Santiago: 2015. p. 1-5.

APÊNDICE 1: *VHDL* PARA A SIGMOIDE

```
-- ==================================
--                                Bibliotecas
-- ==================================
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.fixed_float_types.all;
use ieee.fixed_pkg.all;
-- ==================================
--                                Portas
-- ==================================

entity sigmoide is port (
        cont            : in std_logic_vector(3 downto 0); -- contador sig
        r               : in std_logic; -- Reset
        u               : in std_logic_vector(20 downto 0);
        Y               : out std_logic_vector(17 downto 0)
);
end sigmoide;

architecture arq of sigmoide is

-- ==================================
--                             Variaveis
-- ==================================

        signal up1          : sfixed(4 downto -16); -- entrada pnt fixo 1
        signal up2          : sfixed(4 downto -16); -- entrada pnt fixo 2
        signal yp1          : sfixed(1 downto -16); -- saida pnt fixo 1
        signal yp2          : sfixed(1 downto -16); -- saida pnt fixo 2

        signal k            : sfixed(2 downto 0); -- Constante

        signal L3 : sfixed(2 downto -16);
        signal L4 : sfixed(2 downto -16);

        signal a31          : sfixed(1 downto -16);
        signal a32           : sfixed(1 downto -16);
        signal a33           : sfixed(1 downto -16);
        signal a34           : sfixed(1 downto -16);
        signal a35           : sfixed(1 downto -2);

        signal a41           : sfixed(1 downto -16);
        signal a42           : sfixed(1 downto -16);
        signal a43           : sfixed(1 downto -16);
        signal a44           : sfixed(1 downto -16);
        signal a45           : sfixed(1 downto -16);
```

```
signal a51              : sfixed(1 downto -16);
signal a52              : sfixed(1 downto -16);
signal a53              : sfixed(1 downto -16);
signal a54              : sfixed(1 downto -16);
signal a55              : sfixed(1 downto -16);

begin

        -- ================================
        --                  Conversao para pnt fixo
        -- ================================

        up1 <= to_sfixed(u,up1);

        process (cont,r) is
        begin
                if (r = '1') then -- Reset
                Y <= (others => '0');
                yp1 <= (others => '0');
                up2 <= (others => '0');

                -- ================================
--                                      Nós de Transicao
                -- ================================

                L3 <= to_sfixed(1.498031040981520,L3);
                L4 <= to_sfixed(3.8958933527845554,L4);

                -- ================================
--                                      Constantes
                -- ================================

                k <= to_sfixed(1,k);

                a31 <= to_sfixed(0.004852294921875,a31);
                a32 <= to_sfixed(-0.025527954101563,a32);
                a33 <= to_sfixed(0.002090454101563,a33);
                a34 <= to_sfixed(0.249618530273438,a34);
                a35 <= to_sfixed(0.5, a35);

                a41 <= to_sfixed(-0.000442504882813,a41);
                a42 <= to_sfixed(0.010818481445313,a42);
                a43 <= to_sfixed(-0.094146728515625,a43);
                a44 <= to_sfixed(0.365737915039063,a44);
                a45 <= to_sfixed(0.446441650390625,a45);

                a51 <= to_sfixed(-0.000122070312500,a51);
                a52 <= to_sfixed(0.003433227539063,a52);
                a53 <= to_sfixed(-0.036605834960938,a53);
                a54 <= to_sfixed(0.176330566406250,a54);
```

```
                a55 <= to_sfixed(0.673919677734375,a55);

        else

                if (cont = 0) then
                        if (up1 >= 0) then
                                up2 <= resize(up1,up2); -- referencia igual a entrada
                        else
                                up2 <= resize(-up1,up2); -- referemcia é invertida
                        end if;
                end if;

                if (cont = 1) then
                        if ((0 <= up2) and (up2 < L3)) then -- regiao entre L2 e L3
yp1<=resize(((a31*up2*up2*up2*up2)+(a32*up2*up2*up2)+(a33*up2*up2)+(a34*up2)+a35),yp1);
                        elsif ((L3 <= up2) and (up2 < L4)) then -- regiao entre L3 e L4
yp1<=resize(((a41*up2*up2*up2*up2)+(a42*up2*up2*up2)+(a43*up2*up2)+(a44*up2)+a45),yp1);
                        elsif ((L4 <= up2) and (up2 < 8)) then -- regiao entre L4 e 8
yp1<=resize(((a51*up2*up2*up2*up2)+(a52*up2*up2*up2)+(a53*up2*up2)+(a54*up2)+a55),yp1);
                        elsif (up2 >= 8) then -- regiao acima de 8
                                yp1 <= to_sfixed(1, yp1);
                        end if;
                end if;

                if (cont = 2) then
                        if (up1>=0) then
                                yp2 <= yp1;
                        else
                                yp2 <= resize((k-yp1),yp2);
                        end if;
                end if;

                if (cont = 3) then
                        Y <= to_slv(yp2);
                end if;

        end if;
        end process;
end arq;
```

APÊNDICE 2: *VHDL* PARA O VETOR DE REFERENCIA

```
-- ==================================
--                  Bibliotecas
-- ==================================
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use ieee.fixed_float_types.all;
use ieee.fixed_pkg.all;

-- ==================================
--                    Portas
-- ==================================
entity fc is port (
        CLOCK_50        : in std_logic; -- Clock
        RST             : in std_logic; -- Reset
        Vr              : in std_logic_vector(27 downto 0);
        V               : out std_logic_vector(31 downto 0)
);
end fc;

architecture arq of fc is

        -- ==================================
        --                  Componente
        -- ==================================
        component sigmoide is port (
                cont            : in std_logic_vector(5 downto 0); -- contador sigmoide
                r               : in std_logic; -- Reset
                u               : in std_logic_vector(26 downto 0);
                Y               : out std_logic_vector(17 downto 0)
        );
        end component;

        -- ==================================
        --                  Variaveis
        -- ==================================
        signal cont_g                   : std_logic_vector(5 downto 0); -- Contador Global
        signal Vr_p                     : sfixed(11 downto -16); -- Vr em pnt fixo
        signal V_p                      : sfixed(15 downto -16); -- V em pnt fixo

        signal Vr_min                   : sfixed(11 downto 0); -- Vr_min
        signal V_min                    : sfixed(11 downto 0); -- V_min
        signal k1                       : sfixed(1 downto -16); -- 1/(Vr_max-Vr_min)
        signal k2                       : sfixed(14 downto -1); -- (V_max - Vmin)/2 (range)
        signal k3                       : sfixed(1 downto 0); -- constante 3
        signal Vr_n                     : sfixed(1 downto -16); -- Vr normalizado
        signal V_n                      : sfixed(1 downto - 16); -- V normalizado
```

```
signal cont_s                   : std_logic_vector(5 downto 0); -- contador Sigmoide

signal x111                     : sfixed(10 downto -16); -- Entrada 1 Neuronio 1 Layer 1 (sigmoide, pnt fixo)
signal x121                     : sfixed(10 downto -16); -- Entrada 1 Neuronio 2 Layer 1
signal x131                     : sfixed(10 downto -16); -- Entrada 1 Neuronio 3 Layer 1
signal x141                     : sfixed(10 downto -16); -- Entrada 1 Neuronio 4 Layer 1
signal x151                     : sfixed(10 downto -16); -- Entrada 1 Neuronio 5 Layer 1

signal x111_u                   : std_logic_vector(26 downto 0); -- Entrada 1 Neuronio 1 Layer 1 (sigmoide)
signal x121_u                   : std_logic_vector(26 downto 0); -- Entrada 1 Neuronio 2 Layer 1
signal x131_u                   : std_logic_vector(26 downto 0); -- Entrada 1 Neuronio 3 Layer 1
signal x141_u                   : std_logic_vector(26 downto 0); -- Entrada 1 Neuronio 4 Layer 1
signal x151_u                   : std_logic_vector(26 downto 0); -- Entrada 1 Neuronio 5 Layer 1

signal x112                     : sfixed(1 downto -16); -- Entrada 1 Neuronio 1 Layer 2 (pnt fixo)
signal x212                     : sfixed(1 downto -16); -- Entrada 2 Neuronio 1 Layer 2
signal x312                     : sfixed(1 downto -16); -- Entrada 3 Neuronio 1 Layer 2
signal x412                     : sfixed(1 downto -16); -- Entrada 4 Neuronio 1 Layer 2
signal x512                     : sfixed(1 downto -16); -- Entrada 5 Neuronio 1 Layer 2

signal x112_y                   : std_logic_vector(17 downto 0); -- Entrada 1 Neuronio 1 Layer 2
signal x212_y                   : std_logic_vector(17 downto 0); -- Entrada 2 Neuronio 1 Layer 2
signal x312_y                   : std_logic_vector(17 downto 0); -- Entrada 3 Neuronio 1 Layer 2
signal x412_y                   : std_logic_vector(17 downto 0); -- Entrada 4 Neuronio 1 Layer 2
signal x512_y                   : std_logic_vector(17 downto 0); -- Entrada 5 Neuronio 1 Layer 2

signal w111                     : sfixed(9 downto -16); -- Peso 1 Neuronio 1 Layer 1
signal w121                     : sfixed(7 downto -16); -- Peso 1 Neuronio 2 Layer 1
signal w131                     : sfixed(5 downto -16); -- Peso 1 Neuronio 3 Layer 1
signal w141                     : sfixed(3 downto -16); -- Peso 1 Neuronio 4 Layer 1
signal w151                     : sfixed(5 downto -16); -- Peso 1 Neuronio 5 Layer 1

signal w112                     : sfixed(7 downto -16); -- Peso 1 Neuronio 1 Layer 2
signal w212                     : sfixed(1 downto -16); -- Peso 2 Neuronio 1 Layer 2
signal w312                     : sfixed(3 downto -16); -- Peso 3 Neuronio 1 Layer 2
signal w412                     : sfixed(1 downto -16); -- Peso 4 Neuronio 1 Layer 2
signal w512                     : sfixed(1 downto -16); -- Peso 5 Neuronio 1 Layer 2

signal b11                       sfixed(9 downto -16); -- Bias 1 Layer 1
signal b21                      : sfixed(7 downto -16); -- Bias 2 Layer 1
signal b31                      : sfixed(5 downto -16); -- Bias 3 Layer 1
signal b41                      : sfixed(2 downto -16); -- Bias 4 Layer 1
signal b51                      : sfixed(5 downto -16); -- Bias 5 Layer 1

signal b12                      : sfixed(2 downto -16); -- Bias 1 Layer 2

begin
        -- ================================
        --                Portmap
        -- ================================
```

```
pm1: sigmoide port map(
		cont		=> cont_s,
		r			=> RST,
		u			=> x111_u,
		Y			=> x112_y
);

pm2: sigmoide port map(
		cont		=> cont_s,
		r			=> RST,
		u			=> x121_u,
		Y			=> x212_y
);

pm3: sigmoide port map(
		cont		=> cont_s,
		r			=> RST,
		u			=> x131_u,
		Y			=> x312_y
);

pm4: sigmoide port map(
		cont		=> cont_s,
		r			=> RST,
		u			=> x141_u,
		Y			=> x412_y
);

pm5: sigmoide port map(
		cont		=> cont_s,
		r			=> RST,
		u			=> x151_u,
		Y			=> x512_y
);

-- ==================================
--		Conversao para pnt fixo
-- ==================================
Vr_p <= to_sfixed(Vr,Vr_p);

process (CLOCK_50,RST) is
begin

		if (RST = '1') then
			-- ==================================
			--			Reset
			-- ==================================

			cont_g	<= (others =>'0');
			cont_s	<= (others =>'0');
			V_p			<= (others =>'0');
```

```
		-- ==================================
		--                Constantes
		-- ==================================
		Vr_min	<= to_sfixed(573,Vr_min);
		V_min	<= to_sfixed(573,V_min);
		k1	<= to_sfixed(0.031446540880503,k1); -- 2/(Vr_max-Vr_min)
		k2	<= to_sfixed(6078.5,k2); --(V_max - Vmin)/2 (range)
		k3	<= to_sfixed(-1,k3);

		w111	<= to_sfixed(200.19725025832179,w111);
		w121	<= to_sfixed(-63.126769968307045,w121);
		w131	<= to_sfixed(-9.8056007176635127,w131);
		w141	<= to_sfixed(-4.3460451248472198,w141);
		w151	<= to_sfixed(15.317760924802784,w151);

		b11	<= to_sfixed(-203.57715679689417,b11);
		b21	<= to_sfixed(61.615290882940307,b21);
		b31	<= to_sfixed(12.068423577031799,b31);
		b41	<= to_sfixed(2.6703514455600357,b41);

		b51	<= to_sfixed(10.383761257387681,b51);

		w112	<= to_sfixed(39.666957117703888,w112);
		w212	<= to_sfixed(-0.37465469153382369,w212);
		w312	<= to_sfixed(-3.2912146386505436,w312);
		w412	<= to_sfixed(-0.1143093786332272,w412);
		w512	<= to_sfixed(0.003512522457469548,w512);

		b12	<= to_sfixed(2.7800486171357868,b12);

	elsif (rising_edge(CLOCK_50)) then
		-- ==================================
		--                Contadores
		-- ==================================

		if (cont_g = 12) then -- contador geral
			cont_g <= (others=>'0'); -- reseta contador geral
		else
			cont_g <= cont_g + 1; -- incrementa contador geral
		end if;

		if ((cont_g >= 3) and (cont_g < 8)) then -- contador da sigmoide

			if (cont_s = 5) then
				cont_s <= (others=>'0'); -- reseta contador sigmoide
			else
				cont_s <= cont_s + 1; -- incrementa contador sigmoide
			end if;
		end if;
		-- ==================================
```

```
if (cont_g = 1) then -- Normalizacao

        Vr_n <= resize((((Vr_p-Vr_min)*k1)+k3),Vr_n);

end if;

if (cont_g = 2) then
        -- ================================
        --                RNA Layer 1
        -- ================================
        x111 <= resize(((Vr_n*w111)+b11),x111);
        x121 <= resize(((Vr_n*w121)+b21),x121);
        x131 <= resize(((Vr_n*w131)+b31),x131);
        x141 <= resize(((Vr_n*w141)+b41),x141);
        x151 <= resize(((Vr_n*w151)+b51),x151);


end if;

if (cont_g = 3) then
        x111_u <= to_slv(x111);
        x121_u <= to_slv(x121);
        x131_u <= to_slv(x131);
        x141_u <= to_slv(x141);
        x151_u <= to_slv(x151);

end if;

if (cont_g = 8) then
        x112 <= to_sfixed(x112_y,x112);
        x212 <= to_sfixed(x212_y,x212);
        x312 <= to_sfixed(x312_y,x312);
        x412 <= to_sfixed(x412_y,x412);
        x512 <= to_sfixed(x512_y,x112);

end if;

if (cont_g = 9) then
        -- ================================
        --                RNA Layer 2
        -- ================================
V_n<=resize(((x112*w112)+(x212*w212)+(x312*w312)+(x412*w412)+(x512*w512)+b12),V_n);

end if;

if (cont_g = 10) then -- desnormalizacao
        V_p <= resize((((V_n-k1)*k2)+V_min),V_p);

end if;
```

```
                    if (cont_g = 11) then -- saida
                        V <= to_slv(V_p); -- envia para a saida
                    end if;

                end if;
        end process;
end arq;
```