DESENVOLVIMENTO DE UM SOFTWARE UTILIZANDO UM SISTEMA HÍBRIDO COM LÓGICA *FUZZY* E CONTROLE PID EM INVERSORES

RODRIGO DOS SANTOS TORRALBO

CAMPO GRANDE

2012

UNIVERSIDADE FEDERAL DE MATO GROSSO DO SUL
PROGRAMA DE PÓS-GRADUAÇÃO
EM ENGENHARIA ELÉTRICA

Desenvolvimento de um software utilizando um sistema híbrido com lógica *fuzzy* e controle PID em inversores

Dissertação apresentada ao Programa de Mestrado em Engenharia Elétrica – Área de Concentração: Eletrônica de Potência – da Universidade Federal de Mato Grosso do Sul, como parte para obtenção do título de mestre em Engenharia Elétrica, elaborada sob orientação do Profº Dr. Valmir Machado Pereira.

RODRIGO DOS SANTOS TORRALBO

CAMPO GRANDE

2012

Desenvolvimento de um software utilizando um sistema híbrido com lógica *fuzzy* e controle PID em inversores

RODRIGO DOS SANTOS TORRALBO

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

_____

Valmir Machado Pereira, Dr.
Orientador

_____

Luciana Cambraia Leite, Dra.
Coordenadora do Programa de Pós-Graduação em Engenharia Elétrica

Banca examinadora:

_____

Valmir Machado Pereira, Dr.
Presidente

_____

Edson Antônio Batista, Dr.

_____

Cristiano Quevedo Andrea, Dr.

_____

Paulo Irineu Kolterman, Dr.

À Deus, minha esposa Suellen,
minha mãe Leila, meu irmão Fernando
e especialmente meu pai Alfredo.

Agradecimentos

Primeiramente agradeço a Deus, por me ajudar nesse longo caminho.

Ao meu pai que mesmo longe sei que tem cuidado de mim, a minha mãe que não me deixou desistir em nenhum etapa da vida, ao meu irmão que sempre me auxiliou em tudo que precisei e especialmente a minha esposa amada que sem a qual eu não teria chegado onde cheguei.

Ao professor Dr. Valmir Machado Pereira pelo apoio e pelas contribuições que foram fundamentais para o desenvolvimento deste trabalho.

Agradecimento especial à UFMS por ceder seu laboratório e kit de elétrica no decorrer do desenvolvimento do projeto e ao CNPQ pelos auxílios financeiros concedidos. Não podemos deixar de agradecer também os professores Dr. Edson Batista, Dr. Nicolau Pereira Filho e Dr. Luciana Cambraia Leite , pelas sugestões e por todo conhecimento fornecido para que esse trabalho pudesse ser concluído.

Aos alunos do PPGEE/UFMS, e aos meus grandes amigos e incentivadores Marcelo Cristiano Junior, Diego Henrique Franco, Carlos Vinícius Vanti, Sérgio Vianna Farias e Leandro Elias Basmage.

Resumo da Dissertação apresentada a UFMS como parte dos requisitos necessários para a obtenção do grau de Mestre em Engenharia Elétrica.

DESENVOLVIMENTO DE UM SOFTWARE UTILIZANDO UM SISTEMA HÍBRIDO COM LÓGICA *FUZZY* E CONTROLE PID EM INVERSORES

RODRIGO DOS SANTOS TORRALBO

Dezembro/ 2012

Orientador: Dr. Valmir Machado Pereira.

Área de Concentração: Eletrônica de Potência.

Palavras-chave: PID, Inversores, Software, Máquinas elétricas, Lógica Fuzzy.

Número de Páginas: 93

RESUMO: Atualmente, a grande maioria dos sistemas de controle industriais utiliza o controlador PID (Proporcional, Integral e Derivativo) convencional. Neste tipo de controle os parâmetros de ganhos são fixos durante toda a operação, tornando esse controlador ineficiente para o controle de sistemas que possuem distúrbios ou fatores desconhecidos durante a operação. Por outro lado, a lógica Fuzzy dispensa o conhecimento da modelagem matemática do processo controlado, sendo composta por regras que sintetizam o conhecimento e a experiência do engenheiro de controle. Diante desse contexto, o objetivo principal deste trabalho é desenvolver um software para utilização de um controlador híbrido, composto do controlador PID existente em inversores de tensão utilizados para alimentação de motores de indução acrescido de um sistema Fuzzy. O software proposto é utilizado para ajustar os ganhos do controlador PID, conseguindo assim um refinamento no controle de velocidade do motor de indução trifásico. A comunicação entre o software e o inversor é realizada através da porta serial do computador, permitindo que a aplicação possa ser adaptada para ser utilizada em outros inversores. Antes da implementação do software realiza-se a simulação do sistema proposto, para análise e comparação dos resultados obtidos. A ferramenta computacional Matlab/Simulink®, através do Fuzzy Logical Toolbox, foi utilizada para a implementação do controle Fuzzy. Os resultados obtidos através das simulações foram satisfatórios considerando a comparação com o sistema utilizando PID convencional, sendo a principal contribuição da dissertação o desenvolvimento de um software que altera os ganhos do controle PID em um inversor de tensão comercial.

Abstract of Dissertation presented to UFMS as a partial fulfillment of the requirements for the degree of Master in Electrical Engineering.

DEVELOPMENT OF SOFTWARE USING A HYBRID SYSTEM WITH FUZZY LOGIC AND PID CONTROL IN INVERTERS

RODRIGO DOS SANTOS TORRALBO

December/ 2012

Advisor: Dr. Valmir Machado Pereira.

Area of Concentration: Power Eletronic.

Keywords: Inverters, Programming Language, Eletric Machines, Fuzzy.

Number of Pages: 93

ABSTRACT: Presently, the great majority of industrial control systems use a conventional PID controller (Proportional, Integral and Derivative). On this type of control, the PID parameters or gains do not change during the entire operation, which makes this controller not effective when used on systems with disturbances or unknown factors during operation. Furthermore, the fuzzy logic does not require knowledge of the controlled process mathematical modeling. It consists of rules that synthesize the knowledge and experience of the control engineer. Given this context, the main objective of this work is to develop software using a hybrid controller. This controller is composed of the PID controller from the voltage inverter used in induction motors supply and of a Fuzzy System. The proposed software is used to adjust the PID controller gains, reaching an improvement in the speed control of three-phase induction motors. The communication between the software and the inverter is performed via the computer serial port, allowing the application to be adapted in order to be used in other inverters. Before implementing the software, it is necessary to simulate the proposed system for analysis and comparison of the results obtained. The Fuzzy Logical Toolbox from Matlab/Simulink® was used for the implementation of the fuzzy control. As the main contribution is the development of software that changes the gains of a PID controller in a commercial voltage inverter, the results obtained through the simulations were satisfactory when compared to the system using conventional PID.

Lista de Figuras

Lista de Tabelas

Lista de Siglas

CA: Corrente Alternada

CC: Corrente Contínua

CLP: Controlador Lógico Programável

CSI: *Current Source Inverters* (Inversores Fonte de Corrente)

DMC: *Distributed Measurement and Control*

FTP: *File Transfer Protocol*

GPL: *GNU General Public License*

HTTP: *Hyper Text Transfer Protocol*

IEEE: *Institute of Electrical and Eletronics Engineers*

IHM: Interface Homem Máquina

IP: *Internet Protocol*

Lisp: List Processing

MI: Máquina de Indução

MIT: Motor de Indução Trifásico

NIST: *National Institute of Standards and Technology*

OO: Orientação a Objetos

PID: Proporcional, Integral e Derivativo

Prolog: *Programming Logic*

PWM: *Pulse Width Modulation* (Modulação por Largura de Pulso)

POO: Programação Orientada a Objetos

RNA: Redes Neurais Artificiais

SCADA: *Supervisory Control and Data Aquisition*

TCP: *Transmission Control Protocol*

VSI: *Voltage Source Inverters* (Inversores Fonte de Tensão)

Sumário

1. Introdução

1.1 Apresentação

O cenário globalizado da economia mundial exige que as empresas atinjam padrões crescentes de excelência. Estes novos paradigmas envolvem o binômio de eficiência (baixo custo de produção) e eficácia (produto com alta confiabilidade) (MAGALHÃES; RAMOS; SCHILLING, 1992). Neste sentido os computadores cada vez mais assumem um papel de fundamental importância para a indústria, como no controle dos diversos processos e sistemas inerentes a ela.

Para suprir essa corrida constante por produtos de melhor qualidade e maior quantidade, houve o surgimento da automação industrial, esse setor da economia veio acompanhado de novos desafios, como o menor consumo de energia, menor tempo de produção e maior precisão de ajustes e medidas. Para se vencer esses desafios, técnicas modernas de controle são aplicadas e inventadas a cada dia, isso permite o aprimoramento do controle que não é possível com técnicas convencionais.

A grande maioria dos sistemas de controle existentes utilizam o controlador PID (Proporcional, Integral e Derivativo) convencional. Neste tipo de controle têm-se malhas que geram valores de realimentação, utilizadas para o controle do sistema, que são basicamente analógicas e reguladas de acordo com a saída e o erro. Tradicionalmente no controlador PID os parâmetros de ganhos, proporcional (K*p*), integral (K*i*) e derivativo (K*d*), são fixos durante toda a operação. Consequentemente, esse controlador se torna ineficiente para o controle de sistemas que possuem distúrbios ou fatores desconhecidos durante a operação (SINTHIPSOMBOON; HUNSACHAROONROJ; KHEDARI; PONGAEN; PRATUMSUWAN, 2011).

Atualmente existem outros tipos de controles, utilizados tanto na Engenharia Elétrica, quanto na medicina, agricultura, astronomia, biologia e as mais diversas áreas. Isso se tornou possível graças ao surgimento da Inteligência Artificial que permite trabalhar com sistemas especialistas que utilizam o conhecimento do próprio operador. Esses conhecimentos geralmente são baseados em redes neurais artificiais (RNA), algoritmos genéticos, lógica de primeira ordem e lógica *fuzzy*.

O conhecimento adquirido para implementação da lógica *fuzzy* é adquirido através de um especialista ou por meio de tabelas, esses dados são utilizados para a construção da base de conhecimento que será utilizada pelo controlador, formulando suas regras. Uma das vantagens deste tipo de controlador é que, as entradas podem ser passadas muitas vezes na forma de linguagem natural (ZADEH, 1965).

Sistemas de controle utilizando lógica *fuzzy* têm sido utilizados com sucesso nas mais diversas áreas, tais como: sistemas automobilísticos, aviação, eletrodomésticos, sistemas que auxiliam na

tomada de decisões, sistemas hospitalares, engenharia dos materiais, controle industrial e outros.

Segundo Ortega (2001), Siler e Bucley (2005), Bellman e Zadeh (1970), a utilização de regras difusas e variáveis linguísticas conferem ao sistema de controle várias vantagens, como, por exemplo:

1. Simplificação do modelo do processo;
2. Melhor tratamento das imprecisões inerentes aos sensores utilizadas;
3. Facilidade na especificação das regras de controle, em linguagem próxima da natural;
4. Satisfação de múltiplos objetivos de controle e
5. Facilidade de incorporação do conhecimento de especialistas humanos.

Mas, existe também as desvantagens desse tipo de controle, que são:

1. Necessitam de mais simulação e testes;
2. Não aprendem facilmente;
3. Dificuldades de estabelecer regras corretamente e
4. Não há uma definição matemática precisa.

Porém é necessário adicionar elementos entre o controlador difuso e o processo controlado. Esses elementos são conhecidos como fuzzificador e defuzzificador e estão presentes na entrada e saída do controle. Esses elementos exigem um ajuste fino para o melhor aproveitamento do controlador.

1.2 Justificativa

No setor industrial o processo de produção é realizado de maneira exata e calculada mas, mesmo com o extenso estudo, muitas vezes um sistema encontra dificuldades e imprevistos que podem diminuir a capacidade de operação e em alguns casos causar erros que comprometem toda a produção. A lógica *fuzzy* se torna eficaz no controle de motores devido a sua capacidade de tratar as não linearidades inerentes ao modelo matemático do motor de indução (BADR; ELTAMALY; ALOLAH, 2010).

Segundo Ogata (2003), o modelo de controle PID utiliza uma abordagem matemática muito forte, tornando-se complexo. Esse tipo de controlador é comum ser encontrado na maioria das indústrias dificultando ainda mais o ajuste do sistema normalmente aplicados no acionamento de esteiras, braços mecânicos e diversos outros equipamentos utilizados industrialmente.

Na maioria das vezes uma fábrica possui uma estrutura bem definida de máquinas que trabalham

individualmente, posto que não se pode alterar um processo sem comprometer o outro, assim esse trabalho propõe o desenvolvimento de um software que utilize a lógica *fuzzy* para ajustar os ganhos de um controlador PID, existente em um inversor de potência comercial, que realiza o controle de velocidade de um motor trifásico, sem influenciar significativamente na estrutura existente.

Atualmente diversos trabalhos estão sendo desenvolvidos utilizando os controladores híbridos, isso se deve ao fato de se poder utilizar somente as vantagens de cada controlador e tentar diminuir essas desvantagens, Kalhoodashti e Shahbazian (2011), assim como Tripura e Babu (2011) realizam a simulação de um sistema *fuzzy*-PI, utilizando um inversor de tensão com controle vetorial indireto para o controle de velocidade de um motor de indução.

Wahyunggoro e Saad (2008) realizam uma simulação ajustando os ganhos de um PI, utilizando a lógica *fuzzy*, mas para realizar o controle de um servomotor.

Muruganantham e Palani (2012) desenvolvem um sistema *fuzzy*-PI e realizam o experimento em bancada, para controlar um inversor de um PMBLDCM (Permanent Magnet Brushless DC Motor), assim como Narmadha e Thyagarajan (2010), que realizam o mesmo experimento, mas utilizando um inversor *Sensorless.*

Thao et al. (2010), realizam o controle de um inversor de um célula fotovoltaica utilizando um controlador PID-*fuzzy*, que ajusta os ganhos Kp, Ki e Kd do PID utilizando a lógica *fuzzy*, assim como Pavlica e Petrovacki (1998) que utilizaram o mesmo sistema para controlar duas plantas experimentais de um laboratório.

A grande maioria dos autores realizaram experimentos utilizando o controlador híbrido *fuzzy*-PI, enquanto aqueles que utilizam *fuzzy*-PID não o fazem na intenção de controlar a velocidade de um motor de indução trifásica e sim para controles específicos, o presente trabalho propõe desenvolver um software que realize o controle de velocidade de um motor de indução trifásica utilizando o controlador híbrido *fuzzy*-PID, além de realizar simulações utilizando esse controlador proposto, tanto para um inversor utilizando controle escalar, quanto o controle vetorial.

O presente trabalho faz uso de um inversor de potência e motor de indução trifásico (MIT), muito utilizados industrialmente, e também um computador conectado a porta serial do inversor. O sistema proposto visa ainda facilitar realização de ajustes caso seja necessário, já que o controle *fuzzy* nesse caso se torna mais simples de entender e combater os imprevistos que um sistema eventualmente possua.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo geral dessa dissertação é a elaboração de um software para realizar o controle de velocidade de um motor de indução trifásico, possibilitando analisar uma arquitetura híbrida de um controle fuzzy-PID que utiliza um controlador fuzzy para ajustar os ganhos do controlador PID presente em um inversor de tensão comercial, através da própria porta serial do inversor.

1.3.2 Objetivos Específicos

Como objetivos específicos deste trabalho destaca-se:

a) Estudar as linguagens de programação e ambientes de desenvolvimento existentes;

b) Realizar um estudo dos controladores *fuzzy*;

c) Simular a estratégia de controle projetada usando o software Matlab/Simulink;

d) Realizar estudo do inversor WEG CFW-09;

e) Implementar o software proposto nesse trabalho;

f) Analisar os resultados alcançados a partir da aplicação do controle proposto.

1.4 Estrutura do trabalho

Este trabalho está estruturado em 5 (cinco) capítulos obedecendo a seguinte sequência:

Capítulo 1 - Aborda a introdução sobre o tema destacando os objetivos gerais e específicos.

Capítulo 2 - Apresenta a fundamentação teórica, destacando os estudos realizados sobre motores de indução trifásicos, conversores de frequência (inversores de tensão), estratégias de controle, softwares supervisórios, paradigmas de programação, linguagens de programação e a lógica *fuzzy*.

Capítulo 3 - Apresenta as simulações e análises dos controladores propostos, para implementação do software.

Capítulo 4 – Apresenta o aplicativo computacional (software) desenvolvido e os resultados obtidos.

Capítulo 5 – Apresenta as conclusões realizadas durante o trabalho e sugestões de trabalhos futuros.

2. Fundamentação teórica

2.1 Introdução

Neste capítulo são apresentados os estudos realizados para o desenvolvimento e implementação do software proposto nesse trabalho, em que se podem destacar os seguintes temas: motores de indução trifásicos, inversores de potência, engenharia de software, controle e automação.

2.2 Motores de indução trifásicos

2.2.1 Introdução

O motor de indução também conhecido como motor assíncrono é amplamente difundido e utilizado nas indústrias como máquina motriz, atingindo um índice de 80% a 85% das máquinas motrizes instaladas. O modelo construtivo mais simples e de maior utilização é o motor de indução trifásico de rotor em gaiola, que apresenta rendimento próximo a 90% e possui algumas vantagens com relação às outras máquinas elétricas: baixo custo, robustez construtiva (por não apresentar comutadores, anéis coletores, nem quaisquer contatos móveis entre o rotor e o estator), baixa manutenção e proteção natural contra curto-circuito (PEREIRA, 2001).

São máquinas em que os enrolamentos do estator são alimentados com tensões CA trifásicas equilibradas e que produzem tensões induzidas nos enrolamentos do rotor devido à ação de transformação. Dada a característica trifásica da alimentação do estator e à distribuição espacial dos enrolamentos, o campo produzido pelo estator é girante, ou seja, sua resultante possui um movimento rotacional. O campo produzido pelas correntes induzidas no rotor terá a mesma característica, procurando sempre acompanhar o campo girante do estator, de modo que, da interação de ambos campos magnéticos será produzido o torque que leva a máquina à rotação.

Para a maior parte das aplicações da máquina de indução operando como motor, o modelo clássico em regime permanente é útil. No entanto, quando se deseja simular a máquina de indução como um componente de um sistema de controle em malha fechada e sujeito a perturbações, um estudo do comportamento dinâmico é fundamental (PEREIRA, 2001).

2.2.2 Modelo dinâmico do motor de indução

Na modelagem da máquina de indução é importante levar em consideração alguns aspectos, como é relatado em (PEREIRA; POMILIO, 2001). A análise dinâmica de um motor de indução, com base em suas equações diferenciais em variáveis naturais (ou variáveis da máquina) é bastante complicada devido à variação senoidal das indutâncias mútuas entre os enrolamentos do estator e do

rotor com relação ao deslocamento angular do rotor. Ou seja, quando o rotor gira, esses termos acoplados variam com o tempo. Em consequência, as equações diferenciais das tensões apresentam indutâncias variáveis no tempo (equações diferenciais não-lineares). Por isso, foram desenvolvidas técnicas baseadas em transformações matemáticas que têm por finalidade eliminar essa variação temporal dos coeficientes das equações de estado da máquina em relação à posição angular do rotor (SZCZESNY; RONKOWSKI, 1991) (KRAUSE; WASYNCZUK; SUDHOFF, 1986) (KRAUSE; THOMAS, 1965).

A) Transformação de eixos de referência

Se a máquina de indução é ligada a um sistema trifásico equilibrado, a *transformação trifásica-bifásica*, ou *teoria dos dois eixos ou transformação* ***dq0*** é largamente empregada para estudos dinâmicos (ONG, 1997) (NOVOTNY; LIPO, 1996) (KRAUSE; WASYNCZUK; SUDHOFF, 1986). Essa utilização decorre que a transformação do modelo polifásico para o modelo bifásico permite a utilização de técnicas de controle utilizadas em máquinas de corrente contínua, tornando a máquina de indução dinamicamente equivalente a uma simples máquina de corrente contínua com excitação independente.

Nessa teoria, os parâmetros variáveis no tempo são eliminados e as variáveis e parâmetros são expressos em um sistema de dois eixos ortogonais (ou mutuamente desacoplados): direto (*d*) e quadratura (*q*). A interpretação trigonométrica da mudança de variáveis para um sistema de referência comum que está girando em uma velocidade ω na direção da rotação do rotor é mostrada na Figura 2.1. A transformação é expressa em função do ângulo, θ, entre o eixo *q* e o eixo *as*. O eixo *q* adianta-se do eixo *d* por π/2.

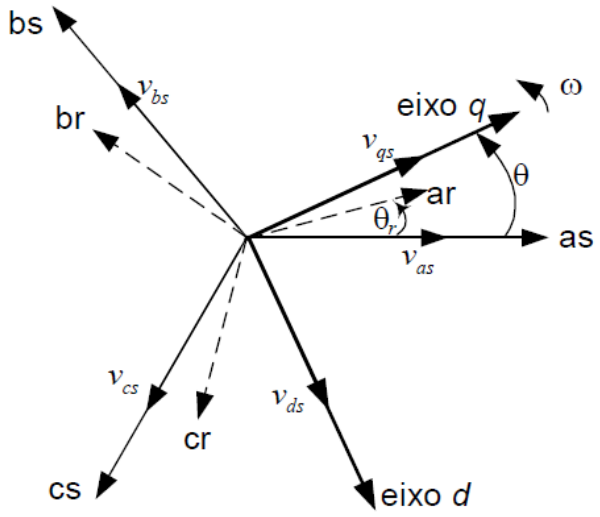


Figura 2.1: Interpretação trigonométrica da mudança de variáveis *abc* para ***dq*** de uma máquina de indução trifásica.

Na equação (2.1), apresenta-se a transformação de *abc* para o eixo de referência ***dq0***:

$$f_{qd0} = K_{qd0} f_{abc} \tag{2.1}$$

Onde $(f_{qd0})^T = [f_{qs} f_{ds} f_{0s}]$ e $(f_{abc})^T = [f_{as} f_{bs} f_{cs}]$.

A variável *f* pode representar as tensões de fase, fluxos concatenados ou correntes da máquina. Os subscritos *q*, *d* e 0 se referem aos eixos quadratura, direto e de sequência zero, respectivamente. Os subscritos *a*, *b* e *c* se referem as variáveis naturais da máquina.

A matriz de transformação $\mathbf{K}_{qd0}$ é definida pela equação (2.2):

$$K_{qd0} = \frac{2}{3}\begin{vmatrix} \cos(\theta) & \cos(\theta - 120^o) & \cos(\theta + 120^o) \\ \sin(\theta) & \sin(\theta - 120^o) & \sin(\theta + 120^o) \\ 0{,}5 & 0{,}5 & 0{,}5 \end{vmatrix} \tag{2.2}$$

e sua inversa é dada pela equação (2.3):

$$(K_{qd0})^{-1} = K_{abc} = \begin{vmatrix} \cos(\theta) & \sin(\theta) & 1 \\ \cos(\theta - 120^o) & \sin(\theta - 120^o) & 1 \\ \cos(\theta + 120^o) & \sin(\theta + 120^o) & 1 \end{vmatrix} \tag{2.3}$$

Para a condição trifásica equilibrada a sequência zero não existe. Esta foi considerada somente para resultar em relações de transformação únicas. É conveniente fazer q = 0, tal que o eixo *q* coincida com o eixo *as*. O modelo dinâmico ***dq*** de uma máquina pode ser expresso em um eixo de referência estacionário ou girante. No eixo estacionário, os eixos de referência *d* e *q* são fixados sobre o estator ($\omega = 0$). O eixo de referência girante pode ser fixado sobre o rotor ($\omega = \omega_r$) ou mover-se na velocidade do campo girante (síncrono) ($\omega = \omega_s$). A vantagem do eixo de referência síncrono está em que, com alimentação senoidal e em regime permanente, as variáveis aparecem como quantidades CC, podendo ser representadas por seus valores eficazes.

B) Representação da máquina de indução em um eixo de referência arbitrário

É conveniente desenvolver as equações para um sistema de eixos de referência com velocidade arbitrária ω e, dessas equações gerais, obter as equações para qualquer referência, podendo esta ser o campo girante, o rotor ou o estator.

Normalmente os parâmetros da máquina são medidos com relação aos enrolamentos do estator. Com as variáveis do rotor referidas para o lado do estator e com a indutância própria separada em componentes de dispersão e magnetização, as equações de tensão podem ser expressas conforme as esquações (2.4), (2.5), (2.6) e (2.7) (ONG, 1997) (KRAUSE; WASYNCZUK; SUDHOFF, 1986)

$$v_{qs} = p\lambda_{qs} . \omega\lambda_{ds} . R_s i_{qs} \tag{2.4}$$

$$v_{qs} = p\lambda_{ds} . \omega\lambda_{qs} . R_s i_{ds} \tag{2.5}$$

$$v'_{qr} = p\lambda'_{qr} + (\omega - \omega_r)\lambda'_{dr} + R'_r i'_{qr} \tag{2.6}$$

$$v'_{dr} = p\lambda'_{dr} + (\omega - \omega_r)\lambda'_{qr} + R'_r i'_{dr} \tag{2.7}$$

sendo que as equações dos fluxos concatenados são definidas conforme as equações (2.8), (2.9), (2.10) e (2.11).

$$\lambda_{qs} = L_{ls} i_{qs} + M(i_{qs} + i'_{qr}) \tag{2.8}$$

$$\lambda_{ds} = L_{ls} i_{ds} + M(i_{ds} + i'_{dr}) \tag{2.9}$$

$$\lambda'_{qr} = L'_{lr} i_{qr} + M(i_{qs} + i'_{qr}) \tag{2.10}$$

$$\lambda'_{dr} = L'_{lr} i_{dr} + M(i_{ds} + i'_{dr}) \tag{2.11}$$

O torque eletromagnético desenvolvido pela máquina em variáveis ***dq*** é dado pela equação 2.12:

$$T_e = (\frac{3}{2})(\frac{P}{2})(\lambda_{ds} i_{qs} - \lambda_{ds} i_{qs}) \tag{2.12}$$

O comportamento dinâmico do sistema eletromecânico é determinado pela equação 2.13:

$$T_{mec} = T_L + B\,\omega_r + \frac{2J}{P}(p\,\omega_r) \quad , \quad T_{mec} = T_e \qquad (2.13)$$

Os apóstrofos denotam quantidades do rotor referidas para o lado do estator. As equações (2.4) – (2.11) sugerem o circuito equivalente mostrado na Figura 2.2. Tem-se que p é o operador d/dt, ω é a velocidade arbitrária em que gira (na direção da rotação do rotor) o sistema de eixos de referência, ω_r é a velocidade angular elétrica equivalente do rotor. Os parâmetros R_s e R'_r são a resistências do estator e do rotor respectivamente referida ao estator. As quantidades L_s, M e L'_r são as indutâncias por fase do estator, de magnetização e do rotor referida ao estator, respectivamente. Em (2.12) – (2.13) P é o número de polos da máquina; J é a constante de inércia do rotor em kg.m^2; B é a constante de atrito rotacional em kg.m^2/s e T_L é o conjugado de carga em Nm. O modelo da máquina de indução representado em (2.4) a (2.13) corresponde ao modo de operação como motor. Para funcionamento como gerador é suficiente considerar o torque da carga T_L em (2.13) como negativo.

Caso não disponíveis, os parâmetros a serem utilizados nas equações anteriores podem ser determinados valendo-se do modelo em regime permanente mediante os ensaios em vazio e de curto-circuito.

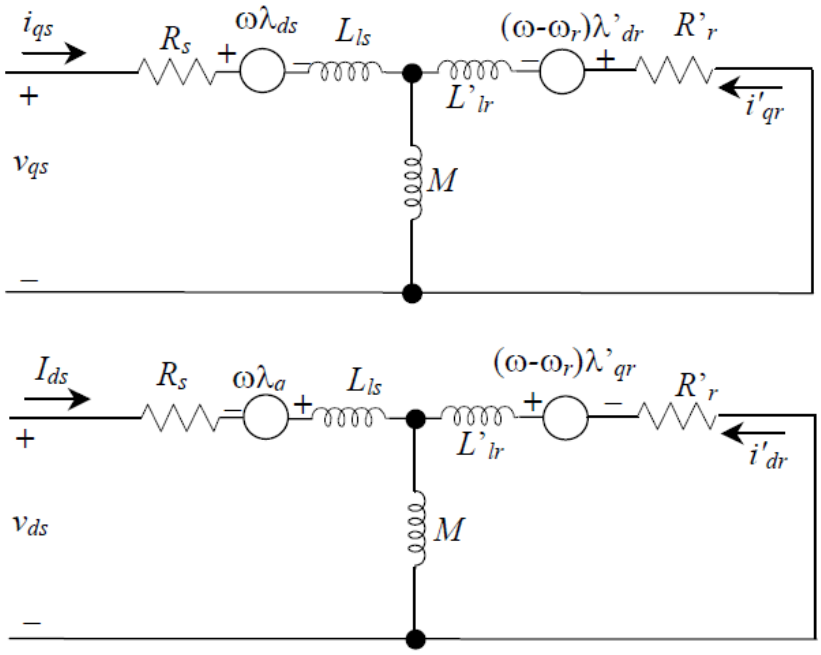


Figura 2.2: *Representação por circuito equivalente do modelo de uma máquina de indução em um eixo de referência arbitrário.*

2.2.3 Modelo da Máquina de Indução em Regime Permanente

O modelo de circuito por fase da máquina de indução, com todos os parâmetros refletidos ao lado do estator é apresentado na Figura 2.3. Pode ser derivado tomando como base o circuito da Figura 2.2 ou partindo das equações das tensões da máquina em regime permanente e resolvendo-as em relação à corrente. O circuito é constituído basicamente pelas resistências e reatâncias de dispersão dos enrolamentos do estator e do rotor (R_s, X_s) e ($R'r$, X'_r), respectivamente. X_m é a reatância de magnetização do entreferro e R_m representa a resistência para as perdas de excitação (ou no núcleo). V_s é a tensão terminal por fase no estator e E_s é tensão de entreferro por fase. O escorregamento s é a velocidade de escorregamento da *f.m.m* do rotor por unidade da velocidade angular elétrica da onda de *f.m.m* do estator (ω_s), o que pode ser expresso pela equação (2.14):

$$s = \frac{\omega_s - \omega_r}{\omega_s} \qquad (2.14)$$

em que ω_r é a velocidade angular elétrica do rotor. Para uma máquina com P pólos, ω_s é definida pela equação (2.15):

$$\omega_s = \frac{2\omega}{P} \qquad (2.15)$$

onde ω é a frequência da rede em rad/s.

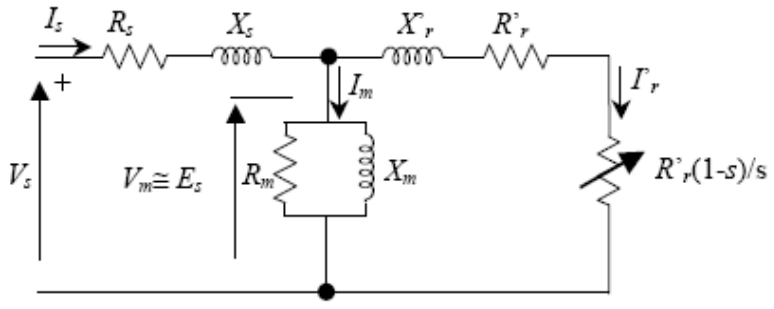


Figura 2.3: *Circuito equivalente por fase de um motor de indução em regime permanente.*

Desprezando as perdas no núcleo, tem-se que a potência transferida eletromagneticamente do estator para o rotor, a chamada potência de entreferro, P_g, é a diferença entre a potência de entrada, P_i, e as perdas no estator, isto é dado pela equação (2.16):

$$P_g = P_i - I_s^2 . R_s [W / fase] \qquad (2.16)$$

Essa é a potência total no rotor dissipada no resistor R'_r/s. Portanto, para uma máquina trifásica têm-se a equação (2.17):

$$P_g = 3I'^2_r \frac{R'_r}{s} \qquad (2.17)$$

A potência total no rotor pode ser dividida em duas componentes:

- Perdas ôhmicas no cobre do rotor (resistência R'_r) são definadas pela equação (2.18):

$$P_j = 3I'^2_r R'_r \qquad (2.18)$$

- Potência que aparece num resistor tendo um valor ôhmico $(1-s)R'_r/s$, que corresponde à carga, ou seja, a potência eletromagnética desenvolvida pela máquina. Então, subtraindo P_j de P_g obtém-se a equação (2.19):

$$P_d = P_g - P_j = (1-s) P_g \qquad (2.19)$$

A potência de perdas rotacionais, P_{rot}, deve ser subtraída de P_d para então se obter a potência mecânica desenvolvida no eixo da máquina P_{mec}, conforme demostrado na equação (2.20):

$$P_{mec} = P_d - P_{rot} \qquad (2.20)$$

Caso as perdas rotacionais não sejam consideradas, pode-se escrever as equações (2.21), (2.22) e (2.23)

$$P_{mec} = (1-s) P_g \qquad (2.21)$$

$$P_{mec} = 3I'^2_r . R'^2_r \frac{(1-s)}{s} \qquad (2.22)$$

O conjugado eletromagnético desenvolvido (T_{mec}) pode ser obtido lembrando que a potência mecânica é igual ao conjugado multiplicado pela velocidade, assim as esquações (2.23), (2.24) e (2.25).

$$P_{mec} = \frac{2}{P} \omega_r T_{mec} \qquad (2.23)$$

$$\omega_r = (1 - s)\omega_s \qquad (2.24)$$

$$T_{mec} = \frac{P}{2} \frac{3}{\omega_r} \frac{R'^2_r}{s} I'^2_r \qquad (2.25)$$

2.3 Inversores

2.3.1 Introdução

A automação surgiu para auxiliar no aumento de produção e redução de custos que a indústria necessita atualmente, com isso houve a necessidade de desenvolvimento e estudo de equipamentos de diversas variedades de aplicação e setores industriais que pudessem ajudar a automação a cumprir seus objetivos. Na automação industrial alguns dos equipamentos mais utilizados são os conversores de frequência ou inversores de tensão, comercialmente denominados “inversores de frequência”, conjuntamente com os CLPs .

Um inversor de tensão basicamente é um dispositivo capaz de converter correntes CC em correntes CA com a frequência e tensão ou corrente desejada, usualmente usado com a finalidade de controlar a velocidade de um motor. Nesse tipo de equipamento a tensão de saída possui uma forma de onda periódica que embora não senoidal, pode ser considerada como tal.

Segundo Ahmed (2001) existem diversos tipos de inversores, dependendo do número de fases, se utilizam ou não semicondutores de potência, de acordo com os príncipios utilizados na comutação e as formas de onda da saída, mas basicamente esses inversores podem ser classificados como inversores fonte de tensão (*voltage source inverters – VSI*) e inversores fonte de corrente (*current source inverters – CSI*). Dentre os diversos tipos pode-se destacar os inversores trifásicos, que são muito utilizados industrialmente, principalmente no controle de velocidade de motores trifásicos.

2.3.2 Componentes principais

O conversor de tensão possui basicamente quatro componentes principais que são: retificador (conversor CA-CC), filtro, bloco conversor e por fim, a unidade microprocessada que permite controlar a amplitude e a frequência da tensão fundamental de saída. Na Figura 2.4 mostra-se o diagrama de blocos do conversor de frequência.

O retificador é composto por pontes de diodo trifásicas ou monofásicas e tem a função de retificar a tensão e a frequência da rede e transformar em corrente contínua. O filtro é constituído por capacitores eletrolíticos com a função de diminuir as ondulações na tensão que foi retificada e garantir o fornecimento de tensão contínua para o inversor. O inversor tem a função de transformar a tensão contínua em tensão alternada com frequência variável.

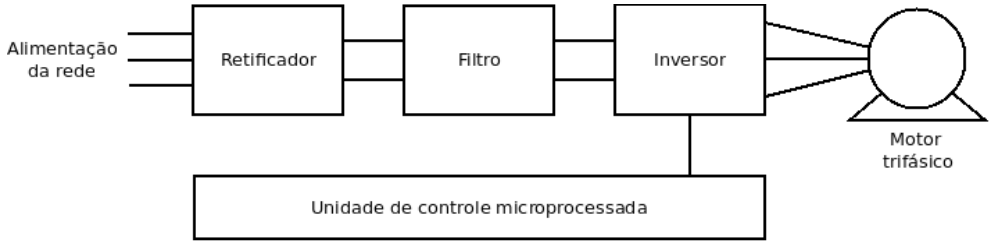


Figura 2.4: *Diagrama de blocos de um conversor de frequência.*

A Figura 2.5 ilustra, através de diagrama simplificado, o conversor de frequência para motores trifásicos.

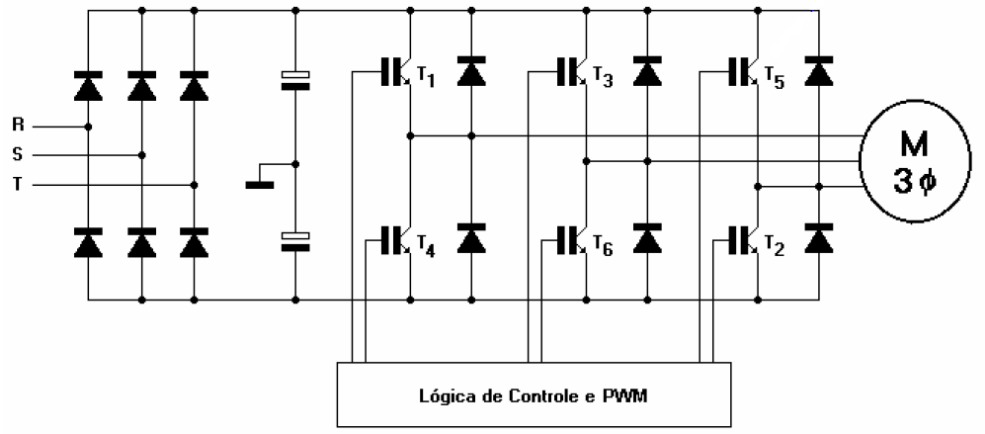


Figura 2.5: *Esquema de um conversor de frequência trifásico. Fonte: Rodrigues (2007)*

Dependendo da combinação das chaves abertas ou fechadas podem-se obter na saída do conversor diferentes formas de ondas. Os conversores utilizam a técnica de chaveamento denominada PWM (*Pulse Width Modulation*) ou modulação por largura de pulso.

Existe uma grande variedade de PWM trifásicos, que permitem a geração de ondas de tensão de

frequência variável. A operação de um inversor PWM é estudada com técnicas usadas para gerar formas de ondas de sinais de disparo. Estas formas de onda são produzidas através da comparação de uma portadora triangular de alta frequência com uma onda de referência de baixa frequência senoidal. O controle da frequência de saída é obtido, variando a taxa de comutação dos dispositivos do inversor. Dessa forma, a tensão de saída e a frequência podem ser rapidamente alteradas dentro do circuito do inversor obtendo uma resposta transiente rápida. O inversor trifásico deve fornecer uma alimentação equilibrada com formas de onda de tensão idênticas, separadas por 120 graus. É essencial, entretanto, que os sinais de disparo de cada perna do inversor, ou meia ponte, sejam idênticos e tenham um deslocamento de fase de 120 graus entre si.

Os conversores de frequência podem ser são divididos de acordo com o tipo de controle, que são: Controle Escalar e Controle Vetorial.

O funcionamento do conversor de frequência com controle escalar baseia-se na estratégia de comando denominada “V/f” constante, que mantém o fluxo do entreferro constante, aproximadamente igual ao nominal, para qualquer velocidade de funcionamento do motor.

O controle vetorial é feito através da decomposição vetorial, possibilitando a variação independente do torque e do fluxo. Possuem alta precisão de controle da velocidade e de torque do motor. Podem ser do tipo normal, que necessita de um motor com sensor de velocidade, e do tipo *sensorlesss*, que não necessita do sensor. Sua precisão é considerada satisfatória, mas com algumas limitações, principalmente em baixas rotações.

Para (GONELLA, 2007) os conversores de frequência com controle escalar são utilizados em sistemas que não requerem muita dinâmica e nem elevada precisão no controle de torque. A precisão para este tipo de controle é de até 0,5% para sistemas sem variação de carga e de 3% a 5% com variação de carga de 0% a 100% do torque nominal.

O circuito de potência do conversor de frequência vetorial não difere do conversor com controle escalar (V/f constante), pois são compostos dos mesmos blocos funcionais, mas possuem estratégias diferentes. No caso escalar a referência de velocidade é usada como sinal para gerar parâmetros de tensão e frequência apropriados e disparar os semicondutores de potência. No caso vetorial, o modelo do motor de indução permite a utilização de técnicas de controle utilizadas em máquinas de corrente contínua, através da transformação do modelo polifásico para o modelo bifásico, tornando a máquina de indução dinamicamente equivalente a uma máquina de corrente contínua com excitação separada (GONELLA, 2007).

2.4 Estratégias de controle

2.4.1 Controle escalar

Dentre as técnicas de controles existentes o controle escalar é a mais comum no controle de velocidade de um motor trifásico, onde o fluxo de escorregamento do entreferro é controlado através da relação constante entre tensão e frequência aplicadas no motor de indução trifásico . A malha é fechada com a medição da velocidade do rotor, possibilitando o controle de torque através do escorregamento (BADR; ELTAMALY; ALOLAH, 2010).

Através do equacionamento do motor de indução trifásico em regime permanente, pode-se concluir que o torque da MI pode ser controlado através do escorregamento. Esta relação é descrita pela equação (2.1).

$$T_e = \frac{s\,|U_m|}{(R'_r/3)\,\omega_e} \tag{2.1}$$

Sendo:

- s : escorregamento;
- R'_r : resistência do rotor;
- ω_e : frequência elétrica do estator;
- T_e : torque elétrico;
- U_m : magnitude da tensão na indutância de magnetização.

Dessa forma, para que o torque seja proporcional ao escorregamento, a razão entre a tensão magnetizante e a frequência elétrica aplicada deve ser mantida constante. O sistema de controle utilizado é mostrado na Figura 2.6.

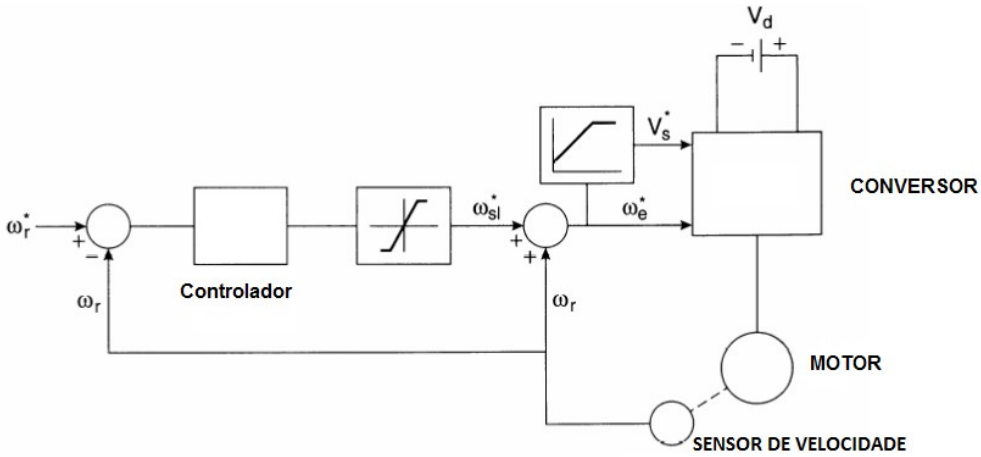


Figura 2.6: *Controle escalar.*

O controle utiliza a medição da velocidade do rotor para cálculo do erro e fechamento da malha. O controlador então determina o valor da velocidade de escorregamento do motor. Esta velocidade é limitada de modo a limitar o torque. Dessa forma, a frequência elétrica é determinada, e a tensão é calculada de modo a manter uma razão constante.

2.4.1 Controle Vetorial

No motor de indução trifásico, pode-se determinar o torque através da amplitude e pela fase da corrente, por isso a denominação controle vetorial. Diversas técnicas de controle vetorial foram desenvolvidas ao longo dos anos, entre elas o método de aceleração do campo e o controle com orientação do campo, inclusive obtendo muito destaque no meio acadêmico e industrial.

No controle por orientação do campo, o torque é controlado indiretamente, ao determinar a decomposição em quadratura para o motor, de tal maneira que o eixo real coincida com o fluxo magnético da máquina (VAS, 1990) (BLASCKE, 1972) (OGASARAWA; AKAGI; NABAE, 1988). Esse tipo de controle acaba sendo sensível as variações de parâmetros do MIT, principalmente a resistência do rotor ou estator (KARANAYIL; MUHAMMED; GRANTHAN, 2005). Nesse caso observadores estimadores de estado podem ser necessários para que os parâmetros sejam determinados, afim de não degradar o desempenho do sistema.

O controle de orientação de campo pode ser realizado tanto com a orientação do fluxo do rotor, quanto com a orientação do fluxo do estator, que possuem desempenhos muito parecidos. Uma das vantagens da orientação do fluxo do estator é a menor sensibilidade a variações dos parâmetros. No entanto não se produz uma relação linear entre o torque e o escorregamento (ROMEU, 2006).

O controle indireto do campo é possível desde que se tenha o conhecimento da velocidade e da posição mecânica do rotor, para isso é possível utilizar a técnicas utilizando sensores para isso ou utilizar a técnica denominada *sensorless* (sem sensores), que possui menor custo em acionamentos

de baixa potência e confiabilidade em acionamentos de potência elevada. (BOLDEA, 2008)

As técnicas de acionamento sem sensores ganharam grande adesão nos últimos anos, devido a eliminação dos sensores mecânicos, e vêm sendo amplamente discutida no meio acadêmico e utilizada com sucesso no meio industrial. No entanto, a observação e estimação da velocidade angular do rotor possui modelos matemáticos complexos, que são difíceis de formular e que exigem grande capacidade computacional para implementação.

Na técnica *sensorless* utiliza-se constantemente estimadores de malha aberta com monitoração de correntes e tensões do estator, observadores de estados, modelos de referência adaptativos e diversas técnicas de inteligência computacional, como lógica *fuzzy*, redes neurais artificiais e sistemas neuro-*fuzzy* (VAS, 1998).

2.5 Softwares supervisórios

Os *drivers* (CLP, inversores e IHM) estão em uma crescente melhora em relação a ruídos, entradas/saídas, processamento, comunicações e no aspecto de software. Neste aspecto diversas empresas desenvolvem softwares específicos para serem utilizados com *drivers*, como por exemplo, Elipse SCADA da Elipse Software, RSView da Rockwell, e WinCC da Siemens. A maioria dos softwares supervisórios são proprietários, ou seja, possuem sistema de comunicação que funcionam especificamente para uma linha de *drivers*. Uma opção atrativa para desenvolver software supervisórios é utilizar plataformas abertas que favoreçam a aplicabilidade com diversos fabricantes de *drivers*. O NIST (National Institute of Standards and Technology) desenvolveu um software supervisório padronizado, ou seja, um modelo de software que pode ser utilizado com diferentes tecnologias. Este software foi desenvolvido com tecnologia Java e especificamente a biblioteca Applet para fornecer informações atualizadas dos estados dos transdutores (sensores de temperatura e válvulas) e da direção do fluxo de água para a refrigeração de um ambiente. Na Figura 2.7, mostra-se a tela do sistema de DMC (*Distributed Measurement and Control*) com supervisão via Internet desenvolvida pelo NIST com base no padrão IEEE 1451.1.

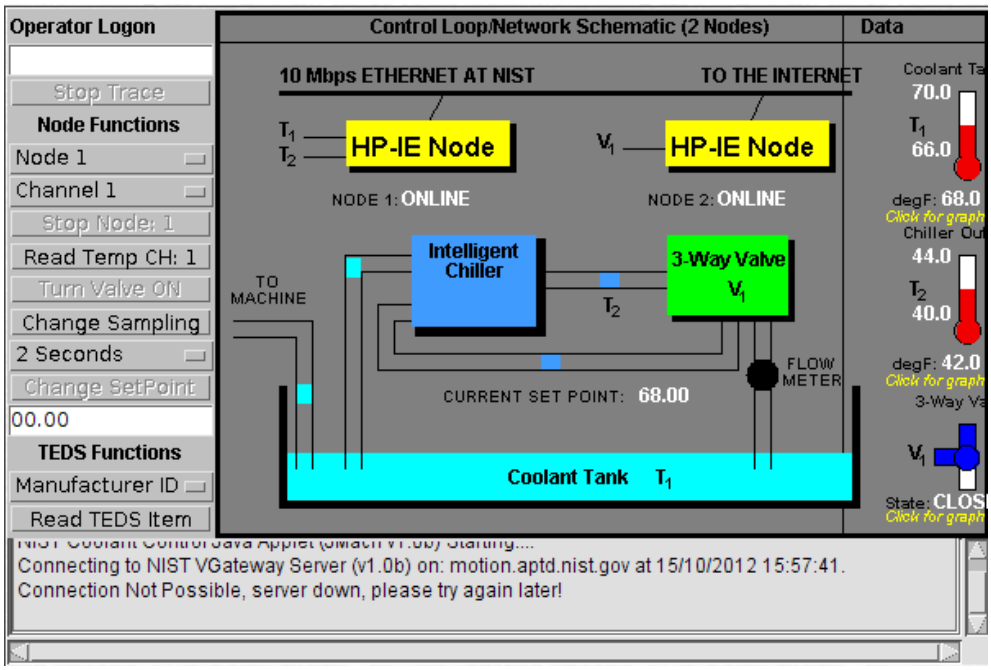


Figura 2.7: *Tela do software supervisório do NIST IEEE 1451. Fonte: http://motion.aptd.nist.gov/P1451/ISADemo.htm (2012)*

A padronização de um software supervisório depende basicamente do protocolo de comunicação que deverá ser utilizado entre o Computador e o *driver*. O protocolo utilizado geralmente com os softwares supervisórios, pode ter acoplado a si equipamentos que possuem inteligência para desempenhar funções específicas de controle tais como PID, controle de fluxo de informações e processos. Podendo ter tempos de transferências longos, deve ser capaz de comunicar-se por vários tipos de dados tais como discretos, analógicos, parâmetros, programas e informações do usuário. (SILVA; CRUZ; ROSADO, 2006). Portanto, os softwares supervisórios, devem proporcionar ao projetista uma ampla gama de comunicação com o andamento dos processos industriais e conter interfaces que favoreçam a real compreensão dos eventos. A interface na qual o projetista interage com o dispositivo, máquina ou qualquer outra ferramenta, vêm se tornando cada vez mais amigável, substituindo os antigos painéis de operações, fazendo com que o sistema automatizado seja mais flexível e produtivo. Devido à necessidade de uma interface amigável entre a máquina (processo indutrial) e o projetista, segundo Moraes e Castrucci (2007), devem-se utilizar dois modos distintos de telas:

- Modo de desenvolvimento: Criação de telas gráficas e animações representativas do processo;
- Modo *Run Time*: Janela animada que mostra o andamento do processo. Deve ser capaz de tomar dados, armazená-los, gerar gráficos de tendências, alarmes, e terem telas desenhadas hierarquicamente e compatíveis com a operação do processo.

De modo geral, o software supervisório deve proporcionar maior precisão e abrangência nas medições, no entanto, as informações mostradas em sua tela são dirigidas ao projetista e devem ser organizadas e com possibilidade de armazenagem das informações, desta forma, a utilização de banco de dados é frequente no desenvolvimento de softwares supervisórios. Na indústria os softwares supervisórios são essenciais na composição de um sistema de Controle Supervisório e Aquisição de dados (SCADA - *Supervisory Control and Data Aquisition*). O sistema SCADA destina-se ao gerenciamento e controle para unidades industriais cujos elementos estejam distribuídos ao longo de grandes distâncias (PUPO, 2002).

Visto que não é apenas um software, e sim um sistema que se enfoca na função de supervisão, se posiciona logo acima do hardware o qual é conectado, geralmente através de CLPs ou em outros módulos de hardware e controle comercial (JOAQUIM, 2006).

O sistema SCADA pode ser aplicado em:

- Oleodutos (produtos químicos e gasosos);
- Distribuição de tratamento de água;
- Sistema de esgoto;
- Linha de distribuição de tratamento de água;
- Linha de tratamento de minério;
- Sistemas de transporte como ferrovias: metrôs, trânsitos em cidades e etc.

Para desenvolver um software supervisório deve-se atentar na plataforma operacional e no paradigma de programação, pois são fatores que podem facilitar o desenvolvimento e tornar mais flexível o aplicativo.

2.6 Paradigmas de Programação

Segundo Baranauskas (1998), as linguagens de programação podem ser classificadas segundo a ótica dos "meios" diferentes, onde os problemas são representados e resolvidos e esta abordagem leva a uma reflexão sobre metodologias de uso de linguagens de programação no contexto educacional, sendo essas entendidas como paradigmas de programação.

Paradigma pode ser entendido como um conceito, modelo ou padrão a ser seguido, os paradigmas de programação diferenciam as linguagens de acordo com suas funcionalidades e objetivos de cada uma.

Existem quatro tipos básicos de paradigmas de programação:

- Paradigma Imperativo - baseado em ações sequenciais que manipulam as variáveis.
- Paradigma Lógico - composto por axiomas ou fatos e regras de inferência que estabelecem

relações entre os fatos.

- Paradigma Funcional - usa expressões e funções matemáticas diminuindo a necessidade de comandos e variáveis.
- Paradigma Orientado a Objetos - trabalha com classes e objetos que trocam informações entre si.

2.6.1 Paradigma Imperativo

Essa metodologia também pode ser denominada procedural, por incluir sub-rotinas ou procedimentos como formas de estruturação dos programas produzidos. O paradigma imperativo pode ser definido como a execução de tarefas de forma sequencial, onde o fluxo de controle da execução pela máquina é ditado por sequenciação e por comandos de repetição.

Segundo Watt (1990), esse tipo de programação foi o primeiro a surgir, sendo inspirado na arquitetura de Von Newmann, sendo essa arquitetura uma das primeiras arquiteturas proposta para o desenvolvimento de computadores. Nos dias atuais o paradigma imperativo permanece dominante, sendo as linguagens C, COBOL, PASCAL e BASIC as mais conhecidas a utilizar essa metodologia.

2.6.2 Paradigma Lógico

O paradigma lógico utiliza uma lista de sentenças para formar conclusões, podendo essas serem declarativas quanto imperativas, sendo assim um programa não contém instruções explícitas à máquina. Em vez disso ele estabelece “fatos” e “regras” sobre um problema, como um conjunto de axiomas lógicos.

Esse tipo de programação é representado como um modelo abstrato de computação. A linguagem Prolog *(Programming Logic)* surgiu nos inícios dos anos 70, dos esforços de Robert Kowalski, Maarten van Emden e Alain Colmerauer e é uma linguagem desenvolvida em máquinas sequenciais, que mais se aproxima do modelo de programação presente no paradigma lógico (WATT, 1990).

2.6.3 Paradigma Funcional

O paradigma funcional surgiu com o desenvolvimento da linguagem Lisp *(List Processing),* por John McCarthy em 1958 . Essa linguagem foi projetada em uma época em que só existia processamento numérico para atender os interesses dos grupos que trabalhavam em Inteligência Artificial (BARANAUSKAS, 1998).

Esse modelo de programação trata a computação como um conjunto de funções matemáticas, sendo

que essas funções podem ou não terem parâmetros e um simples valor de retorno, onde os parâmetros são os valores de entrada das funções e o valor de retorno é o resultado de uma função.

Nesse paradigma o computador atua apenas como uma máquina que avalia as funções e o programa define e compõe as funções. Escrever um "programa" de acordo com esse paradigma consiste em definir funções e aplicar as funções para o problema em questão e o processo de "execução" pela máquina consiste em avaliar as aplicações das funções envolvidas.

2.6.4 Paradigma Orientado a Objetos

A orientação a objetos (OO) surgiu em meados de 1967 quando Johan Dahl e Kristen Nygaard criaram a linguagem Simula 1967, em 1980 Alan Kay criador do termo orientação a objetos, aperfeiçoou os conceitos de OO e criou a linguagem Smalltalk. Apesar de ter mais de 40 anos de existência, a orientação a objetos ganhou maior aceitação nos últimos anos.

O objetivo da orientação a objetos é facilitar a programação tornando os problemas computacionais mais próximos do mundo real, levando em consideração que o mundo é povoado por objetos que enviam e recebem mensagens e são capazes de reagir a essas mensagens.
Na programação orientada a objetos (POO), as unidades de programa são objetos e as mensagens possibilitam a comunicação entre esses objetos e é através delas que que uma operação de um objeto é requisitada. Em POO um programa bem sucedido tem ao final todas a mensagens respondidas e o problema inicial solucionado.
Os principais conceitos utilizados em POO são:

- **Classes** - Podem ser definidas como uma coleção de objetos, uma classe contém a descrição dos métodos e atributos que seus objetos possuirão;
- **Objetos -** Um objeto é uma instância de uma classe e engloba operações (métodos) e um estado (determinado pelos atributos);
- **Mensagens -** São requisições feitas ao objeto para que uma ação seja executada, essas solicitações são feitas entre objetos, o objeto receptor executa a ação e devolve uma resposta ao objeto requisitante;
- **Métodos -** São similares as funções, são ações que a classe pode executar. Um método consiste na descrição das operações que um objeto executa quando recebe uma mensagem;
- **Atributos -** Existem dois tipos de atributos: os de objeto e os de classe. O atributo de objeto representa o estado do objeto e cada objeto de uma classe tem um valor de atributo. O atributo de classe é o mesmo para todos os objetos dessa classe;
- **Herança –** A herança permite que um objeto herde características (métodos e atributos) de

outros objetos, devido a essa propriedade é possível realizar uma hierarquia entre objetos.

- **Polimorfismo -** Permite que um ou mais objetos produzam respostas diferentes (de acordo com seus métodos) para uma mesma mensagem.

Entre as linguagens de programação orientada a objetos bem sucedidas podemos destacar as linguagens C++, Java, Object Pascal (Delphi), C#, entre outras.

2.7 Linguagens de programação

Uma linguagem de programação em termos simples pode ser definida como um conjunto de instruções que seguem regras lógicas afim de compor programas de computador para resolução de problemas específicos (TIOBE, 2001).

Segundo Jamsa e Klander (1999) as instruções que o computador executa são séries de dígitos binários (1s e 0s), que representam sinais eletrônicos que ocorrem dentro do computador. Antigamente os programas eram escritos em linguagem binária, mas com a evolução da tecnologia o tamanho dos códigos de softwares aumentou de tal maneira que ficou inviável programar em binário. Daí surgiram as linguagens de programação.

As linguagens de programação são usualmente diferenciadas por níveis de comunicação com o computador, os níveis mais baixos são os que mais se aproximam da linguagem natural do computador (binária) e as mais altas são as mais próximas da linguagem natural para os programadores .

Atualmente existem inúmeras linguagens de programação e a cada dia surgem mais, porém poucas tornam-se referência em termos de uso e aplicação. Conforme a ***Tabela 2.1*** (TIOBE, 2011), as mais populares atualmente são: C, C++ e Java.

Tabela 2.1: Ranking das linguagens mais utilizadas. Fonte: Tiobe (2011)

Posição Out 2012	Posição Out 2011	Linguagem de programação	Pontuações Out 2012	Diferença Out 2011	Situação
1	2	C	19822,00%	+2.11%	A
2	1	Java	17193,00%	-0.72%	A
3	6	Objective-C	9477,00%	+3.23%	A
4	3	C++	9260,00%	+0.19%	A
5	5	C#	6530,00%	-0.19%	A
6	4	PHP	5669,00%	-1.15%	A
7	7	(Visual) Basic	5120,00%	+0.57%	A
8	8	Python	3895,00%	-0.05%	A
9	9	Perl	2126,00%	-0.31%	A
10	11	Ruby	1802,00%	+0.28%	A
11	10	JavaScript	1261,00%	-0.93%	A
12	12	Delphi/Object Pascal	1097,00%	-0.01%	A
13	13	Lisp	947,00%	-0.08%	A
14	18	Pascal	839,00%	+0.12%	A
15	16	Lua	728,00%	-0.07%	A
16	20	Ada	654,00%	+0.04%	B
17	15	PL/SQL	630,00%	-0.27%	B
18	25	Visual Basic .NET	599,00%	+0.12%	A--
19	21	MATLAB	591,00%	+0.02%	B
20	19	Assembly	568,00%	-0.05%	B

2.7.1 C/C++

A linguagem C foi criada em 1972 por Dennis M. Ritchie com intuito de escrever sistemas operacionais, mas devido a sua eficácia, acabou se tornando uma linguagem de uso geral (ROCHA, 2006). C é uma linguagem procedural, usualmente chamada de médio nível porque mescla elementos de linguagem de alto nível com funcionalidades de linguagem Assembly, que se trata de uma linguagem de baixo nível. O C trabalha com um compilador cujo a função é transformar um código-fonte em código executável, ou seja, linguagem de alto nível com linguagem de máquina (baixo nível).

Dentre as muitas características da linguagem C, podemos destacar:

- Portabilidade;
- Modularidade;
- Recursos de baixo nível;
- Geração de código eficiente;
- Simplicidade;
- Facilidade de uso;

- Pode ser usada para os mais diversos fins;
- Indicada para escrever desde sistemas operacionais, banco de dados, compiladores até editores de textos, entre outros.

O C++ é uma extensão do C acrescido de novos recursos como orientação a objetos. A linguagem C++ foi desenvolvida inicialmente por Bjarne Stroustrup na AT&T, de 1979 a 1983, à partir da linguagem C, tendo como ideia principal a de agregar o conceito de classes e orientação à objetos, àquela linguagem. Razão pela qual inicialmente chamava-se de “C com classes”.

Principais características do C++:

- Orientação à objetos;
- Portabilidade;
- Brevidade;
- Programação modular;
- Compatibilidade com C;
- Velocidade.

2.7.2 Java

Criada pela Sun MicroSystems, Java é uma linguagem de alto nível, orientada a objetos e multiplataforma. Seus códigos (.java) podem ser executados em qualquer sistema operacional, que tenha o interpretador do Java instalado, sem precisar ser recompilado. A linguagem Java é semelhante ao C++, mas simplificada para eliminar características que causam erros comuns em programação.

As principais características da linguagem Java são:

- Orientação a objetos;
- Portabilidade - os códigos Java podem ser executados em qualquer sistema operacional (que tenha sua máquina virtual instalada);
- Recursos de Rede - Possui diversas bibliotecas de rotina que facilitam a cooperação entre protocolos TCP/IP como o HTTP e FTP;
- Bytecode interpretado ao invés de compilado;

Outras vantagens também podem ser destacadas:

- Possui facilidades para criação de programas distribuídos e multitarefa;
- Desalocação de memória por processo do coletor de lixo;
- Carga Dinâmica de Código - Programas em Java são formados por uma coleção de classes armazenadas independentemente e que podem ser carregadas no momento de utilização.

2.7.3 Qt

Qt é uma classe de bibliotecas e *framework* multiplataforma para desenvolvimento em C++ criado pela empresa norueguesa Trolltech, mas atualmente pertence a Nokia, apesar de contar com a comunidade *open source* para o desenvolvimento e evolução das bibliotecas. Inicialmente o Qt foi distribuído com versões comerciais e com a licença GPL (GNU General Public License), que garante que o programa pode ser utilizado sem nenhuma restrição, inclusive podendo haver a alteração do mesmo.

Outra característica importante do Qt é o fato de ser multiplataforma, que permite a criação de software para praticamente todos os sistemas operacionais existentes no mercado, para isso na hora de compilar basta informar ao compilador para qual sistema operacional será utilizado. Por se tratar também de um framework o Qt possui algumas ferramentes de programação que podem auxiliar do desenvolvimento de um software, dessas ferramentas pode-se citar o Qt Designer, que se trata de uma ferramenta que auxilia no desenvolvimento de interfaces.

2.8 Lógica Fuzzy

A Lógica Fuzzy, foi criada em 1965 por Lofti A. Zadeh, professor do Departamento de Engenharia Elétrica da Univesidade da Califórnia em Berkeley, essa lógica pode ser entendida como uma generalização da lógica clássica, onde são possíveis infinitos valores lógicos existentes entre a verdade e a falsidade.

Segundo Souza (2004), a diferença entre a lógica Fuzzy e a lógica clássica, é que a Fuzzy permite mais do que apenas dois resultados, diferente da logica clássica, que permite apenas o verdadeiro e o falso. O controlador fuzzy é construído a partir de uma base de dados e uma base de regras, que utilizam variáveis linguísticas (SO; TSE; LEE, 1996). Os resultados da lógica Fuzzy, que compõem a base de dados, podem ser expressos linguisticamente, como: “grande”, “muito grande”, “pequeno” e “muito pequeno”. Tais valores estariam contidos em um conjunto Fuzzy, onde cada conjunto A, é definido em termo de relevância a um conjunto universal U, por uma função denominada função de pertinência, associando a cada elemento x um número $\mu A(x)$, no intervalo fechado [0,1] que caracteriza o grau de pertinência de x em A. Sendo assim o fator de pertinência

poderia assumir qualquer valor entre 0 e 1, representando completa exclusão ou completa inclusão, respectivamente. Essa função de pertinência possui a forma $\mu A:X \rightarrow [0,1]$.

A base de dados provê a informação necessária, geralmente correspondente ao conhecimento de um especialista, para a construção da base de regras, as quais são utilizadas pelo controlador. A base de regras, contém um conjunto de regras do tipo "se - então" baseadas em variáveis linguísticas, essas regras constituem um meio pelo qual o controlador manipula as variáveis linguísticas de modo a atingir um determinado objetivo (TANG; MAN; CHEN; KWONG, 2001).

Inicialmente o sistema de controle sofre um processo chamado de "Fuzzificação", onde é feito um mapeamento do domínio de números reais para o domínio de números Fuzzy. Em seguida, efetua-se a inferência sobre a base de regras obtendo os valores dos termos das variáveis de saída. O mecanismo de inferência define a maneira de como as regras são combinadas, provendo a base de regras (MENDEL, 1995). Dos diversos procedimentos inferenciais existentes em Fuzzy, os mais utilizados são Mamdani (WOODARD, 1996) e Takagi-Sugeno-Kang (MENDEL, 2001). Por fim as variáveis de saída sofrem um processo denominado "Defuzzificação", que consiste em converter os dados Fuzzy gerados pelo controle em valores numéricos precisos. Para isso na inferência Mamdani, são utilizadas várias técnicas como valor máximo, média dos máximos, média local dos máximos, centro de gravidade, ponto central da área e o centro da média (DRIANKOV; HELLENDOORN; REINFRANK, 1993) (DUTTA, 1993).

Uma regra Se (antecedente) Então (consequente) é definida pelo produto cartesiano Fuzzy dos conjuntos Fuzzy que a compõem. A inferência de Mamdani agrega as regras usando o operador lógico OU modelado pelo operador máximo e, em cada regra, o operador lógico E é modelado pelo operador mínimo.

O controlador fuzzy pode ser descrito de acordo com a Figura 2.8. Assim (MATTAVELLI; ROSSETTO; SPIAZZI; TENTI, 1995):

- Fuzzificação: converte as variáveis de entrada em variáveis linguísticas;
- Inferência: avaliação das entradas utilizando a base de regras, determinam saídas em forma de varáveis linguísticas;
- Defuzzificação: converte as variáveis linguísticas de saída em varáveis físicas.

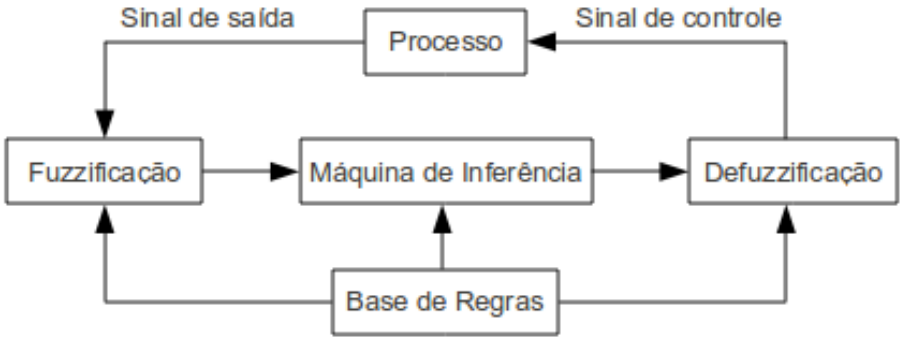


Figura 2.8: *Sistema de inferência fuzzy.*

A lógica Fuzzy vem tornando possível aproximar a máquina do raciocínio humano, propondo soluções mais realistas a problemas que apenas o cérebro humano era capaz de interpretar e resolver (SANTOS, 2003).

2.9 Fuzzy-PID

Uma tendência nos dias atuais é a utilização de controladores híbridos *fuzzy*-PID, que pode ser também classificado como *fuzzy*-PD, suprindo dessa maneira os pontos fracos que cada um dos controladores possui e utilizando-se o que cada controlador realiza de melhor no ato de se controlar a planta (AL-ODIENAT; AL-LAWAMA, 2008). Os controladores *fuzzy*-PID, na prática, possuem a habilidade de auto ajustar-se e adaptar-se melhor às não-linearidades e variações dos parâmetros do processo controlado.

Os primeiros passos para a elaboração de um controlador *fuzzy*-PID, são: a definição do universo de referência (domínio) das variáveis do controlador, a normalização em um determinado intervalo, a definição das funções de pertinência e a definição das regras de controle.

Quando se define a normalização a ser utilizada, definem-se fatores de escala para as variáveis associadas. Já, as funções de pertinência são definidas a partir da escolha dos valores linguísticos necessários ao controle e as regras de controle que associam as variáveis de entrada com as de saída. São elaboradas em função do conhecimento que os operadores e ou engenheiros possam ter do processo a ser controlado.

Na literatura constam três categorias para a classificação dos controladores *fuzzy*-PID: Ação direta, escalonamento de ganho e controle híbrido *fuzzy*-PID. A categoria de ação direta também pode ser classificada em três subcategorias, de acordo com o número de entradas: uma, duas e três entradas de ação direta no controlador *fuzzy*-PID (AKBIYIK *et al*, 2005) (ERENOGLU *et al*, 2006). Neste trabalho foi utilizada a técnica da ação direta. A classificação dos controladores *fuzzy*-PID pode ser vista na Figura 2.9.

Figura 2.9: *Classificação dos controladores Fuzzy-PID. Fonte: Adaptado de (ERENOGLU et al, 2006).*

A estrutura genérica do controlador *fuzzy*-PID, de ação direta, possui duas entradas que geralmente são o erro (e), que é o desvio entre a variável de processo e o seu valor desejado e a derivada no tempo desse sinal de erro que alimentam uma base de regras *fuzzy*, uma variável de saída (u), que é a variável manipulada.

A utilização da lógica Fuzzy, em conjunto com controladores PID, incorpora certo nível de inteligência artificial aos controladores convencionais, tornando-os mais eficazes.

2.10 Considerações finais

Neste capítulo foi possível realizar um estudo dos motores de indução trifásicos, dentre os diversos tipos de motores existentes esse trabalho utiliza o motor de indução trifásico de rotor em gaiola visto que é o mais utilizado industrialmente e possui vantagens significativas para esse trabalho se comparado com outros modelos, como por exemplo a baixa manutenção exigida por esse modelo. Houve também um estudo dos tipos de inversores existentes para se entender o funcionamento do inversor de tensão comercial que será utilizado no trabalho, além de algumas estratégias de controle utilizadas no controle de motores, como o controle escalar e o controle vetorial.

Houve o estudo de sistemas supervisórios atuais, para ajudar no desenvolvimento do software proposto, visando que o mesmo fosse de fácil utilização pelo usuário, além do estudo dos paradigmas e linguagens existentes atualmente, para a escolha do método de programação que poderia ser utilizado, sendo que o paradigma orientado a objetos é o ideal para esse trabalho, com

isso houve a escolha da linguagem C/C++, utilizando as bibliotecas e o framework Qt para o desenvolvimento do software.

Outro estudo realizado nesse capítulo, foi a lógica *fuzzy* e os controladores híbridos *fuzzy*-PID, esse estudo ajudou no desenvolvimento do projeto devido ao fato, que entendendo esses controladores é possível realizar um ajuste mais bem elaborado dos ganhos do PID que será utilizado para controlar a velocidade do motor de indução trifásico.

No próximo capítulo são realizadas as simulações, tanto com a estratégia de controle escalar, como com o controle vetorial, isso irá ajudar a entender o controlador proposto e analisar os resultados obtidos.

3. Simulação e análise

3.1 Introdução

Este capítulo apresenta a simulação de dois sistemas híbridos, propostos nesse trabalho, onde utiliza-se um sistema Fuzzy para ajustar os ganhos de um controlador PID, uma das simulações utiliza o controle escalar para acionamento de um motor de indução trifásico e a outra é realizada com controle vetorial.

A principal finalidade de realizar as simulações é verificar se o sistema híbrido proposto será eficaz no controle de velocidade do motor trifásico, utilizando um controlador PID como referência para comparar o desempenho dos dois controles e saber se no controle vetorial ou escalar o resultado obtido é satisfatório.

As simulações foram desenvolvidas através do software de modelagem Simulink/Matlab® com dados e parâmetros fornecidos pelo fabricante e/ou obtidos a partir do protótipo. O software Simulink é uma ferramenta para modelamento, simulação e análise de sistemas dinâmicos. Sua interface primária é uma ferramenta de diagramação gráfica por blocos e bibliotecas customizáveis de blocos. Na simulação recorreu-se do uso de algumas das bibliotecas específicas de blocos como SimDriveline e SimPowerSystems, que dispõem da representação de modelos físicos, não havendo assim a necessidade de criar um modelo para cada componente do sistema, como por exemplo um modelo de máquina de indução trifásica, tal como apresentado no Capítulo 2. Para a simulação do controle vetorial foi utilizado um exemplo pertencente ao Simulink denominado "power_acdrive.mdl".

3.2 Simulação com controle escalar

Esta seção apresenta a simulação do sistema híbrido, proposto nesse trabalho, onde utilizando a técnica de controle escalar, esse sistema híbrido utiliza no *fuzzy,* duas variáveis linguísticas como entradas, o erro de velocidade (E) e a derivada do erro (DE), e três saídas: saída do proporcional (SP), saída do integral (SI) e saída do derivativo (SD), cada uma delas correspondendo aos ganhos do PID. Na fase de fuzzificação, essas variáveis foram codificadas dentro de um universo de discurso comum com valores normalizados entre [-1, 1].

Todas as saídas são classificadas nos conjuntos fuzzy, definidas por: NG (Negativo Grande), NM (Negativo Médio), NP (Negativo Pequeno), ZE (Zero), PP (Positivo Pequeno), PM (Positivo Médio), PG (Positivo Grande). O conjunto de regras da saída do proporcional (SP) foram definidas de acordo com a Tabela 3.1, da saída do integral (SI) foram definidas de acordo com a Tabela 3.2 e

saída do derivativo de acordo com a Tabela 3.3. Os bancos de regras de cada um dos ganhos foram ajustados através do método de tentativa e erro, de modo a se obter o desempenho do controlador híbrido proposto.

A estrutura típica de cada regra é a seguinte: Saída do proporcional – Regra 1: SE o erro de velocidade é ZE (Zero) e a derivada do erro é NG (Negativo Grande) ENTÃO a SP (saída do proporcional) é NG (Negativo Grande).

Tabela 3.1: *Base de regras do proporcional no escalar*

		Erro de Velocidade						
		NG	**NM**	**NP**	**ZE**	**PP**	**PM**	**PG**
Derivada do Erro	**NG**				NG			
	NM				NM			
	NP							
	ZE	NG	NM		ZE		PP	PG
	PP							
	PM							
	PG							

Tabela 3.2: *Base de regras do integral no escalar*

		Erro de Velocidade						
		NG	**NM**	**NP**	**ZE**	**PP**	**PM**	**PG**
Derivada do Erro	**NG**				NG			
	NM				NM			
	NP				ZE			
	ZE	NG	NM		ZE	PP	PM	PG
	PP							
	PM							
	PG							

Tabela 3.3: *Base de regras do derivativo no escalar*

		Erro de Velocidade						
		NG	**NM**	**NP**	**ZE**	**PP**	**PM**	**PG**
Derivada do Erro	**NG**	PG	PG		NP			
	NM	PG	PP		PG			
	NP				PG			
	ZE	NM	NM	NP	PG	PG	PM	PG
	PP				PG			
	PM				PG			PP
	PG				PP	PM	PP	PP

Para cada saída do controle, foram escolhidas sete funções de pertinência triangulares distribuídas ao longo do universo de discurso das variáveis de erro da velocidade, derivada do erro e as saídas de cada ganho do controlador PID, este número de funções é ideal em sistemas Fuzzy, conforme mostrado na Figura 3.1.

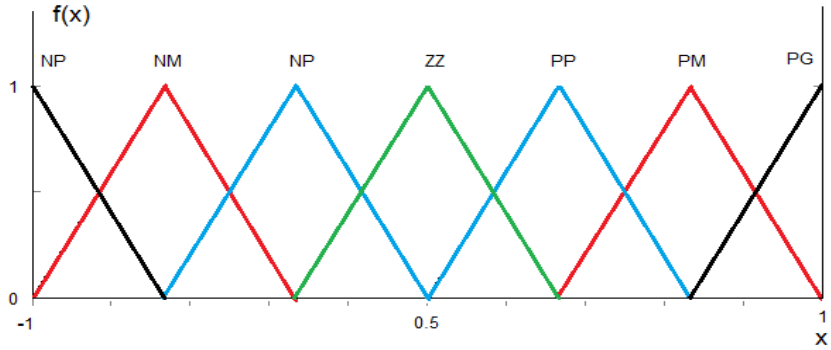


Figura 3.1: *Funções de pertinência das saídas.*

Foi utilizado o método de inferência de Mandani para a determinação do grau de pertinência das variáveis de entrada através dos operadores de mínimo e máximo. O grau de pertinência das saídas é determinado utilizando o centro de área das figuras determinadas pelas entradas.

A Figura 3.2 mostra a estrutura do controlador fuzzy proposto, onde a velocidade de escorregamento é dada pela integração da variação de velocidade de escorregamento (VS) determinada pela saída do PID, sendo os ganhos ajustados utilizando saída do proporcional (SP), saída da integral (SI) e saída da derivada (SD) do fuzzy, que recebe como entrada o erro (E) e a derivada do erro (DE) que foram calculadas anteriormente.

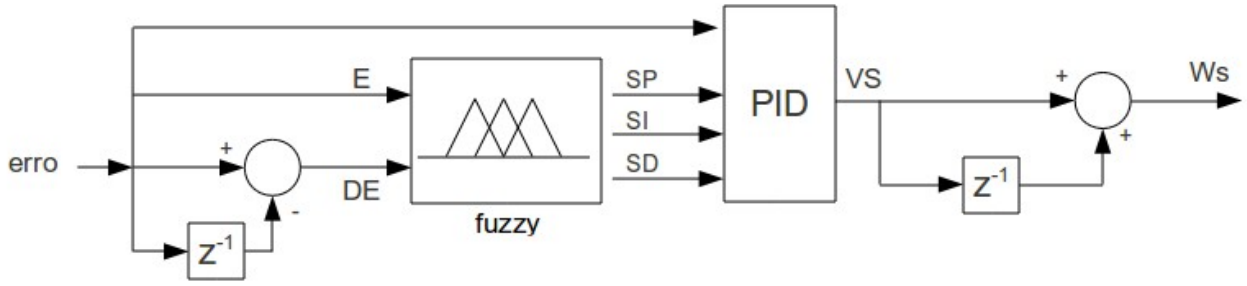


Figura 3.2: *Controlador fuzzy proposto.*

A simulação utilizada nesse sistema híbrido, utilizando o controle escalar, pode ser vista na Figura 3.3. Os dados do motor utilizado são definidos na Tabela 3.4.

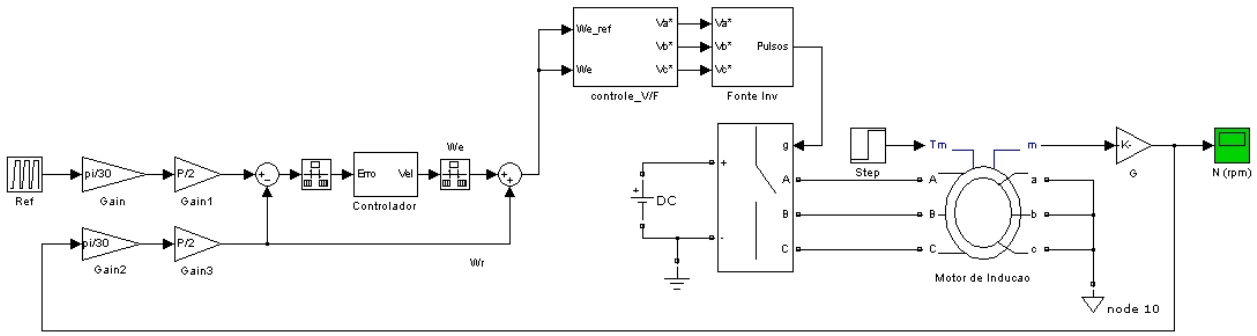


Figura 3.3: *Simulação do controle escalar*

Tabela 3.4: *Dados do motor de indução no controle escalar*

Parâmetro	Valor
Potência nominal (P)	$14960\,W$
Tensão nominal (Vn)	$220\,V$
Frequência de sincronismo (fe)	$60\,Hz$
Número de pares de polos (p)	4
Velocidade de sincronismo (ωe)	$377\,rad/s$
Resistência do estator (Rs)	$0,01062\,\Omega$
Reatância do estator (Xs)	$0,2145\,\Omega$
Indutância do estator (Ls)	$5,689\,e^{-4}\,H$
Resistência do rotor (Rr)	$0,0764\,\Omega$
Reatância do rotor (Xr)	$0,2145\,\Omega$
Indutância do rotor (Lr)	$5,689\,e^{-4}\,H$
Reatância de magnetização (Xm)	$5,834\,\Omega$
Indutância mútua (Lm)	$0,0154\,H$
Momento de inércia (J)	$0,089\,Kg.m^2$

Primeiramente utilizou-se o controlador PID convencional, houve um ajuste de ganho do proporcional em 15, o do integral em 0 e o do derivativo em 0,1, como referência de velocidade foi gerado um padrão composto por uma sequência de rampa e degrau, assim como um degrau no torque de carga. A velocidade obtida com o controlador PID pode ser vista na Figura 3.4.

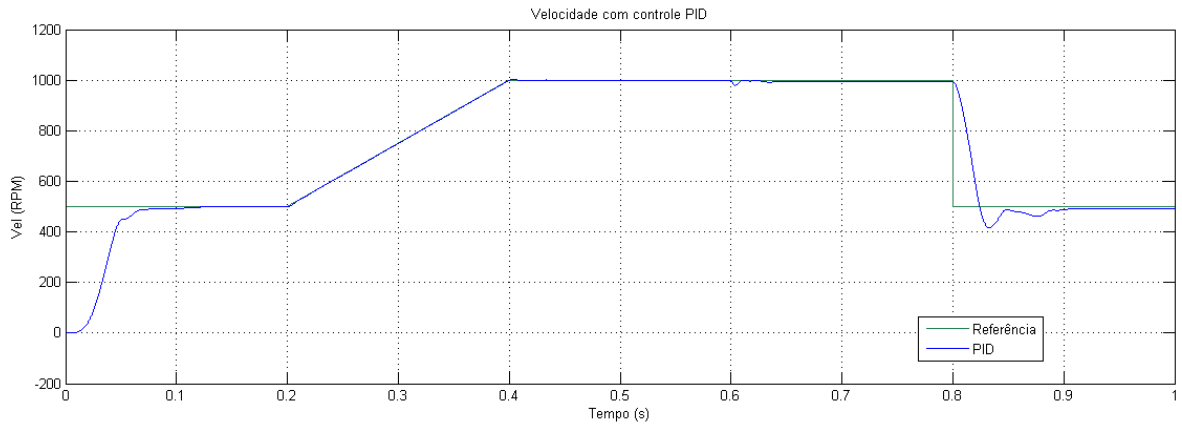


Figura 3.4: *Velocidade com controlador PID na simulação do controle escalar.*

Para verificar o desempenho do controlador proposto foi utilizado o sistema de controle que pode ser visto na Figura 3.5, onde é possível observar que as saídas do controlador fuzzy estão ligadas diretamente aos ganhos do PID.

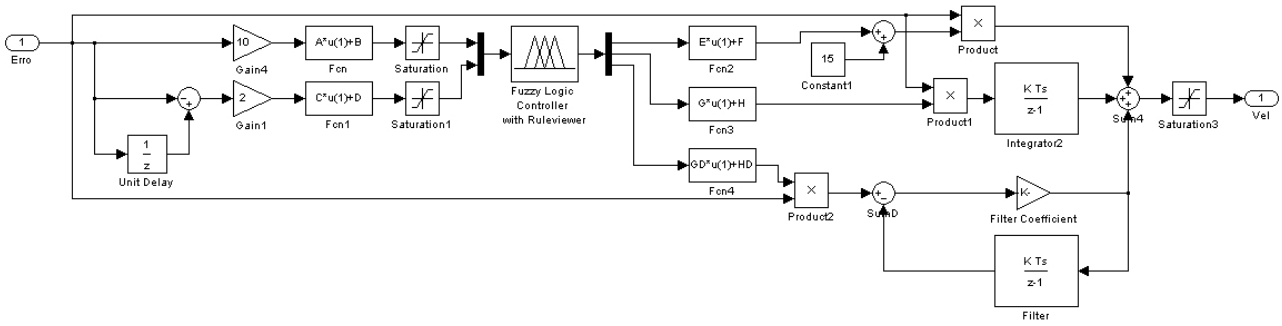


Figura 3.5: *Controlador proposto na simulação com o escalar.*

Nesse sistema as entradas e saídas do controlador fuzzy foram normalizadas conforme o código na Figura 3.6, onde é mostrada a normalização nas entradas e nas saídas respectivamente.

```
%Entradas
A = 2/(1000-(-1000);
B = -1*((-1000+1000)/(1000-(-1000)
C = 2/(0.1-(-0.1))
D = -1*((-0.1+0.1)/(0.1-(-0.1)
```

```
%Saidas
E = (10-(-10))/2;
F = (10+(-10))/2;
G = (1-(-1))/2;
H = (1+(-1))/2;
GD = (0.1-(-0.1))/2;
HD = (0.1+(-0.1))/2;
```

Figura 3.6: *Normalizações nas entradas e nas saídas do controle escalar.*

Utilizando a mesma referência de velocidade utilizada no controle PID, utilizando o controlador proposto obteve-se o gráfico apresentado na Figura 3.7. Pode-se observar que a velocidade manteve-se muito próximo da referência.

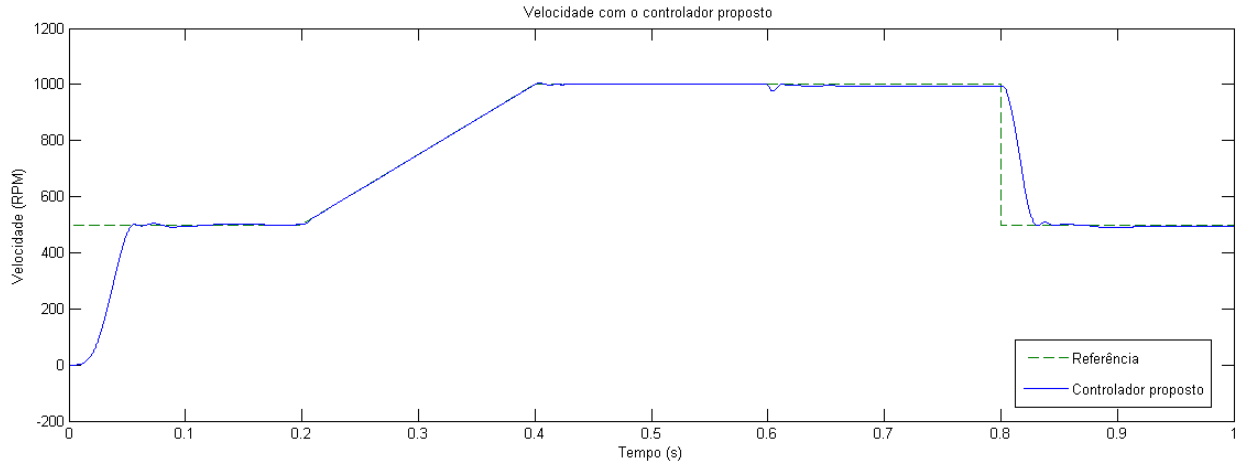


Figura 3.7: *Velocidade com o controlador proposto na simulação escalar.*

Pode-se observar que o controlador proposto obteve uma melhor resposta quando comparada com o PID convencional, essa comparação pode ser verificada na Figura 3.8, onde vemos o controlador com PID comum e o controlador proposto.

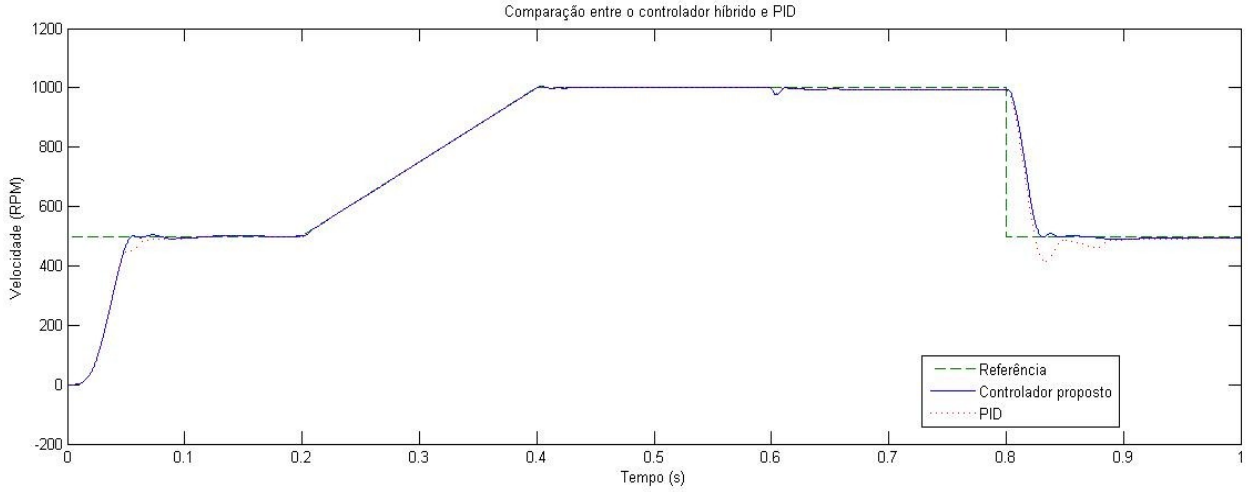


Figura 3.8: *Comparação entre o controlador proposto e o PID no escalar.*

A Figura 3.9 mostra uma comparação do torque no PID e o torque do controlador proposto. Nota-se que no instante em que a carga aumenta, o torque do motor no controlador proposto e no controlador PID, permanecem praticamente iguais, sendo que no controlador proposto a variação é levemente inferior, se comparada com o controlador PID, o mesmo acontece no tempo 0,8s onde a referência de velocidade é reduzida pela metade.

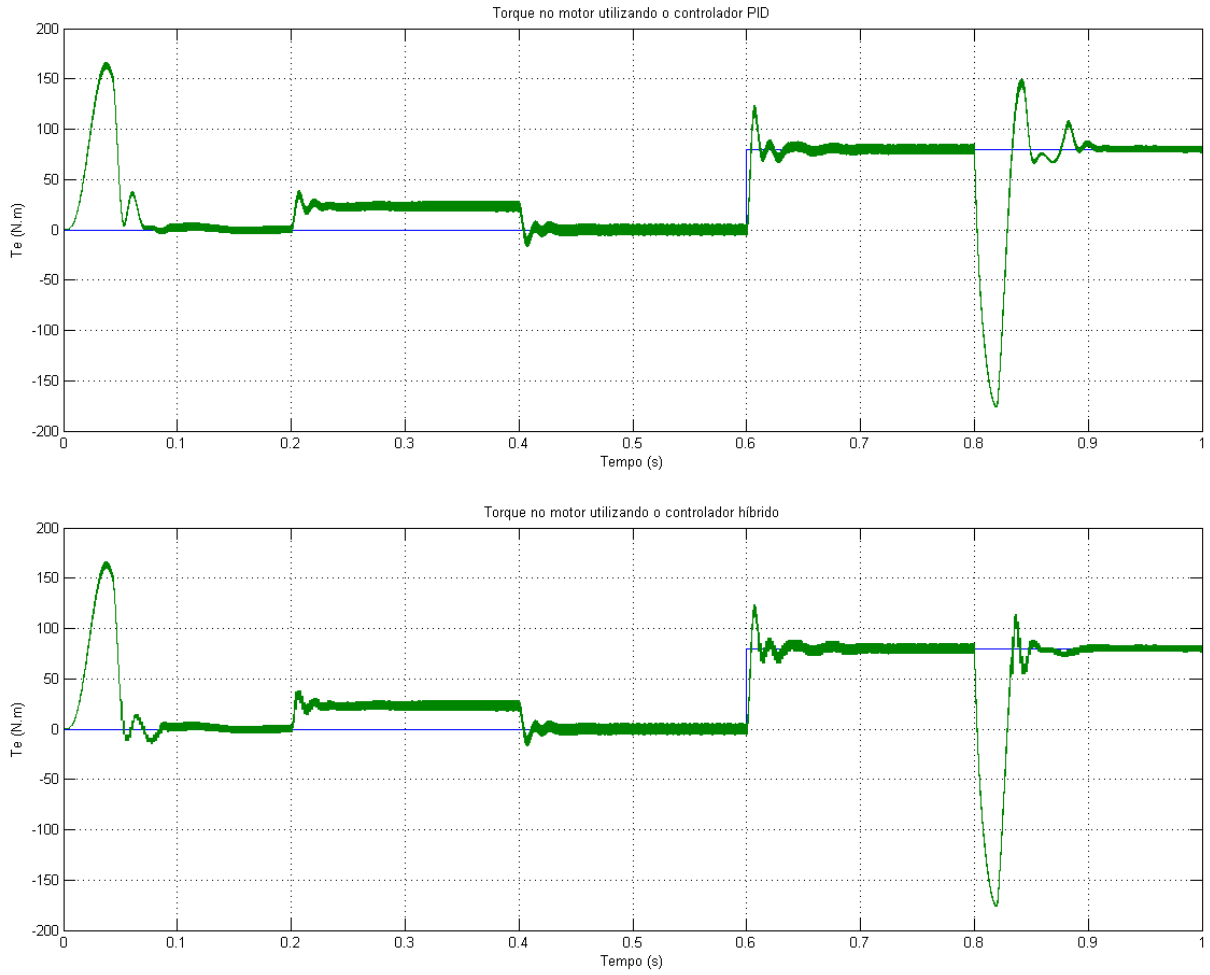


Figura 3.9: *Comparação de torque entre os dois controladores escalares.*

3.3 Simulação com controle vetorial

Assim como o controlador híbrido proposto na simulação anterior, utilizando o controle escalar, a simulação do controlador híbrido utilizando o controle vetorial também possui as mesmas duas variáveis linguísticas de entrada e as três variáveis linguísticas de saída. Assim como na simulação utilizando o controle escalar, as variáveis foram codificadas dentro de um universo de discurso comum com valores normalizados entre [-1, 1].

As bases de regras foram ajustadas para utilização do controle vetorial, conforme ***Tabela 3.5***, ***Tabela 3.6*** e ***Tabela 3.7***, respectivamente representando o conjunto de regras da saída do proporcional (SP), saída do integral (SI) e saída do derivativo.

A estrutura de cada regra segue a mesma lógica utilizada na simulação anterior.

Tabela 3.5: *Base de regras do proporcional no vetorial*

		Erro de Velocidade						
		NG	**NM**	**NP**	**ZE**	**PP**	**PM**	**PG**
Derivada do Erro	**NG**				PG			
	NM				PG			
	NP				PM			
	ZE	NG	NM	NP	PG	PP	PM	PG
	PP				PP			
	PM				PM			
	PG				PG			

Tabela 3.6: *Base de regras do integral no vetorial*

		Erro de Velocidade						
		NG	**NM**	**NP**	**ZE**	**PP**	**PM**	**PG**
Derivada do Erro	**NG**							
	NM							
	NP				PP			
	ZE	NG	NM	NP	ZE	PP	PG	PG
	PP							
	PM				PP			
	PG				PP			

Tabela 3.7: *Base de regras do derivativo no vetorial*

		Erro de Velocidade						
		NG	**NM**	**NP**	**ZE**	**PP**	**PM**	**PG**
Derivada do Erro	**NG**							
	NM							
	NP				ZE			
	ZE	NG	NM	NP	PP	PP	PP	PG
	PP							
	PM							
	PG							

Para cada saída do controle, existem sete funções de pertinência triangulares distribuídas ao longo do universo de discurso, assim como na simulação do sistema híbrido anterior.

O método de inferência de Mandani foi usado para a determinação do grau de pertinência das variáveis de entrada através dos operadores de mínimo e máximo. O grau de pertinência das saídas é determinado utilizando o centro de área das figuras determinadas pelas entradas.

Para simulação foi realizada com base no exemplo “power_acdrive.mdl” presente no software SIMULINK, que também foi utilizado para realizar a simulação. Na Figura 3.10 é possível visualizar o exemplo utilizado nessa simulação. Na Figura 3.11 é mostrado o controlador vetorial.

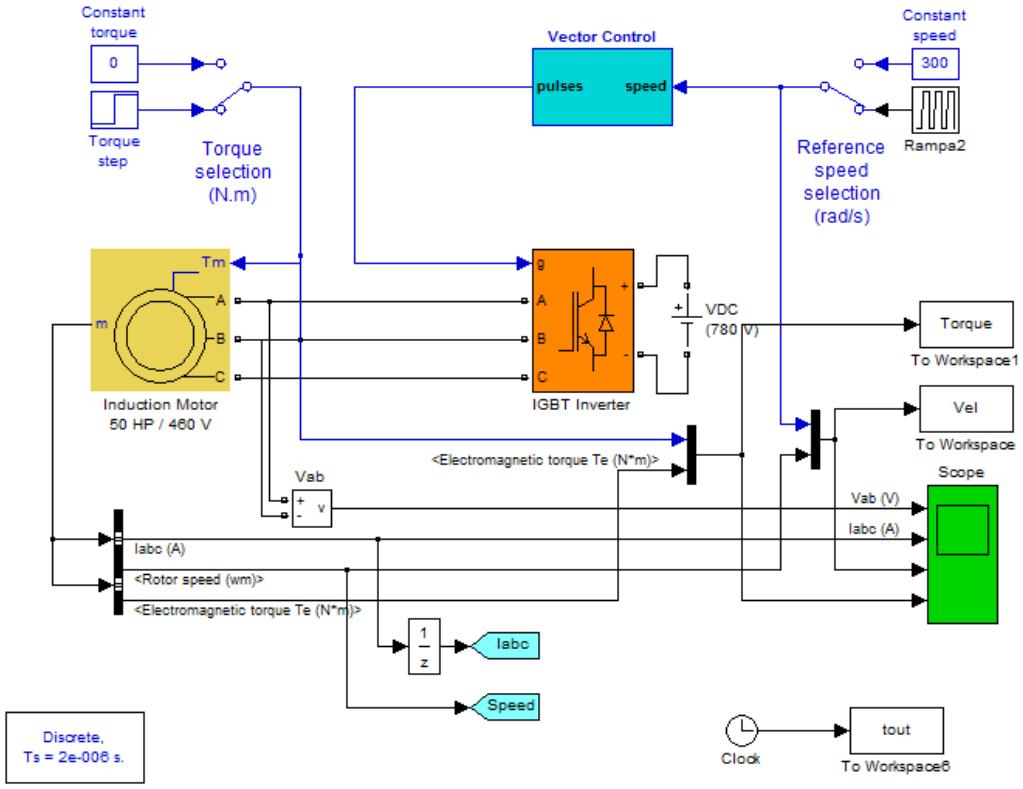


Figura 3.10: *Exemplo utilizado na simulação do controle vetorial.*

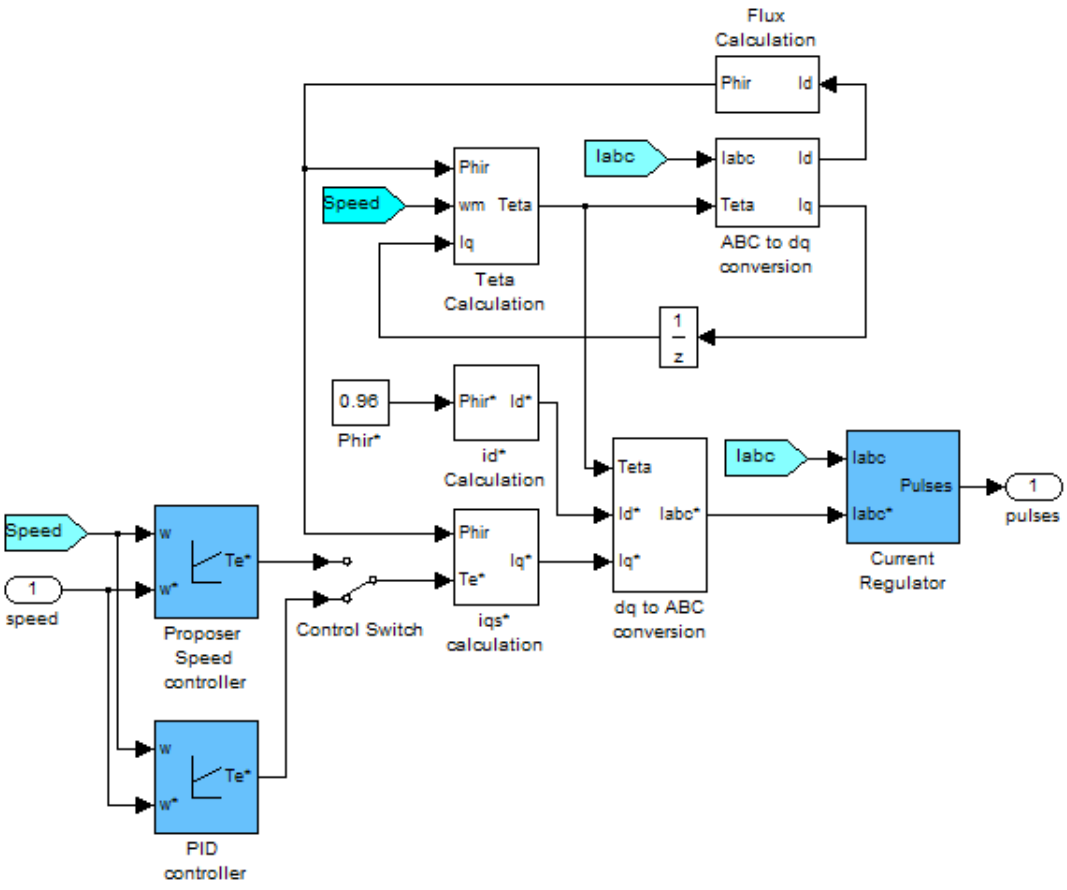


Figura 3.11: *Controlador vetorial.*

Utilizando o PID, sem o controlador proposto, houve um ajuste de ganho do proporcional em 13, o do integral em 0,3 e o do derivativo em 0,05. Como referência de velocidade foi gerado um padrão composto por uma sequência de rampa e degrau, assim como um degrau no torque de carga. A velocidade obtida com o controlador PID pode ser vista na Figura 3.12, onde a velocidade foi medida em rad/s.

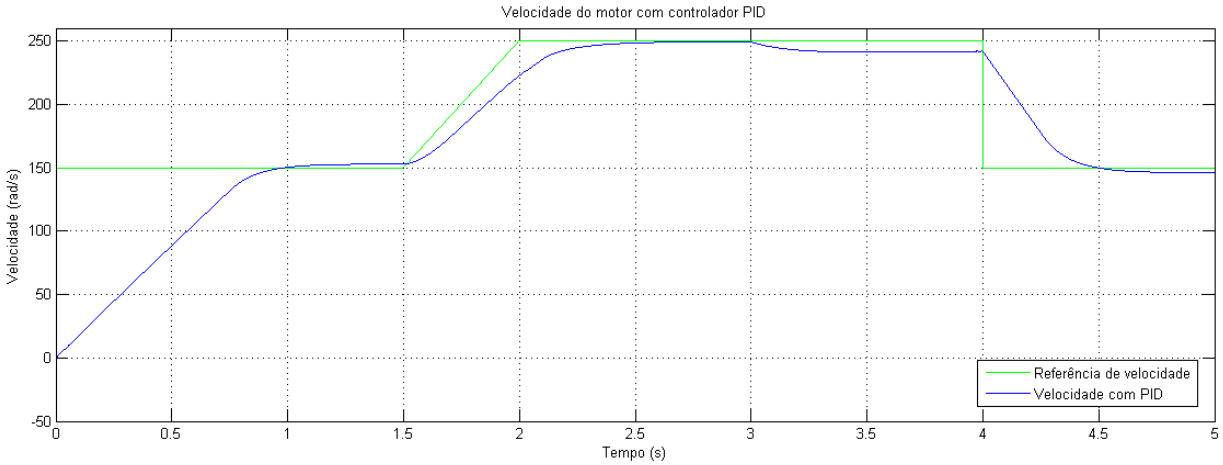


Figura 3.12: *Velocidade com controlador PID com controle vetorial.*

O controlador proposto utilizando o controle vetorial pode ser visto na Figura 3.13, onde é possível observar que as saídas do controlador fuzzy estão ligadas diretamente aos ganhos do PID.

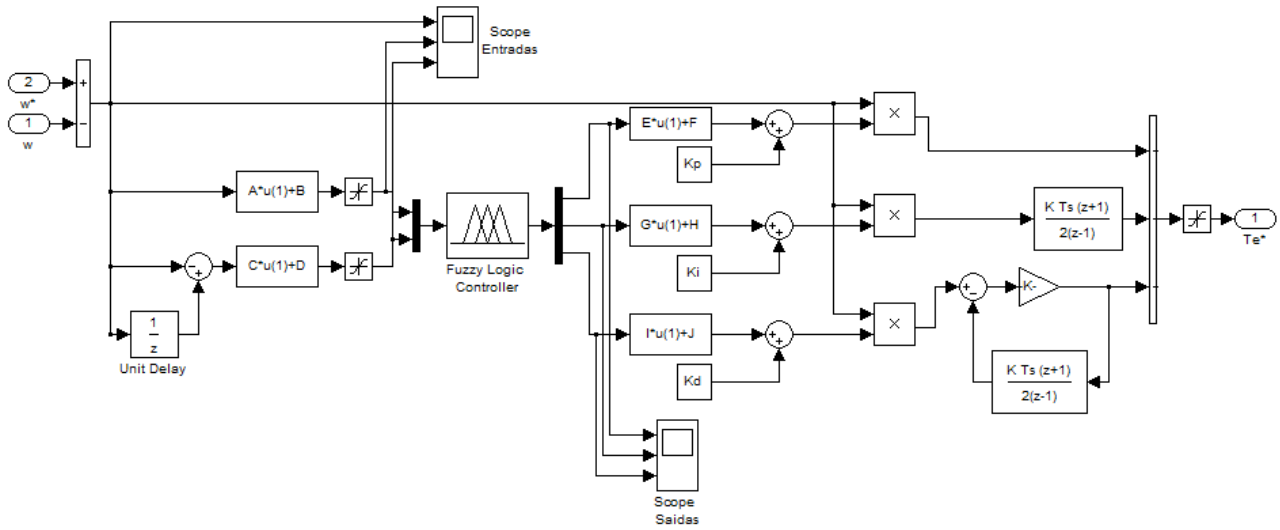


Figura 3.13: *Controlador proposto utilizando controle vetorial*

Nesse sistema as entradas e saídas do controlador fuzzy foram normalizadas conforme a Figura 3.14 onde é mostrada a normalização nas entradas e das saídas respectivamente.

```
%Entradas
Emax =  300;  Emin = -300;
A = 2/(Emax-Emin); %0,0034
B = -1*((Emin+Emax)/(Emax-Emin)); %0

CEmax =  0.0004; CEmin = -0.0004;
C = 2/(CEmax-CEmin);
D = -1*((CEmin+CEmax)/(CEmax-CEmin));
```

```
%Saidas
Kpmax =  40; Kpmin = -40;
E = (Kpmax-Kpmin)/2;
F = (Kpmax+Kpmin)/2;

Kimax =  3; Kimin = -3;
G = (Kimax-Kimin)/2;
H = (Kimax+Kimin)/2;

Kdmax =  1; Kdmin = -1;
I = (Kdmax-Kdmin)/2;
J = (Kdmax+Kdmin)/2;
```

Figura 3.14: *Normalizações nas entradas e saídas utilizando controle vetorial*

Utilizando a mesma referência de velocidade utilizada no controle PID, utilizando o controlador proposto obteve-se o gráfico apresentado na Figura 3.15. Pode-se observar que a velocidade manteve-se muito próximo da referência.

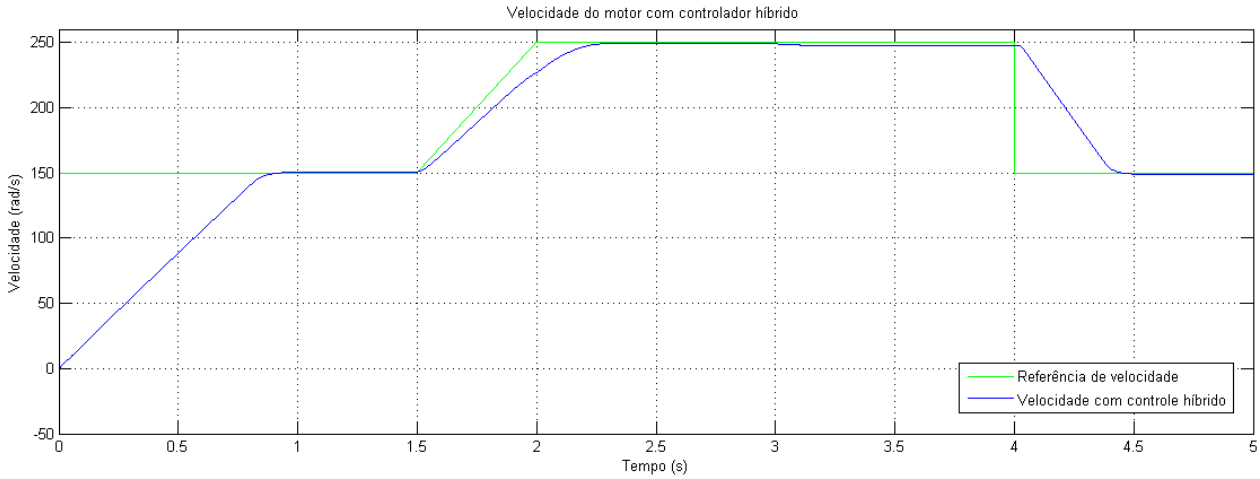


Figura 3.15: *Velocidade com o controlador híbrido utilizando controle vetorial.*

Nesse gráfico é difícil observar qualquer diferença entre o controlador híbrido e o controle PID, mas na comparação vista na Figura 3.16, é possível notar que em alguns pontos o controlador híbrido conseguiu uma resposta melhor, principalmente no ponto em que há um aumento do torque.

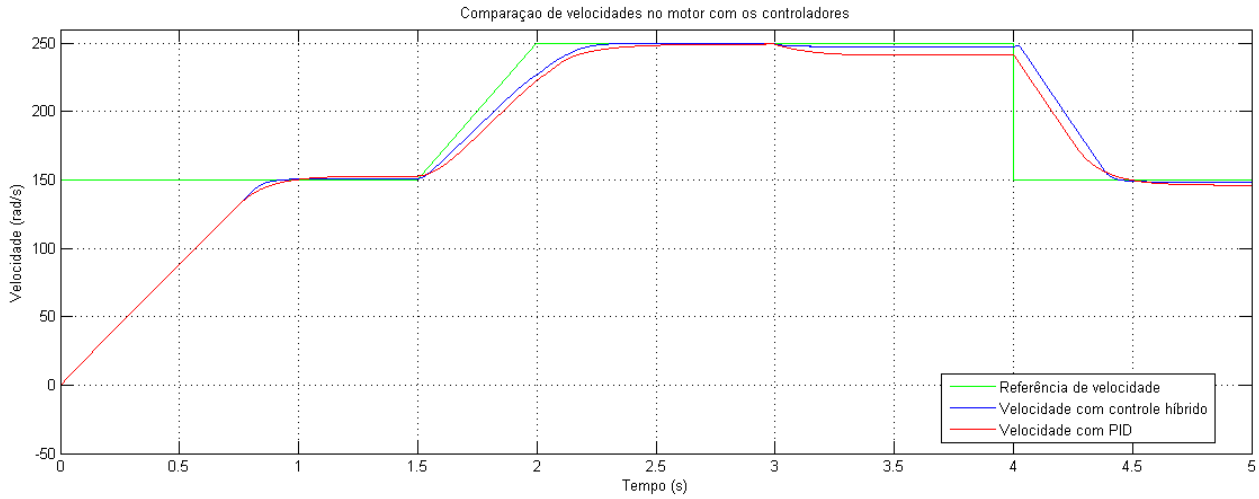


Figura 3.16: *Comparação entre o controlador hibrido e o PID no controle vetorial.*

A Figura 3.17 mostra uma comparação do torque no PID e o torque do controlador híbrido utilizando a técnica de controle vetorial. Nota-se que ambos os torque permanecem praticamente iguais, com leves variações entre alguns pontos.

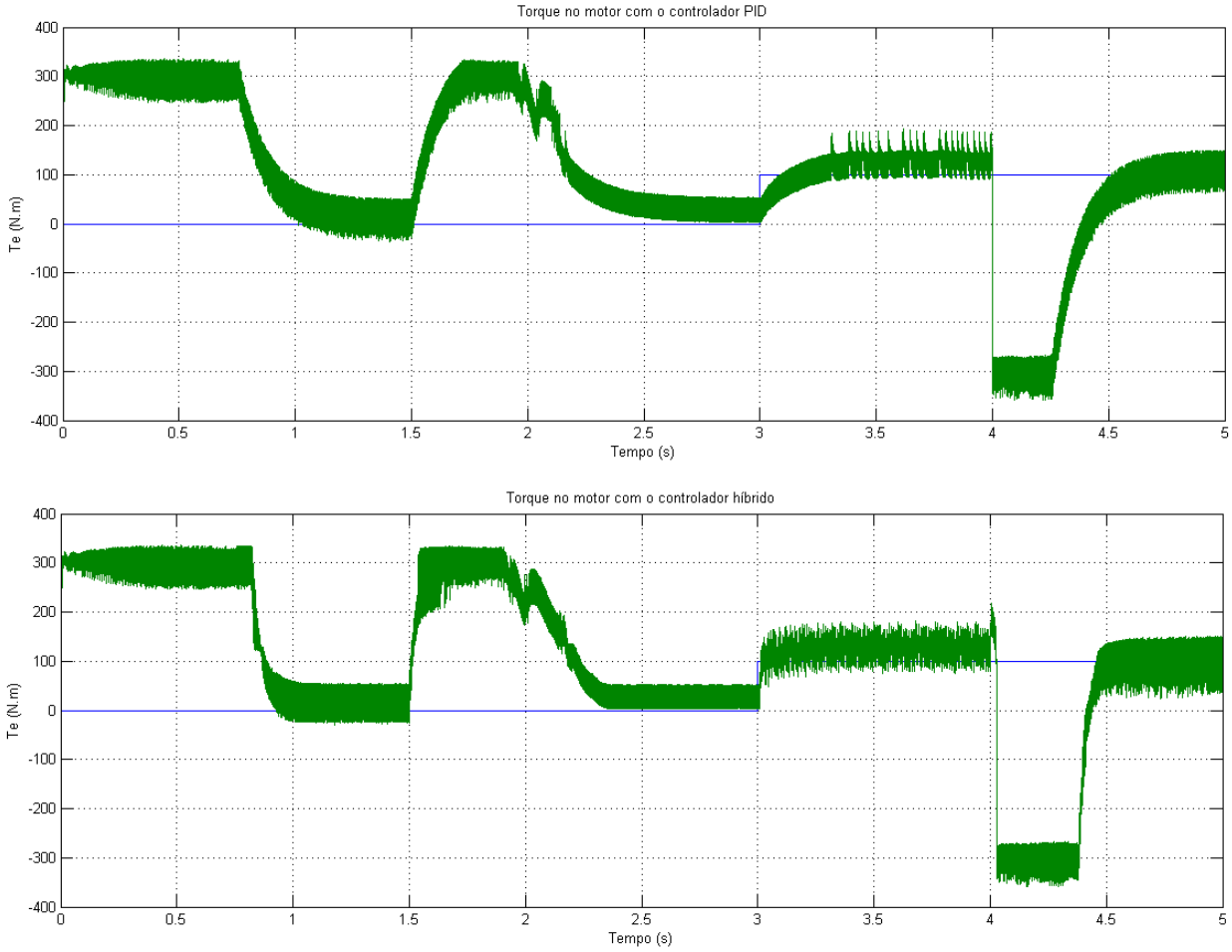


Figura 3.17: *Comparação de torque entre os dois controladores no controle vetorial.*

3.4 Considerações finais

Nesse capítulo houve a realização de duas simulações, para verificar a eficiência do sistema híbrido *fuzzy*-PID. Foi possível verificar que no controle escalar, o controlador híbrido proposto possui uma melhor resposta, do que quando se está utilizando a técnica de controle vetorial, mas com ambos os controladores foi possível realizar uma melhora de desempenho quando se comparado apenas com o controlador PID, principalmente nos pontos de transição.

No próximo capítulo é mostrado o ambiente e materiais utilizados para o desenvolvimento do software proposto nesse trabalho, é apresentado também o código-fonte do aplicativo e a análise dos resultados obtidos, utilizando o mesmo para ajustar o PID interno do inversor de tensão comercial utilizado nesse projeto.

4. O Aplicativo Computacional

4.1 Fluxograma do software

Antes de se realizar a implementação do software há sempre a necessidade de desenvolver um fluxograma das etapas que deverão existir no software para auxiliar o desenvolvimento da aplicação. A Figura 4.1 apresenta o fluxograma do software que será implementado nesse trabalho.

De forma resumida pode-se descrever as etapas do fluxograma como se segue:

a) Início – nesse fase o software irá inicializar o ambiente de execução, como a configuração e preparação da porta serial, nessa etapa será também configurada a velocidade de referência que o motor deverá operar e a preparação do sistema fuzzy;

b) Lê velocidade – logo que iniciar e o motor começar a girar o software fará uma leitura da velocidade atual do motor, de maneira periódica, utilizando a porta serial com o protocolo RS-232;

c) Armazena Leitura – nessa etapa o software deverá armazenar a velocidade obtida para ser utilizada junto com todas as outras velocidades obtidas anteriormente. Esse vetor será utilizado para gerar o gráfico de velocidade e assim analisar a resposta obtida com o sistema;

d) Mudança – nessa fase haverá a verificação de mudança de velocidade ou referência de velocidade. Caso não haja nenhuma mudança o sistema repete os procedimentos voltando ao passo de leitura do motor, do contrário o sistema irá ajustar os novos ganhos do PID através do controlador Fuzzy;

e) Fuzzy – o controlador fuzzy entra em operação gerando os novos valores de ganhos do PID que serão enviados logo em seguida;

f) Modifica inversor – o sistema irá enviar os novos valores de ganhos do PID para o inversor que irá regular o controlador PID interno do inversor.

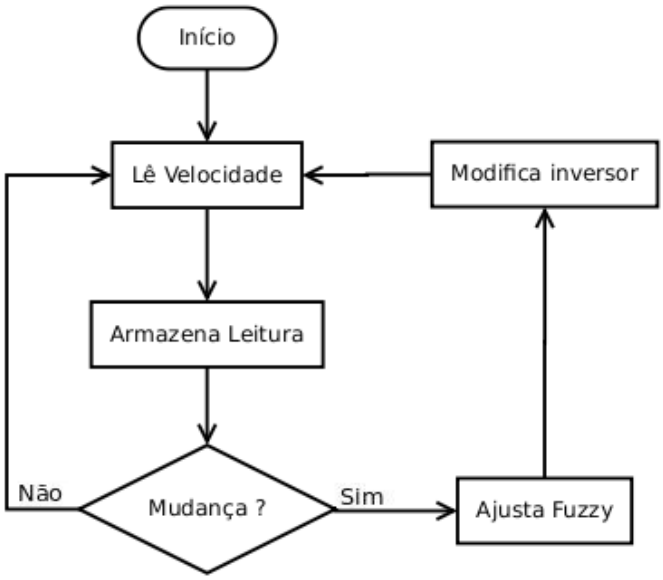


Figura 4.1: *Fluxograma do software proposto.*

Conforme pode-se observar no fluxograma apresentado, o software está dividido em algumas fases consideradas primordiais, como a comunicação do software com o inversor e o controlador Fuzzy. Cada uma dessas fases serão tratadas em um tópico dentro deste capítulo.

Para desenvolvimento do software foi criado duas classes, afim de separar o sistema *fuzzy* do resto do software, para que essa classe possa ser utilizada em futuros trabalhos. O diagrama dessas classes pode ser visualizado na Figura 4.2.

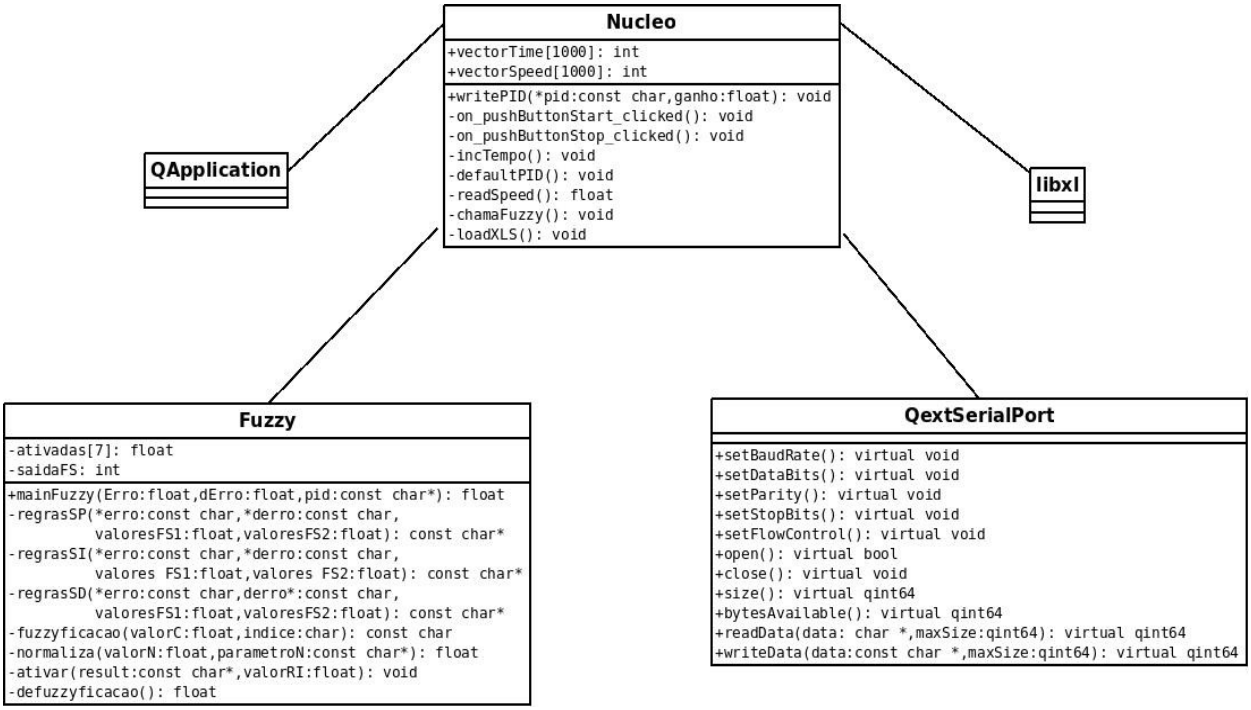


Figura 4.2: *Diagrama de classes do software.*

Analisando o diagrama, têm-se duas classes principais, a classe “Nucleo” e a classe “Fuzzy”, que podem ser visualizadas no Anexo I e Anexo II respectivamente, nesses anexos são apresentadas as classes e suas funções, as principais classes serão detalhadas posteriormente.

4.2 Materiais e métodos

Os materiais, utilizados nesse projeto, são um motor de indução trifásico (MIT), um inversor de tensão, uma interface de comunicação serial, ligado ao inversor, e também um computador que é utilizado para execução do software proposto. Cada um desses componentes utilizados nesse trabalho serão descritos com mais detalhes. Na Figura 4.3 abaixo é possível visualizar esse ambiente.

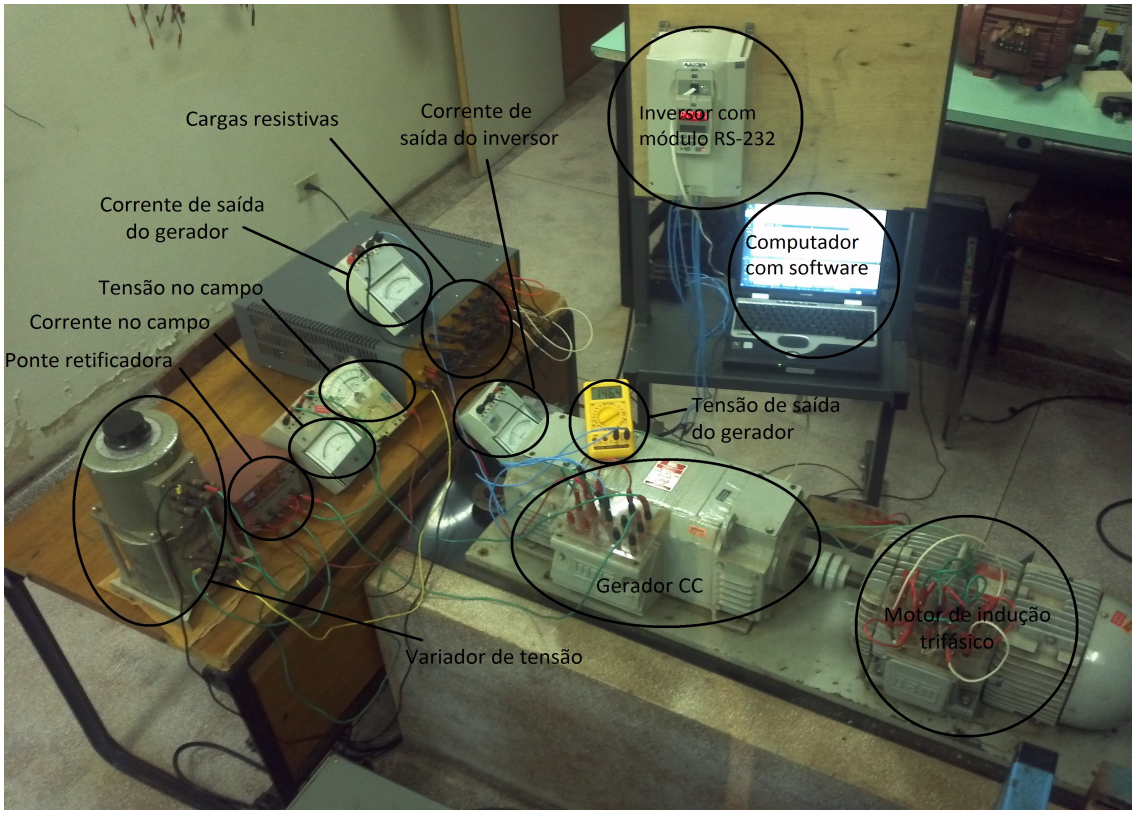


Figura 4.3: *Materiais utilizados.*

4.2.1 Motor de indução trifásico

Os parâmetros do motor de indução trifásico utilizados neste trabalho são apresentados na Tabela 4.1, cujos valores foram obtidos através do inversor de tensão que realizou a medição quando o parâmetro P408 (auto-ajuste) foi colocado 1 (um) e a partir dos dados de placa de um motor de 2 kW.

Tabela 4.1: *Parâmetros do MIT*

Fabricante	**WEG**
Potência nominal	2 kW
Tensão nominal	220(Δ)/380(Y)
Frequência	60 *Hz*
Número de pares de polos (*p*)	4
Rotação nominal	1690 *rpm*
Corrente nominal	9,6 *A*
Constante Lr/Rr	0,136 *s*
Constante Tm	0,04 s
Parâmetros elétricos	
Resistência do estator (*Rs*)	0,894 Ω
Indutância de dispersão (σ*ls*)	14.2 *mH*
Corrente de magnetização (I*mr*)	3,3 *A*

4.2.2 Inversor de tensão

O inversor de tensão comercial utilizado neste trabalho é o CFW-09. Um produto de alto desempenho que permite o controle de velocidade e torque de motores de indução trifásicos (WEG, 2010).

Entre as diversas características encontradas nesse inversor podemos destacar as seguintes:

a) Possui controle de tensão imposta V/F (escalar), VVW, Vetorial c/ encoder, Vetorial Sensorless, PWM SVM (Space Vector Modulation), reguladores de controle, fluxo de velocidade em software (full digital);

b) Possui frequência de saída de 0 a 3,4 x “Frequência nominal do motor”;

c) Possui 2 entradas analógicas não isoladas e 6 entradas digitais isoladas;

d) Possui IHM (Interface homem máquina) com teclas, display e LEDs;

e) Através de dispositivos opcionais é possível obter, uma interface de comunicação serial RS-232.

Como é possível observar, o inversor CFW-09 possui diversas características que deverão ser utilizadas, como por exemplo a interface de comunicação serial que será utilizada para obtenção de dados e para atuação sobre o inversor, outro detalhe importante é a entrada analógica, utilizada para o controle de velocidade do motor trifásico.

4.2.3 Interface de comunicação

Utilizando um módulo adicional, que é inserido no lugar da interface homem máquina (IHM) e oferecido pela própria WEG, chamado de "RS-232 SERIAL INTERFACE" é possível obter uma interface de comunicação que pode trabalhar sobre o protocolo Serial RS-232 em conexão RJ-11, presente no módulo. Caso seja necessário a utilização da IHM esse módulo provê a conexão da mesma. A Figura 4.4 a seguir mostra o módulo RS-232.

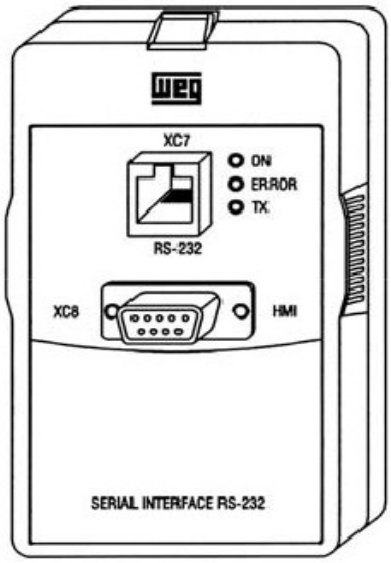


Figura 4.4: *Módulo RS-232.*

Pode-se comandar, parametrizar e supervisionar o CFW-09 através da interface serial RS-232. O protocolo de comunicação é baseado no tipo pergunta resposta conforme normas ISO 1745, ISO 646, com troca de caracteres ASCII entre os inversores e um mestre (PC, PLC, etc.). A taxa de transmissão máxima é de 9600 bps. A interface serial RS-232 é ponto a ponto, não é isolada galvanicamente de 0 volts (V) da eletrônica do inversor e permite distâncias de até 10 metros (m) (WEG, 2010).

Os inversores que possuem interface de comunicação RS-232 possuem um software de controle da transmissão/recepção de dados pela interface serial, de modo a possibilitar o recebimento de dados enviados pelo mestre e o envio de dados solicitados pelo mesmo (WEG, 2010).

O software proposto irá trabalhar sobre o inversor, exercendo a função de mestre, pois será ele o responsável por requisições de informações ou envio de dados para controle de velocidade do motor e outras funções, para que essa comunicação seja perfeita o software deverá trabalhar com o protocolo utilizado pelo inversor, assim o mestre terá condições de realizar as seguintes operações:

a) Habilitar/desabilitar geral;

b) Sentido e rotação;

c) Referência de velocidade;

d) Local/remoto;

e) Ajuste de velocidade;

O protocolo de comunicação utilizado pelo CFW-9 utiliza a norma ISO 1745, sendo assim, é usado somente caracteres de texto sem cabeçalho. Conforme WEG (2010), são utilizadas dois tipos de mensagens, chamadas de telegramas:

- **Telegrama de Leitura:** para consulta do conteúdo de variáveis do inversor;
- **Telegrama de Escrita:** para alterar o conteúdo de variáveis ou envio de comandos para o inversor.

O telegrama de leitura permite ao mestre realizar consulta de variáveis presentes no inversor, nesse tipo de mensagem sempre será esperado uma resposta contendo um valor do inversor. Na Figura 4.5 é mostrado o formato das mensagens utilizados pelo mestre e pelo inversor para um telegrama de leitura.

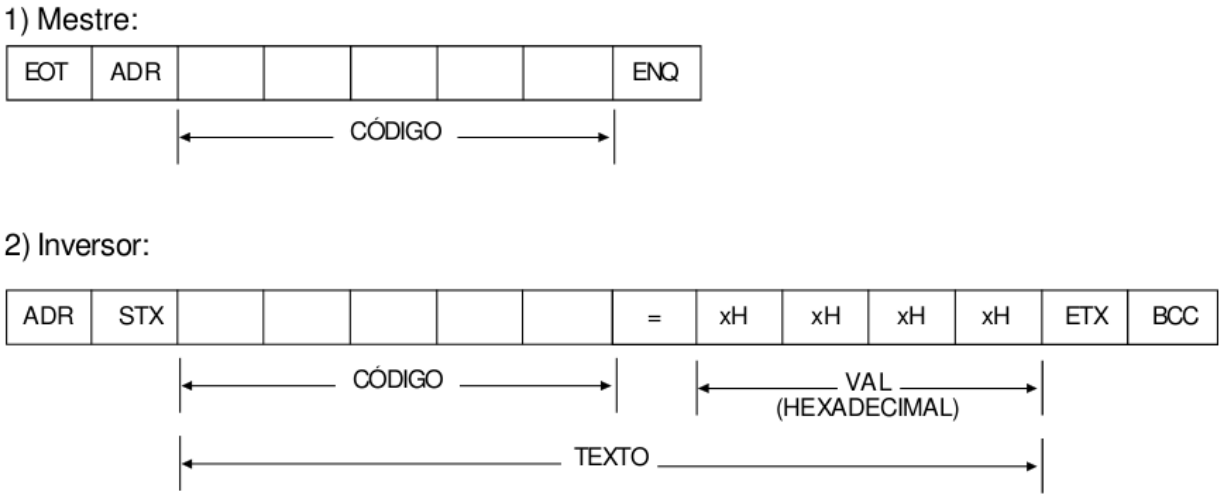


Figura 4.5: *Telegrama de Leitura.*

O telegrama de escrita é utilizado pelo mestre para enviar dados de uma variável ou mesmo um comando ao inversor, nesse tipo de mensagem o inversor deverá informar o mestre se os dados ou comandos foram aceitos. Na Figura 4.6 é possível visualizar os pacotes de transmissão entre o mestre e o inversor.

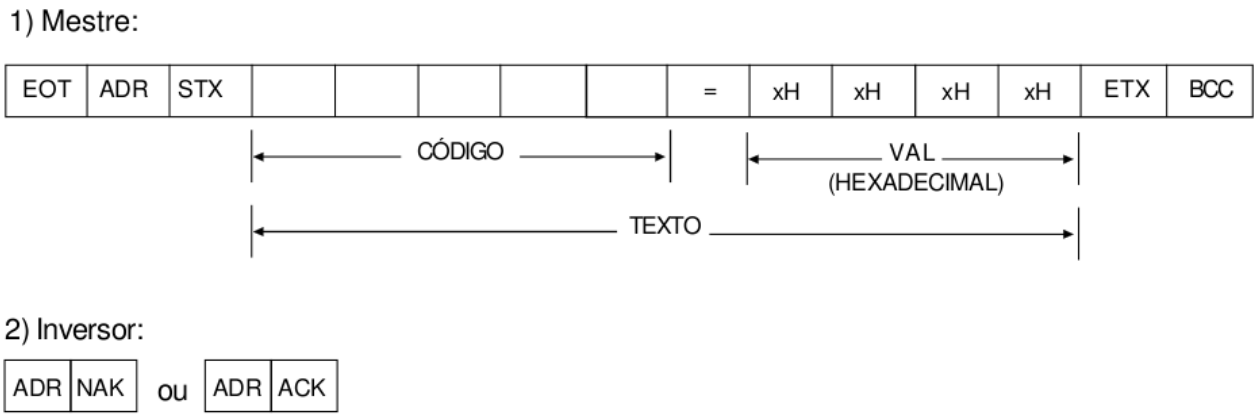


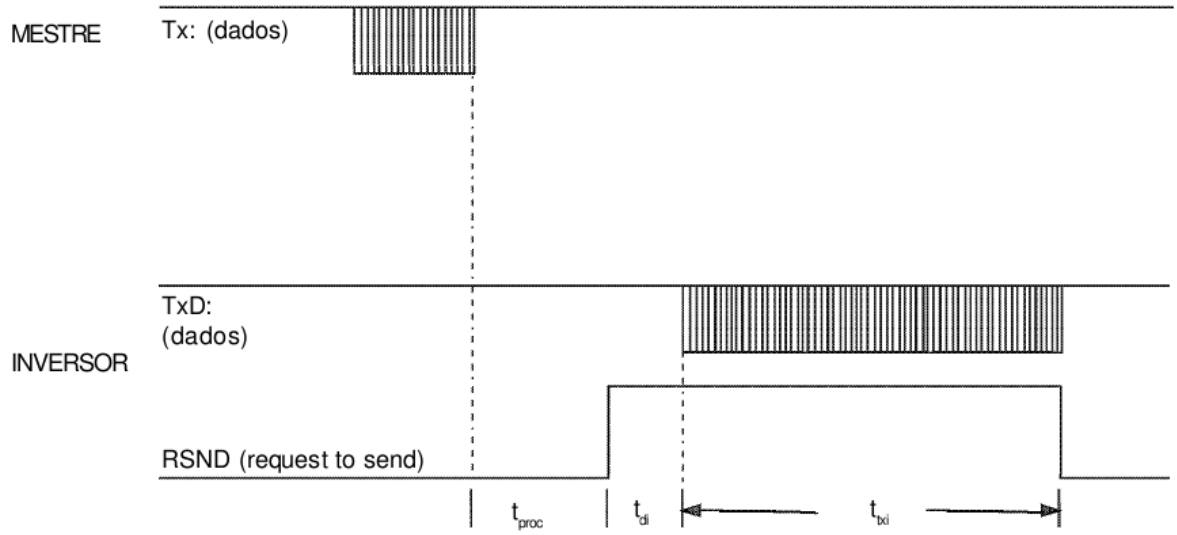
Figura 4.6: *Telegrama de Escrita.*

O protocolo do CFW-09 pode prevenir alguns erros existentes na comunicação, para monitoramento de erro é utilizado a paridade dos caracteres de 7 bits, conforme ISO 656, sendo que o CFW-09 utiliza esquema de paridade par. Caso haja erros de paridade na recepção de dados, o telegrama será ignorado, o mesmo ocorre com erros de sintaxe.

Os erros encontrados durante a comunicação não provocam bloqueio do inversor, não desativam relês de defeitos e informam na variável estado lógico (V02), alguns tipos de erros são:

- Erro de Paridade longitudinal (BCC);
- Erro de Parametrização (Incompatibilidade de parâmetros);
- Variável e parâmetros inexistentes;
- Valor desejado fora dos limites permitidos;
- Tentativa de escrita em variável só de leitura ou comando lógico desabilitado;
- Comunicação serial desativada, quando decorrido o tempo máximo programado.

O protocolo utilizado pelo inversor trabalha em tempos de transmissão e recepção pré definidos, como visto na relação de erros. O fato de ter esses tempos, pode vim a ser um motivo para o inversor não aceitar um comando ou não devolver uma resposta, caso o tempo de transmissão e recepção sejam curtos demais. Na Figura 4.7 pode-se observar os tempos utilizados na comunicação entre o mestre e o inversor.



Tempos (ms)		Típico
T_{proc}		10
T_{di}		5
T_{txi}	leitura	15
	escrita	3

Figura 4.7: *Tempos para Leitura/Escrita de Telegramas .*

4.3 Comunicação Serial

Para estabelecimento da comunicação serial e o software implementado, foram necessárias algumas considerações de como o usuário utilizará o mesmo, por se tratar de um aplicativo que pode ser utilizado no meio acadêmico e industrial, leva-se em conta a integração do mesmo com os diversos sistemas operacionais existentes no mercado, como o Linux, Unix e Windows. Para que isso seja possível será utilizado a biblioteca "QextSerialPort", pois a mesma permite a manipulação da porta serial, levando em consideração os sistemas operacionais.

Através da biblioteca "QextSerialPort" pode-se utilizar as funções presentes para criação e configuração da porta serial, como podemos ver no código fonte presente no Anexo I.

Onde, primeiramente têm-se a criação de um objeto sendo criado utilizando a classe "QextSerialPort", depois especifica-se o dispositivo da porta serial. Em "setDataBits" configura-se a quantidade de bits que será utilizada na transmissão. Para se definir o tipo de paridade utiliza-se a função "setParity", logo depois utiliza-se "setFlowControl". Para definir se haverá controle de fluxo na porta serial, o "setStopBits" é utilizado para definir a quantidade de bits de parada. Em "setBaudRate" configura-se o *Baud Rate* (a velocidade máxima da porta serial) e por último é realizada a abertura da porta serial.

Para o processo de escrita no inversor, foi utilizado a função “writeData” da classe, para isso chama-se a função, presente no objeto “CommPort”, onde é passado a variável que é escrita no inversor, e o tamanho dessa mesma variável, isso irá gerar um código de retorno que na variável “i”, que poderá ser utilizado para verificar se houve algum problema na transmissão dos dados.

O processo de leitura da porta serial é bem parecido com o processo de escrita, onde utiliza-se a função “readData”, que irá armazenar o que receber em uma variável chamada “buff” e a quantidade de bytes em “numBytes”, gerando um código de erro na variável “i”, caso houver algum erro.

4.4 Controle Fuzzy

Para o desenvolvimento do software proposto foi utilizado o framework e conjunto de bibliotecas Qt, pois se trata de um conjunto de biblioteca estável, possui também a possibilidade de criação para diversos sistemas operacionais e a facilidade de desenvolvimento de interface gráfica.

O software possui a capacidade de leitura da velocidade através da porta serial, utiliza esse valor para calcular o erro e a derivada do erro, normaliza essas entradas, faz a fuzzyficação utilizando inferência Mandani Max-Min. As saídas serão defuzzificadas utilizando o método Centro de gravidade (CoA). Todas as saídas serão normalizadas e então enviadas ao inversor modificando os parâmetros P161, P162 e 166, que constituem respectivamente ganho do proporcional, ganho da integral e ganho da derivada.

A inicialização do software é iniciada através da classe “Nucleo”, já apresentada no Anexo I, nessa etapa inicial é criada a interface gráfica mostrada na Figura 4.8, criada utilizando-se as bibliotecas Qt, o sistema inicializado ao acionar o botão “Iniciar”, nesse momento o PID é configurado com os ganhos nos seus valores padrões de fábrica, através da função “defaultPID”, demonstrada no Anexo III, logo após o motor trifásico é acionado.

Nesse momento o software entra em uma estrutura de repetição que permanece até o botão “Parar” ser ativado, essa estrutura faz com que a função “chamaFuzzy” seja realizada, essa lê a velocidade atual do motor, calcula o erro e a derivada do erro, e passa esses valores a função “mainFuzzy” da classe “Fuzzy”. No Anexo IV é demonstrada a função “chamaFuzzy” e no anexo V, a função “readSpeed”, que lê a velocidade, armazena em um vetor e altera a referência de velocidade.

Figura 4.8: *Janela do software proposto.*

Ao iniciar sistema *fuzzy*, será chamada a função "main*Fuzzy"*, esta função recebe o erro, a derivada do erro e o ganho que deve ser aplicado o fuzzy. Ela encontra-se descrita em sua íntegra no anexo VI, sendo esta, responsável por chamar todas as outras funções que integram o sistema *fuzzy* e normalizar os erros obtidos, para isso a função "*normaliza",* que encontra-se no Anexo VII, é chamada. Essa função irá transformar os erros em números crisps. Uma característica interessante desta função é receber o valor que precisa ser normalizado e o parâmetro que determina de que se trata o dado a ser normalizado, podendo ser E (erro), dE (derivada do erro), SP (saída proporcional), SI (saída integral) e SD (saída derivada), depois de normalizado as entradas serão transformadas em números crisps entre -1 e 1.

Logo após ser feita a normalização das entradas a função "*fuzzyficacao"* é chamada. Essa função encontra-se no Anexo VIII e possui a obrigação de transformar os valores *crisps* em valores *fuzzy,* essa função foi escrita de modo que apenas os coeficientes das retas são necessários para quantificar o quanto do valor calculado pertence a um certo conjunto fuzzy. Sendo a equação (3.2) da reta:

$$y = ax + b \tag{3.2}$$

Então para dois pontos, P1(X1, Y1) e P2(X2, Y2), tem-se as equações (3.3) e (3.4):

$$a = \frac{(Y2 - Y1)}{(X2 - X1)} \tag{3.3}$$

$$b = Y1 - (aX1) \tag{3.4}$$

Através dessas equações obtêm-se a Tabela 4.2.

Tabela 4.2: *Coeficientes das retas fuzzy*

	NGd				NMs				NMd			
	X1	Y1	X2	Y2	X1	Y1	X2	Y2	X1	Y1	X2	Y2
	-1,0000	1,0000	-0,6673	0,0000	-1,0000	0,0000	-0,6673	1,0000	-0,6673	1,0000	-0,3341	0,0000
a	-3,00571085061617				3,00571085061617				-3,00120048019208			
b	-2,00571085061617				3,00571085061617				-1,00270108043218			

	NPs				NPd				Zs			
	X1	Y1	X2	Y2	X1	Y1	X2	Y2	X1	Y1	X2	Y2
	-0,6673	0,0000	-0,3341	1,0000	-0,3341	1,0000	0,0000	0,0000	-0,3341	0,0000	0,0000	1,0000
a	3,00120048019208				-2,99311583358275				2,99311583358275			
b	2,00270108043218				0				1			

	Zd				PPs				PPd			
	X1	Y1	X2	Y2	X1	Y1	X2	Y2	X1	Y1	X2	Y2
	0,0000	1,0000	0,3337	0,0000	0,0000	0,0000	0,3337	1,0000	0,3337	1,0000	0,6663	0,0000
a	-2,99670362601139				2,99670362601139				-3,00661455201443			
b	1				0				2,00330727600722			

	PMs				PMd				PGs			
	X1	Y1	X2	Y2	X1	Y1	X2	Y2	X1	Y1	X2	Y2
	0,3337	0,0000	0,6663	1,0000	0,6663	1,0000	1,0000	0,0000	0,6663	0,0000	1,0000	1,0000
a	3,00661455201443				-2,99670362601139				2,99670362601139			
b	-1,00330727600722				2,99670362601139				-1,99670362601139			

Onde as regras NG (Negativo Grande), NM (Negativo Médio), NP (Negativo Pequeno), Z (Zero), PP(Positivo Pequeno), PM(Positivo Médio) e PG (Positivo Grande), possuem reta de subida (s) e reta de descida (d). No código, é possível observar que há a verificação de onde o valor crisp se encontra, e realiza-se o cálculo para quantificar as pertinências ativadas e quais serão essas.

A função *"fuzzyficacao"* é chamada duas vezes para cada entrada, uma vez para achar a primeira pertinência (P) e a outra para achar a segunda pertinência (S).

Depois de encontrado os quatro valores fuzzy, será chamada uma função que irá realizar a inferência max-min e verificar quais serão regras na saída fuzzy, para cada saída existe uma função específica com sua respectiva base de regras, no Anexo IX é possível visualizar a função "*regrasSP"* que possui a base de regras da saída do proporcional, nos Anexos X e XI é possível observar as funções das saídas "*regrasSI" e* "*regrasSD"* respectivamente. Nas três funções de saída, pode-se destacar o trecho onde é feito a inferência calculando qual o menor valor *fuzzy* retornado, através da função "min" presente nas bibliotecas.

A função *"ativar"* no Anexo XII irá verificar todas as regras que foram ativadas e adicionar o valor de cada regra dentro de um vetor de 7 posições, sendo que a posição 0 (zero) equivale a regra NG, 1 (um) regra NM e assim sucessivamente até a posição 6 (seis) que equivale a regra PG. Assim é possível passar apenas esse vetor para a função "*defuzzyficacao"*.

A função *"defuzzyficacao"* encontra-se no Anexo XIII, essa função utiliza os mesmos coeficientes de retas calculados anteriormente na função *"fuzzyficacao"*, então a função varre cada uma das

posições do vetor, tentando encontrar as regras ativadas, para isso cria-se uma variável que irá fazer a varredura seis vezes, primeiramente verifica-se se a regra "NG" foi ativada, pois essa regra e a regra "PG" devem ser calculadas as áreas de maneira diferente das outras regras, pois essas regras se encontram no limite do universo de discursos, depois de verificar as regras "PG" e "NG" foram ativadas, varre-se as outras regras, ao encontrar uma posição diferente de zero no vetor o programa irá verificar se as outras posições posteriores também são diferentes de zero. No código fonte é possível observar a varredura que é feita.

Conforme as regras ativadas são encontradas o software irá calcular a área da regra ativada e o centro dessa regra, esses dois dados serão armazenados e somados aos mesmos dados das próximas regras ativadas, com esses valores serão realizados os cálculo do centro de gravidade (CoA) da figura formada, obtendo assim a saída da classe *"Fuzzy"* e retornando esse valor a classe *"Nucleo"*.

Na classe *"Nucleo"*, a função *"normaliza"* será chamada para normalizar as saídas obtidas, com essas saídas normalizadas a função *"writePID"*, disponível no Anexo XIV, será chamada para cada um dos ganhos, com a função de escrever no inversor os parâmetros correspondente aos ganhos do PID, além disso essa função verifica se a saída do controlador fuzzy está entre os valores aceitos pelo inversor, caso não seja, o valor é ajustado.

Como esses valores recebidos pelo inversor não podem ser diferentes de inteiros a função irá separar cada um dos números e transformá-los em inteiros, inclusive separando também as casas decimais, como pode-se ver no trecho presente na Figura 4.9.

```
int inteiro = (int)floor(ganho);
float decimaltemp = ganho-((float)inteiro);
decimaltemp = decimaltemp*10;
int decimal = (int)decimaltemp;
```

Figura 4.9: *Ajuste das variáveis para escrita no inversor.*

Agora que os valores estão prontos para serem enviados, para isso é necessário criar um *buffer* de dez posições, seguindo o protocolo utilizado para escrever no inversor, coloca-se o parâmetro que deseja escrever e o valor desse parâmetro. Nesse caso foi necessário separar cada número do valor obtido e enviar, para isso houve um deslocamento dos binários para obter os valores inteiros separadamente. No código fonte é possível visualizar os *buffers* enviados para escrever em cada um dos ganhos. Um outro detalhe é o digito verificador do protocolo (BCC), isso é possível realizando uma "and" com todos os dígitos e armazenado em uma variável do tipo "char" chamada de "BCC".

O software ficará em execução até o momento em que for transmitido o sinal de parada, nesse

momento o software irá desligar o motor e parar a execução.

4.5 Resultados

O programa foi executado em três situações diferentes, em todos os experimentos foram geradas trezentas amostras, sendo que na amostra oitenta a referência passa de 875 RPM para 1750 RPM, na amostra cento e cinquenta é a carga que permanece até o final do experimento, na amostra duzentos e quarenta a referência de velocidade volta novamente para 875 RPM, em todos os momentos. Logo após as transições foram realizadas medições de velocidade com um tacômetro .

No primeiro experimento foi utilizado o controle escalar do próprio inversor. Sem a alteração dos valores do PID, não houve alteração desses valores pois o inversor utilizado permite apenas que no controle vetorial esses valores possam ser alterados, então o único objetivo aqui foi apenas poder utilizar isso como base para análise do controlador híbrido, proposto nesse trabalho. Na Figura 4.10, pode-se observar a velocidade utilizando o controle escalar do inversor e na Tabela 4.3 é possível verificar as medições realizadas com o tacômetro nesse sistema.

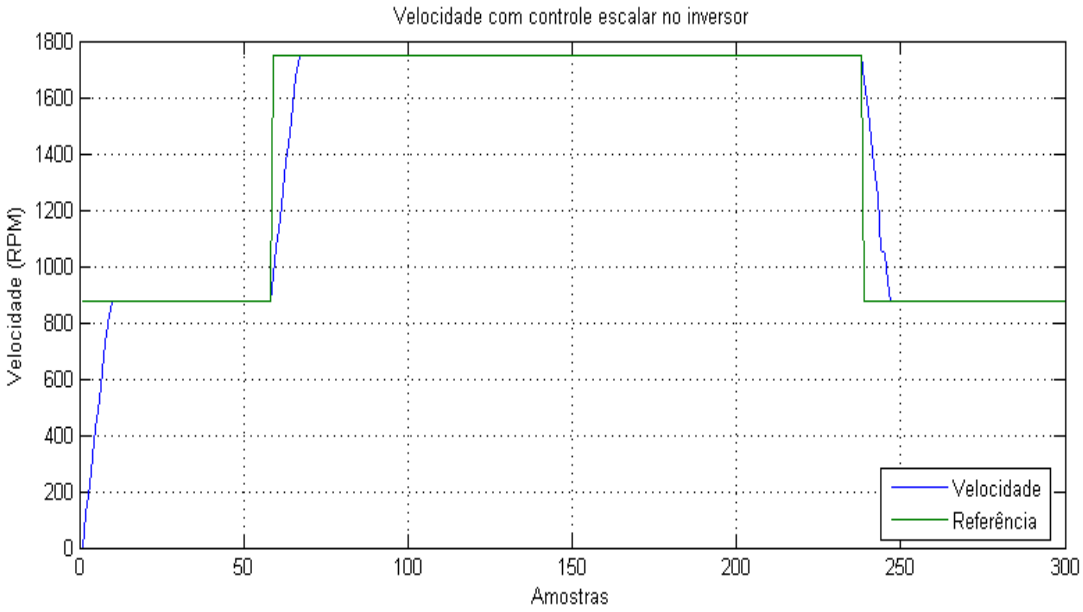


Figura 4.10: *Velocidade do controle escalar no inversor.*

Tabela 4.3: *Medições com o tacômetro no controle escalar*

Situação	**Velocidade (RPM)**
Entre as amostras 30 e 50	858
Entre as amostras 100 e 120	1719
Entre as amostras 180 e 200	1665
Entre as amostras 220 e 240	820

No segundo experimento foi realizado a execução do software utilizando o controle escalar do inversor, para utilizar esse controle foi necessário nas configurações do inversor fixar o parâmetro "P202" (Tipo de controle) igual a 3 (Vetorial Sensorless), e depois acessar o parâmetro "P408" (Auto ajuste) e mudá-lo para 1 (Sem girar), onde o inversor entrará em processo de ajustar alguns parâmetros para que seja compatível com o motor utilizado. Na Figura 4.11 é apresentado a velocidade do motor trifásico utilizando o controle vetorial sem alteração dos ganhos do PID e na Tabela 4.4 é possível visualizar as medições realizadas com o tacômetro.

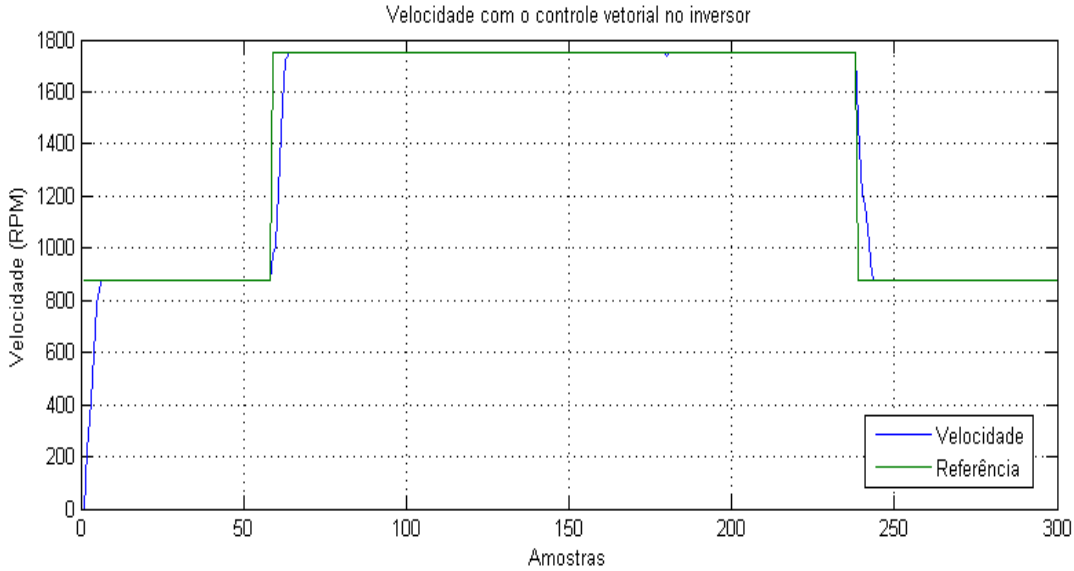


Figura 4.11: *Velocidade do controle vetorial no inversor.*

Tabela 4.4: *Medições com o tacômetro no controle vetorial*

Situação	**Velocidade (RPM)**
Entre as amostras 30 e 50	860
Entre as amostras 100 e 120	1727
Entre as amostras 180 e 200	1678
Entre as amostras 220 e 240	823

O terceiro experimento foi utilizar o controlador proposto nesse trabalho para controlar os ganhos do PID presente no inversor. Com esse controle foi possível obter o gráfico apresentado na Figura 4.12, e na Tabela 4.5 é possível verificar as medições realizadas com o tacômetro utilizando o controlador híbrido proposto.

Figura 4.12: *Velocidades do controle híbrido no inversor.*

Tabela 4.5: *Medições com o tacômetro no controle híbrido*

Situação	**Velocidade (RPM)**
Entre as amostras 30 e 50	861
Entre as amostras 100 e 120	1729
Entre as amostras 180 e 200	1681
Entre as amostras 220 e 240	828

4.6 Considerações finais

Nesse capítulo foi demonstrado os materiais utilizados nesse projeto e o funcionamento do software proposto, houve também uma análise dos resultados obtidos e apesar de a diferença entre as medições serem poucas, acredita-se que obteve-se um resultado satisfatório, devido a comprovação de que é possível realizar o controle alterando-se os ganhos do PID interno do inversor de tensão.

No próximo capítulo são realizadas as conclusões e sugestões de trabalhos futuros, já justificando os resultados obtidos neste capítulo.

5. Conclusões e Sugestões para Trabalhos Futuros

Neste trabalho realizou-se o estudo, simulação e desenvolvimento de software para controle de velocidade de motores de indução baseados em uma técnica de controle que combina o controle clássico e a inteligência artificial. Assim, foi projetado um conjunto híbrido que integrou técnicas de controle PID, juntamente com as técnicas de controle *Fuzzy*. Desse modo, a principal contribuição da dissertação é o estudo para implementação de PID adaptativo com acesso remoto por software em inversor de tensão comercial.

Para o desenvolvimento do trabalho, foi realizado um estudo dos conceitos básicos sobre técnicas de controle clássicas e de lógica *Fuzzy,* motores de indução trifásicos, eletrônica de potência e acionamentos elétricos, assim como as ferramentas computacionais utilizadas para a simulação e visualização dos resultados.

As ferramentas computacionais usadas nas simulações relativas ao estudo e análise de desempenho do controlador proposto foram o software *Matlab/Simulink*, especialmente o *Toolbox* de lógica *Fuzzy* e o uso de algumas das bibliotecas específicas de blocos como o *SimDriveline* e *SimPowerSystems*, onde foram visualizados os resultados por meio de gráficos, facilitando a análise.

O melhor resultado obtido com todos os testes foi utilizando a técnica de controle escalar, mas isso foi possível obter apenas na simulação, apesar da pequena diferença entre o controlador proposto e o controle com os ganhos do PID fixos. Em se tratando de sistemas críticos, caso seja possível implementar o controlador aqui proposto em outros modelos de inversores, que aceitem o ajuste do PID no controle escalar, torna o projeto viável para ser aplicado no meio industrial.

No trabalho aqui descrito chegou-se a conclusão que utilizar o controlador proposto, através da técnica de controle vetorial é eficaz, mas a diferença do resultado entre os ganhos do PID fixos e sendo ajustados pelo sistema se tornam insignificantes. Há a possibilidade de melhoramento caso seja feito um novo ajuste do fuzzy.

Na execução do software houve problemas com relação a escrita e leitura na porta serial, isso porque o tempo de resposta do inversor acaba sendo muito baixo, e em alguns momentos é necessário que software possa dar tempo do inversor responder, isso faz com que tenha-se poucas leituras da velocidade do motor. Para trabalhos futuros seria possível ler a velocidade através do próprio eixo do motor, para isso o aconselhável seria ler a velocidade do eixo e repassar para o software.

Uma das vantagens do controlador proposto é que o controle fuzzy é relativamente simples de implementar quando comparado a outras estratégias de controle não linear, isso possibilita controlar os ganhos do PID de maneira criteriosa.

Utilizando o conjunto de bibliotecas Qt foi possível realizar a comunicação serial de maneira simples e objetiva, inclusive oferecendo a possibilidade de utilizar o sistema em diversos sistemas operacionais, bastando compilar o programa no sistema interessado, Além disso o Qt possibilitou também desenhar com facilidade a interface utilizada pelo programa, apesar de ser uma interface simples, porém funcional.

5.1 Trabalhos Futuros

Abre-se um espectro bastante amplo para novas pesquisas. Nesse sentido, poderia pensar-se em desenvolver a estrutura de controle do acionamento, considerando desde os mecanismos de coleta de informações (sensoriamento da velocidade, por exemplo), o processamento dessas informações e a atuação do controle.

Além disso, para trabalhos futuros sugere-se também:

- Adaptar sistema híbrido proposto para outros inversores de "potência" comerciais que possuam interface serial. Apesar do fato de alguns inversores não possuírem interfaces seriais, ainda há a possibilidade de se criar uma interface que interprete a saída da porta serial e alimente entradas analógicas ou digitais presentes em praticamente todos inversores industriais;

- Utilizar outros métodos de fuzzificação e defuzzificação e tipos de funções de pertinência e compará-los com os resultados obtidos;

- Desenvolver outras metodologias de ajuste e sintonia para o controlador híbrido Fuzzy-PID;

Referências Bibliográficas

AHMED, A. *Eletrônica de potência.* Editora Prentice-Hall, Rio de Janeiro, 2001.

AKBIYIK, B.; EKSIN, I.; GUZELKAYA, M.; YESIL, E. Evaluation of The Performance of Various Fuzzy PID Controller Structures on Benchmark Systems, ELECO'2005, 4rd International Conf. on Electrical and Electronics Engineering, pp. 388-393, Bursa, Turkey, 2005.
AL-ODIENAT, A. I.; AL-LAWAMA, A. A. The advantages of PID Fuzzy controllers over the Conventional types, **American Journal of Applied Sciences**, p. 653-658, 2008.

BADR, B. M.; ELTAMALY, A. M.; ALOLAH, A. I. *Fuzzy controller for three phases induction motor drives*, 2010 International Conf on Autonomous and Intelligent Systems (AIS), pp. 1-6, Jun 2010.

BARANAUSKAS, M. C. C. *Procedimento, função, objeto ou lógica? Linguagens de programação vistas pelos seus paradigmas,* Departamento de Ciências da Computação, UNICAMP/NIED, Campinas, 1998.

BELLMAN, R.; ZADEH, L. A. Decision-making in a fuzzy environment. **Management Science**, v. 17, n. 4, p. 141-164, 1970.

BOLDEA, I. Control issues in adjustable speed drives. **IEEE Industrial Electronics Magazine**, v. 2, n. 3, p. 32–50, Setembro 2008.

BLASCKE, F. The principle of field orientation as applied to the new transvektor closed-loop control system for rotating field machines. **Siemens Review**, n. 5, p. 217–220, 1972.

DRIANKOV, D.; HELLENDOORN, H.; REINFRANK, M. *An Introduction to Fuzzy Control,* Springer-Verlag, 2 ed., Berlin, 1993.

DUTTA, S. Fuzzy Logic Applications: Technological a Strategic Issues, **IEEE Transactions on Engineering Management**, pp. 237-254, 1993.
ERENOGLU, I.; EKSIN I.; YESIL E.; GUZELKAYA, M. *An Intelligent Hybrid Fuzzy PID Controller*. 20th European Conference on Modelling and Simulation, 2006.

GONELLA, G. **Um Estudo Comparativo dos Harmônicos de Torque Gerados pelos Inversores de Tensão de Frequência Variável**, Dissertação (Mestrado em Engenharia Mecânica) - Departamento de Engenharia Mecânica, Universidade Federal Fluminense, Niterói - RJ, 2007.

JAMSA, K.; KLANDER, L. *Programando em C/C++ - A Bíblia*, Makron Books do Brasil Editora, São Paulo, 1999.

JOAQUIM, R. C. **Novas Tecnologias para comunicação entre o Chão de Fábrica e o Sistema Corporativo**. Dissertação (Mestrado em Engenharia Mecânica) - Departamento de Engenharia Mecânica, Universidade de São Paulo, São Carlos – SP, 2006.

KALHOODASHTI, H. E.; SHAHBAZIAN, M. Hybrid Speed Control of Induction Motor using PI and Fuzzy Controller , **International Journal of Computer Applications** , v. 30, n. 11, 2011.

KARANAYIL, B.; MUHAMMED, F.; GRANTHAN, C. Stator and rotor resistance observers for induction motor drive using fuzzy logic and artificial neural networks. **IEEE Transactions on Energy Conversion**, v. 40, n. 4, p. 771–780, 2005.

KRAUSE, P. C.; THOMAS, C. H. Simulation of Symmetrical Induction Machinery, **IEEE Trans. On Power Apparutus and Systems**, Vol. PAS-84, No 11, pp. 1038-1053, Nov. 1965.

KRAUSE, P. C.; WASYNCZUK, O.; SUDHOFF, S. D. *Analysis of Electric Machinery*. MacGraw-Hill, New York, 1986.

MAGALHÃES, N. C. H.; RAMOS, D. S.; SCHILLING, M. T. *Avaliação integrada de desempenho*, Anais IEEE INDUSCON 92, São Paulo, 1992.

MATTAVELLI, P.; ROSSETTO, L.; SPIAZZI, G.; TENTI, P. General-Purpose Fuzzy Controller for DC–DC Converters*,* **IEEE Applied Power Electronics Conference and Exposition**, vol. 2, pp. 723-730, 1995.

MENDEL, J. M. Fuzzy Logic Systems for Engineering: A Tutorial. **Proceedings of the IEEE**, v. 83, n. 3, pp. 345-377, Mar. 1995.

MENDEL, J. M. *Uncertain Rule-Based Fuzzy Logic Systems: Introduction and New Directions*. Prentice Hall, Londres, 2001.

MORAES, C.; CASTRUCCI P. *Engenharia de Automação Industrial*. Editora LTC, São Paulo, 2007.

MURUGANANTHAM , N.; PALANI S. Hybrid Fuzzy-PI Controller Based Speed Control of PMBLDC Motor using Soft-Switching Inverter , **European Journal of Scientific Research** , v. 78, n. 2, pp. 259-272 , 2012.

NARMADHA, T. V. ; THYAGARAJAN T. Fuzzy Logic Based Position-Sensorless Speed Control of Multi Level Inverter Fed PMBLDC Drive , **Journal of Advances in Information Technology**, v. 1, n. 1, pp. 52-58, Fev. 2010 .

NOVOTNY, D. W.; LIPO T. A. *Vector Control and Dynamics of AC Drives*. Claredon Press.

Oxford, 1996.

OGASARAWA, S.; AKAGI, H.; NABAE, A. The generalized theory of indirect vector control for ac machines. **IEEE Transactions on Industry Applications**, v. 24, n. 3, p. 470–478, Maio-Junho 1988.

OGATA, K. *Engenharia de controle moderno.* Editora Prentice-Hall, Rio de Janeiro, 2003.

ONG, C. M. *Dynamic Simulation of Electric Machinery Using Matlab/Simulink*. Prentice Hall

PTR, New Jersey, EUA, 1997.

ORTEGA, N. R. S. **Aplicação da teoria de conjuntos fuzzy a problemas da biomedicina**. Tese de doutorado (Ciências) – Instituto de Física da Universidade de São Paulo, 2001.

PAVLICA, V.; PETROVACKI, S. An Application of PID-Fuzzy Hybrid Controller. Proc. of 1998 IEEE International Conferenceon Control Application, Trieste, Itália, pp. 1-4, Set. 1998 .

PEREIRA, V. M.; POMILIO J. A. Frequency and Voltage Regulation of Induction Generator Driven by Internal Combustion Engine. **Anais do VI Congresso Brasileiro de Eletrônica de Potência**. In: *COBEP'2001*. Florianópolis (SC), Brasil, pp. 729-734, 2001.

PUPO, M. S. **Interface homem-máquina para supervisão de um CLP em controle de processos através da WWW**. Dissertação (Mestrado em Engenharia Elétrica) - Departamento de Engenharia Elétrica, Universidade de São Paulo, São Carlos – SP, 2002.

ROCHA, A. M. A. *Introdução a programação usando C*. 3. ed., Editora FCA, 2006.

RODRIGUES, W. **Critérios para o Uso Eficiente de Inversores de Frequência em Sistemas de Bombeamento de Água**. Tese de Doutorado (Engenharia Civil), 234 p., UNICAMP, Campinas, SP, 2007.

ROMEU, R. *Modelagem do motor de indução*. Relatório Técnico de Laboratório, Departamento de Engenharia Elétrica, Universidade Federal do Rio Grande do Sul, Porto Alegre – RS, 2006.

SANTOS, G. J. C. *Lógica Fuzzy*. Monografia de conclusão (Matemática) - Departamento de Ciências Exatas, Universidade Estadual de Santa Cruz, 2003.

SILER, W.; BUCLEY, J. J. *Fuzzy expert systems and fuzzy reasoning.* John Wiley and Sons, New York: 2005.

SILVA, J. Y. M. F; CRUZ, M. M. F; ROSADO, R. M. *Redes Industriais FIELDBUS*. Trabalho Acadêmico (Ciências da Computação) – Departamento de Ciências da Computação, Universidade de Brasília, 2006.

SINTHIPSOMBOON, K.; HUNSACHAROONROJ I.; KHEDARI J.; PONGAEN, W.; PRATUMSUWAN, P. A hybrid of fuzzy and fuzzy self-tuning PID controller for servo electro-hydraulic system. **6th IEEE Conference on Industrial Electronics and Applications (ICIEA)**, pp. 220-225, Jun. 2011.

SO, W.; TSE, C. K.; LEE, Y. S. Development of a Fuzzy Logic Controller for DC-DC Converters: Design, Computer Simulation, and Experimental Evaluation. **IEEE Transactions on Power Electronics**, v.11, pp. 24-32, 1996.

SOUZA, O. T. L. **Desenvolvimento de um Modelo Fuzzy para Determinação do Latente com Aplicação em Sistemas de Irrigação**. 2010. Dissertação (Mestrado em Engenharia Elétrica) - Departamento de Engenharia Engenharia, UNESP, 2004.

SZCZESNY, R.; RONKOWSKI, M. A. *New Equivalent Circuit Approach to Simulation of Converter - Induction Machine Associations*. Proceedings of the European Conference on Power Electronics and Applications (EPE'91), pp. 4/356-4/361, Firenze, Italy, 1991.

TANG, K. S.; MAN, K. F.; CHEN, G.; KWONG, S. An optimal fuzzy PID controller. **IEEE Transactions on Industry Application**, v. 48, pp. 757-765, 2001.

THAO, N. G. M.; DAT, M. T.; BINH, T. C.; PHUC N. H. PID-Fuzzy Logic Hybrid Controller For Grid-Connected Photovoltaic Inverters. **2010 International Forum on Strategic Technology (IFOST)**, pp.14-144, Out. 2010.

TIOBE *Programming Community Index for August 2011.* Disponível em: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html> Acesso em: 16 Ago. 2011.

TRIPURA P.; BABU Y. S. K. Fuzzy Logic Speed Control of Three Phase Induction Motor Drive , **World Academy of Science, Engineering and Technology**, n. 60, 2011.

VAS, P. *Vector Control of AC Machines*. Oxford University Press, London, UK, 1990.

VAS, P. *Sensorless Vector and Direct Torque Control*. Oxford University Press, London, UK, 1998.

ZADEH, L. A. *Fuzzy sets.* Information and Control, Berkley, pp. 338-353, 1965.

WAHYUNGGORO, O.; SAAD , N. Development of Fuzzy-logic-based Self Tuning PI Controller for Servomotor. **10th International Conference on Control, Automation, Robotics and Vision**, pp. 1545-1550, Dez. 2008.

WATT, D. *Programming Language Concepts and Paradigms*, Prentice-Hall International, Series in Computer Science, 1990.

WEG. Manual do inversor de freqüência CFW – 09. Disponível em: <http://www.weg.net/files/products/WEG-cfw-09-manual-do-usuario-0899.5298-4.0x-manual-portugues-br.pdf> Acesso em: 31 Ago. 2010.

WOODARD, M. A. *Fuzzy Open-Loop Attitude Control for the FAST Spacecraft.* **Proceedings of the NASA AIAA, Guidance, Navigation and Control Conference**, pp. 20-36, 1996.

ANEXO I – Código fonte da classe "*Nucleo*"

```
#include <QtGui>
#include<string.h>
#include<stdlib.h>
#include "nucleo.h"
#include "fuzzy.h"
#include <qtimer.h>
#include "qextserialport/qextserialport.h"
#include "qwt/mainwindow.h"
#include "libxl.h"

Nucleo::Nucleo(QWidget *parent) :
    QWidget(parent),
    ui(new Ui::Nucleo)
{
    ui->setupUi(this);

    #ifdef _TTY_WIN_
        ui->comboBoxPortName->addItem("COM1", QVariant::
                                      fromValue(QString("COM1")));
        ui->comboBoxPortName->addItem("COM2", QVariant::
                                      fromValue(QString("COM2")));
        ui->comboBoxPortName->setCurrentIndex(0); //COM1
    #endif

    #ifdef _TTY_POSIX_
        ui->comboBoxPortName->addItem("ttyUSB0", QVariant::
                                      fromValue(QString("/dev/ttyUSB0")))
;
        ui->comboBoxPortName->addItem("ttyUSB1", QVariant::
                                      fromValue(QString("/dev/ttyUSB1")))
;
        ui->comboBoxPortName->setCurrentIndex(0); //ttyUSB0
    #endif

        CommPort = new QextSerialPort();
        CommPort->setPortName(QString("/dev/ttyUSB0"));
        CommPort->setDataBits(DATA_7);
        CommPort->setParity(PAR_EVEN);
        CommPort->setFlowControl(FLOW_OFF);
        CommPort->setStopBits(STOP_1);
        CommPort->setBaudRate(BAUD9600);
        CommPort->open();

    ui->spinSpeed->setRange(100, 1800);
    ui->spinSpeed->setValue(500);

    receiveTimer = new QTimer(this);
    receiveTimer->setInterval( 0.1 );
    connect(receiveTimer, SIGNAL(timeout()), SLOT(chamaFuzzy()));
    tempoExec = new QTimer(this);
    tempoExec->setInterval( 0.01 );
    connect(tempoExec, SIGNAL(timeout()), SLOT(incTempo()));

}

Nucleo::~Nucleo()
{
    delete ui;
}
```

```
void Nucleo::changeEvent(QEvent *e)
{
    QWidget::changeEvent(e);
    switch (e->type()) {
    case QEvent::LanguageChange:
        ui->retranslateUi(this);
        break;
    default:
        break;
    }
}
void Nucleo::on_pushButtonStart_clicked()
{
    countSpeed=0;
    erroAnt=0;
    ui->comboBoxPortName->setEnabled(false);
    ui->pushButtonStop->setEnabled(true);
    ui->pushButtonStart->setEnabled(false);

    defaultPID();
    char BCC = 0x30^0x30^0x38^0x30^0x33^0x3D^0x00^0x03^0x00^0x03^0x03;
    char liga[]={0x04,0x41,0x02,0x30,0x30,0x38,0x30,0x33,0x3D,
                 0x00,0x03,0x00,0x03,0x03,BCC};
    int i = CommPort->writeData(liga,sizeof(liga));
    sleep(1);
    char buff[2];
    int numBytes = CommPort->bytesAvailable();
    if(numBytes > 0) {
        i = CommPort->readData(buff, 2);
    }
    BCC = 0x30^0x32^0x38^0x32^0x31^0x3D^0x00^0x03^0x06^0x0B^0x03;
    char p121[]={0x04,0x41,0x02,0x30,0x32,0x38,0x32,0x31,0x3D,
                 0x00,0x03,0x06,0x0B,0x03,BCC};
    CommPort->writeData(p121,sizeof(p121));
    sleep(1);
    numBytes = CommPort->bytesAvailable();
    if(numBytes > 0) {
        i = CommPort->readData(buff, 2);
    }
    numReferencia=875;
    tempoExec->start(0);
}
void Nucleo::on_pushButtonStop_clicked()
{
    ui->comboBoxPortName->setEnabled(true);
    ui->pushButtonStop->setEnabled(false);
    ui->pushButtonStart->setEnabled(true);
    char values[]={0x04,0x41,0x02,0x30,0x30,0x38,0x30,0x33,0x3D,0x00,
                 0x01,0x00,0x00,0x03,0x04};
    CommPort->writeData(values,sizeof(values));
    sleep(1);
    defaultPID();
    loadXLS();
    tempoExec->stop();
    receiveTimer->stop();
}

void Nucleo::chamaFuzzy() {...}
void Nucleo::defaultPID() {...}
void Nucleo::incTempo() {...}
void Nucleo::writePID(const char *pid, float ganho) {...}
float Nucleo::readSpeed() {...}
void Nucleo::loadXLS() {...}
```

ANEXO II – Código fonte da classe "*Fuzzy*"

```
#include<string.h>
#include<stdlib.h>

#include "fuzzy.h"

float Fuzzy::mainFuzzy(float Erro, float dErro, const char* pid) {...}

float Fuzzy::normaliza(float valorN, const char* parametroN) {...}

const char* Fuzzy::regrasSP(const char *erro, const char *derro,
                            float valoresFS1, float valoresFS2) {...}

const char* Fuzzy::regrasSI(const char *erro, const char *derro,
                            float valoresFS1, float valoresFS2) {...}

const char* Fuzzy::regrasSD(const char *erro, const char *derro,
                            float valoresFS1, float valoresFS2) {...}

void Fuzzy::ativar(const char* result, float valorRI){...}

const char* Fuzzy::fuzzyficacao(float valorC, char indice){...}

float Fuzzy::defuzzyficacao() {...}
```

ANEXO III – Código fonte da função "*defaultPID*"

```
void Nucleo::defaultPID()
{
    int i, numBytes;
    char buff[2];
    char BCC = 0x30^0x32^0x38^0x36^0x31^0x3D^0x00^0x00^0x04^0x0A^0x03;
    char p161[]={0x04,0x41,0x02,0x30,0x32,0x38,0x36,0x31,0x3D,
                 0x00,0x00,0x04,0x0A,0x03,BCC};
    i = CommPort->writeData(p161,sizeof(p161));
    sleep(1);
    numBytes = CommPort->bytesAvailable();
    if(numBytes > 0) {
        i = CommPort->readData(buff, 2);
    }
    BCC = 0x30^0x32^0x38^0x36^0x32^0x3D^0x00^0x00^0x01^0x07^0x03;
    char p162[]={0x04,0x41,0x02,0x30,0x32,0x38,0x36,0x32,0x3D,
                 0x00,0x00,0x01,0x07,0x03,BCC};
    CommPort->writeData(p162,sizeof(p162));
    sleep(1);
    numBytes = CommPort->bytesAvailable();
    if(numBytes > 0) {
        i = CommPort->readData(buff, 2);
    }
    BCC = 0x30^0x32^0x38^0x36^0x36^0x3D^0x00^0x00^0x00^0x00^0x03;
    char p166[]={0x04,0x41,0x02,0x30,0x32,0x38,0x36,0x36,0x3D,
                 0x00,0x00,0x00,0x00,0x03,BCC};
    CommPort->writeData(p166,sizeof(p166));
    sleep(1);
    numBytes = CommPort->bytesAvailable();
    if(numBytes > 0) {
        i = CommPort->readData(buff, 2);
    }
}
```

ANEXO IV – Código fonte da função "*chamaFuzzy*"

```
void Nucleo::chamaFuzzy()
{
    Erro=0;

    dErro=0;
    Fuzzy* fuzzy=new Fuzzy();
    float saidaFS=0;
    float Errof, dErrof;

    readSpeed();

    Erro=numReferencia-numVelocidade;
    if(Erro<0)
    Erro=0;

    dErro=Erro-ErroAnt;
    ErroAnt=Erro;

    Errof = float (Erro);
    dErrof = float (dErro);

    saidaFS=fuzzy->mainFuzzy(Errof, dErrof, "SP");
    writePID("P", saidaFS);

    readSpeed();

    Erro=numReferencia-numVelocidade;
    if(Erro<0)
         Erro=0;

    dErro=Erro-ErroAnt;
    ErroAnt=Erro;

    Errof = float (Erro);
    dErrof = float (dErro);

    saidaFS=fuzzy->mainFuzzy(Errof, dErrof, "SI");
    writePID("I", saidaFS);

    readSpeed();

    Erro=numReferencia-numVelocidade;
    if(Erro<0)
         Erro=0;

    dErro=Erro-ErroAnt;
    ErroAnt=Erro;

    Errof = float (Erro);
    dErrof = float (dErro);

    saidaFS=fuzzy->mainFuzzy(Errof, dErrof, "SD");
    writePID("D", saidaFS);
}
```

ANEXO V – Código fonte da função "*readSpeed*"

```
float Nucleo::readSpeed()
{
    numVelocidade=0;
    char buffA[20];
    int numBytesB=0;
    numBytesB = CommPort->bytesAvailable();
    if(countSpeed >= 2) {
         CommPort->readData(buffA, 1);
    }
    sleep(1);

    char leVelocidade[]={0x04,0x41,0x30,0x31,0x38,0x30,0x32,0x05};
    int e = CommPort->writeData(leVelocidade,sizeof(leVelocidade));
    int numBytesA = CommPort->bytesAvailable();
    sleep(1);
    if(numBytesA > 0) {
      if(numBytesA > 15) numBytesA = 15;
      char buff[15];

      int i = CommPort->readData(buff, numBytesA);
      buff[numBytesA] = '\0';
      numVelocidade=(buff[8] << 12) | (buff[9] << 8) |
                        (buff[10] << 4) | buff[11];
      ui->lcdSpeed->display(numVelocidade);
      QString msg = buff;
    }
    vectorSpeed[countSpeed]=numVelocidade;
    vectorTime[countSpeed]=varTempo;
    countSpeed=countSpeed+1;

    if (varTempo==20){
        numReferencia=1750;
        char BCC = 0x30^0x32^0x38^0x32^0x31^0x3D^0x00^0x06^0x0D^0x06^0x03;
        char p121[]={0x04,0x41,0x02,0x30,0x32,0x38,0x32,0x31,0x3D,
                        0x00,0x06,0x0D,0x06,0x03,BCC};
        int i = CommPort->writeData(p121,sizeof(p121));
        sleep(1);
        char buff[2];
        int numBytes = CommPort->bytesAvailable();
        if(numBytes > 0) {
            i = CommPort->readData(buff, 2);
        }
    }
    if (varTempo==40){
        numReferencia=875;
        char BCC = 0x30^0x32^0x38^0x32^0x31^0x3D^0x00^0x03^0x06^0x0B^0x03;
        char p121[]={0x04,0x41,0x02,0x30,0x32,0x38,0x32,0x31,0x3D,
                        0x00,0x03,0x06,0x0B,0x03,BCC};
        int i = CommPort->writeData(p121,sizeof(p121));
        sleep(1);
        char buff[2];
        int numBytes = CommPort->bytesAvailable();
        if(numBytes > 0) {
            i = CommPort->readData(buff, 2);
        }
    }
    return numVelocidade;
}
```

ANEXO VI – Código fonte da função "*mainFuzzy*"

```
float Fuzzy::mainFuzzy(float Erro, float dErro, const char* pid)
{
    float ErroN=0, dErroN=0, valoresF[3];

    ErroN=normaliza(Erro, "E");
    dErroN=normaliza(dErro, "dE");

     const char* ErroFP = fuzzyficacao(ErroN, 'P');
     const char* ErroFS = fuzzyficacao(ErroN, 'S');
     valoresF[0]=valorF1;
     valoresF[1]=valorF2;

     const char* dErroFP = fuzzyficacao(dErroN,'P');
     const char* dErroFS = fuzzyficacao(dErroN,'S');
     valoresF[2]=valorF1;
     valoresF[3]=valorF2;

     ativadas[0]=0;
     ativadas[1]=0;
     ativadas[2]=0;
     ativadas[3]=0;
     ativadas[4]=0;
     ativadas[5]=0;
     ativadas[6]=0;
     SaidaFS=0;

     if ((pid="SP"))
     {
        const char* resultSP1 = regrasSP(ErroFP, dErroFP, valoresF[0],
                        valoresF[2]);
        ativar(resultSP1, valorI);
        const char* resultSP2 = regrasSP(ErroFP, dErroFS, valoresF[0], v
                  aloresF[3]);
        ativar(resultSP2, valorI);
        const char* resultSP3 = regrasSP(ErroFS, dErroFP, valoresF[1],
                        valoresF[2]);
        ativar(resultSP3, valorI);
        const char* resultSP4 = regrasSP(ErroFS, dErroFS, valoresF[1],
                        valoresF[3]);
        ativar(resultSP4, valorI);

        SaidaFS=normaliza(defuzzyficacao(), "SP");
     }

     if ((pid="SI")) {...}
     if ((pid="SD")) {...}

    return SaidaFS;

}
```

ANEXO VII – Código fonte da função "*normaliza*"

```
float Fuzzy::normaliza(float valorN, const char* parametroN)
{
    float nA=0, valorC=0;
    if (strcmp(parametroN,"E")==0)
      {
        nA=0.000555556; //Máximo 1800 RPM, Minimo -1800 RPM 2/(1800-(-1800)
Resultado: 0,000555556
        valorC=(nA)*(valorN);
      }
    if (strcmp(parametroN,"dE")==0)
      {
        nA=0.000555556; //Máximo 1800 RPM, Minimo -1800 RPM 2/(1800-(-1800)
Resultado: 0,000555556
        valorC=(nA)*(valorN);
      }
    if (strcmp(parametroN,"SP")==0)
      {
        nA=10; //Máximo 10, Minimo -10 Resultado: (10-(-10))/2=10
        valorC=((nA)*(valorN))+15;
      }
    if (strcmp(parametroN,"SD")==0)
      {
        nA=1; //Máximo 10, Minimo -1 Resultado: (1-(-1))/2=10
        valorC=(nA)*(valorN);
      }
    if (strcmp(parametroN,"SI")==0)
      {
        nA=0.1; //Máximo 0.1, Minimo -0.1 Resultado: (0.1-(-0.1))/2=10
        valorC=(nA)*(valorN);
       }

    return valorC;
}
```

ANEXO VIII – Código fonte da função "*fuzzyficacao*"

```
const char* Fuzzy::fuzzyficacao(float valorC, char indice)
  {
      float a[12]={-3.005710851,3.005710851,-3.00120048,3.00120048,-
2.993115834,2.993115834,-2.996703626,2.996703626,-3.006614552,3.006614552,-
2.996703626,2.996703626};
      float b[12]={-2.005710851,3.005710851,-
1.00270108,2.00270108,0,1,1,0,2.003307276,-1.003307276,2.996703626,-
1.99670362601139};
      const char* result = (char *) malloc (15);
          if ((valorC>=-1)&&(valorC<-0.6673))
          {   valorF1=(a[0]*valorC)+b[0];
              valorF2=(a[1]*valorC)+b[1];
              if (indice=='P'){ result="NG"; }
              else if (indice=='S') { result="NM"; }
              else { result="NULL"; } }
          else if ((valorC>=-0.6673)&&(valorC<-0.3341))
          {   valorF1=(a[2]*valorC)+b[2];
              valorF2=(a[3]*valorC)+b[3];
              if (indice=='P') { result="NM"; }
              else if (indice=='S') { result="NP"; }
              else { result="NULL"; } }
          else if ((valorC>=-0.3341)&&(valorC<0))
          {   valorF1=(a[4]*valorC)+b[4];
              valorF2=(a[5]*valorC)+b[5];
              if (indice=='P'){ result="NP"; }
              else if (indice=='S') { result="Z"; }
              eles { result="NULL"; } }
          else if ((valorC>=0)&&(valorC<0.3337))
          {   valorF1=(a[6]*valorC)+b[6];
              valorF2=(a[7]*valorC)+b[7];
              if (indice=='P') { result="Z"; }
              else if (indice=='S') { result="PP"; }
              eles { result="NULL"; } }
          else if ((valorC>=0.3337)&&(valorC<0.6663))
          {   valorF1=(a[8]*valorC)+b[8];
              valorF2=(a[9]*valorC)+b[9];
              if (indice=='P') { result="PP"; }
              else if (indice=='S') { result="PM"; }
              eles { result="NULL"; } }
          else if ((valorC>=0.6663)&&(valorC<=1))
          {   valorF1=(a[10]*valorC)+b[10];
              valorF2=(a[11]*valorC)+b[11];
              if (indice=='P') { result="PM"; }
              else if (indice=='S')
                  result="PG";
                  result="NULL"; }

          return result;  //Retorna a variavel no universo Fuzzy
          delete[] result;
  }
```

ANEXO IX – Código fonte da função "*saidaSP*"

```
const char* Fuzzy::regrasSP(const char *erro, const char *derro, float
valoresFSP1, float valoresFSP2)
    { const char* saida = (char *) malloc (15);
      if (strcmp(erro,"NG")==0 && strcmp(derro,"Z")==0)
        { saida="NG"; }
      else if ((strcmp(erro,"NM")==0)&&(strcmp(derro,"Z")==0))
        { saida="NM"; }
      else if ((strcmp(erro,"NP")==0)&&(strcmp(derro,"Z")==0))
        { saida="NP"; }
      else if ((strcmp(erro,"Z")==0)&&(strcmp(derro,"Z")==0))
        { saida="PG"; }
      else if ((strcmp(erro,"PP")==0)&&(strcmp(derro,"Z")==0))
        { saida="PP"; }
      else if ((strcmp(erro,"PM")==0)&&(strcmp(derro,"Z")==0))
        { saida="PM"; }
      else if ((strcmp(erro,"PG")==0)&&(strcmp(derro,"Z")==0))
        { saida="PG"; }
      else if ((strcmp(erro,"Z")==0)&&(strcmp(derro,"NG")==0))
        { saida="PG"; }
      else if ((strcmp(erro,"Z")==0)&&(strcmp(derro,"NM")==0))
        { saida="PG"; }
      else if ((strcmp(erro,"Z")==0)&&(strcmp(derro,"NP")==0))
        { saida="PM"; }
      else if ((strcmp(erro,"Z")==0)&&(strcmp(derro,"PP")==0))
        { saida="PP"; }
      else if ((strcmp(erro,"Z")==0)&&(strcmp(derro,"PM")==0))
        { saida="PM"; }
      else if ((strcmp(erro,"Z")==0)&&(strcmp(derro,"PG")==0))
        { saida="PG"; }
      else
      { saida="NULL"; }
      valorI=min(valoresFSP1, valoresFSP2);
      return(saida);
      delete[] saida;
}
```

ANEXO X – Código fonte da função "*saidaSI*"

```
const char* Fuzzy::regrasSI(const char *erro, const char *derro, float
valoresFSP1, float valoresFSP2)
    { const char* saida = (char *) malloc (15);
      if (strcmp(erro,"NG")==0 && strcmp(derro,"Z")==0)
        {
            saida="NG";
        }
      else if ((strcmp(erro,"NM")==0)&&(strcmp(derro,"Z")==0))
        {
            saida="NM";
        }
      else if ((strcmp(erro,"NP")==0)&&(strcmp(derro,"Z")==0))
        {
            saida="NP";
        }
      else if ((strcmp(erro,"Z")==0)&&(strcmp(derro,"Z")==0))
        {
                saida="Z";
        }
      else if ((strcmp(erro,"PP")==0)&&(strcmp(derro,"Z")==0))
        {
                saida="PP";
        }
      else if ((strcmp(erro,"PM")==0)&&(strcmp(derro,"Z")==0))
        {
                saida="PM";
        }
      else if ((strcmp(erro,"PG")==0)&&(strcmp(derro,"Z")==0))
        {
                saida="PG";
        }
      else if ((strcmp(erro,"Z")==0)&&(strcmp(derro,"NP")==0))
        {
                saida="PP";
        }
      else if ((strcmp(erro,"Z")==0)&&(strcmp(derro,"PM")==0))
        {
                saida="PP";
        }
      else if ((strcmp(erro,"Z")==0)&&(strcmp(derro,"PG")==0))
        {
                saida="PP";
        }
      else
      {
        saida="NULL";
      }
      valorI=min(valoresFSP1, valoresFSP2);

      return(saida);
      delete[] saida;

}
```

ANEXO XI – Código fonte da função "*saidaSD*"

```
const char* Fuzzy::regrasSD(const char *erro, const char *derro, float
valoresFSP1, float valoresFSP2)
    {
      const char* saida = (char *) malloc (15);
      if (strcmp(erro,"NG")==0 && strcmp(derro,"Z")==0)
      {
            saida="NG";
      }
    else if ((strcmp(erro,"NM")==0)&&(strcmp(derro,"Z")==0))
      {
            saida="NM";
      }
    else if ((strcmp(erro,"NP")==0)&&(strcmp(derro,"Z")==0))
      {
            saida="NP";
      }
    else if ((strcmp(erro,"Z")==0)&&(strcmp(derro,"Z")==0))
      {
            saida="PP";
      }
    else if ((strcmp(erro,"PP")==0)&&(strcmp(derro,"Z")==0))
      {
            saida="PP";
      }
    else if ((strcmp(erro,"PM")==0)&&(strcmp(derro,"Z")==0))
      {
            saida="PP";
      }
    else if ((strcmp(erro,"PG")==0)&&(strcmp(derro,"Z")==0))
      {
            saida="PG";
      }
    else if ((strcmp(erro,"PG")==0)&&(strcmp(derro,"Z")==0))
      {
            saida="PG";
      }
    else if ((strcmp(erro,"Z")==0)&&(strcmp(derro,"NP")==0))
      {
            saida="Z";
      }
    else
    {
      saida="NULL";  // Caso nenhuma regra seja ativada
    }
    valorI=min(valoresFSP1, valoresFSP2);

    return(saida);
    delete[] saida;

}
```

ANEXO XII – Código fonte da função "*ativar*"

```
void Fuzzy::ativar(const char* result, float valorRI)
{
    if (strcmp(result,"NULL")!=0)
    {
        if (strcmp(result,"NG")==0)
        {
           if (ativadas[0]<=valorRI)
           {
               ativadas[0]=valorRI;
           }
        }
        else if (strcmp(result,"NM")==0)
        {
           if (ativadas[1]<=valorRI)
           {
               ativadas[1]=valorRI;
           }
        }
        else if (strcmp(result,"NP")==0)
        {
           if (ativadas[2]<=valorRI)
           {
               ativadas[2]=valorRI;
           }
        }
        else if (strcmp(result,"Z")==0)
        {
           if (ativadas[3]<=valorRI)
           {
               ativadas[3]=valorRI;
           }
        }
        else if (strcmp(result,"PP")==0)
        {
           if (ativadas[4]<=valorRI)
           {
               ativadas[4]=valorRI;
           }
        }
        else if (strcmp(result,"PM")==0)
        {
           if (ativadas[5]<=valorRI)
           {
               ativadas[5]=valorRI;
           }
        }
        else if (strcmp(result,"PG")==0)
        {
           if (ativadas[6]<=valorRI)
           {
               ativadas[6]=valorRI;
           }
        }
    }
}
```

ANEXO XIII – Código fonte da função "*defuzzificacao*"

```
float Nucleo::defuzzyficacao()
{
  float a[12]={-3.005710851,3.005710851,-3.00120048,3.00120048,-
2.993115834,2.993115834,-2.996703626,2.996703626,-3.006614552,3.006614552,-
2.996703626,2.996703626};
  float b[12]={-2.005710851,3.005710851,-
1.00270108,2.00270108,0,1,1,0,2.003307276,-1.003307276,2.996703626,-
1.99670362601139};

  float saida=0,areatrapezio=0,centro=0,AREACENTRO=0,SOMAAREA=0,
areatriangulo=0, arearetangulo=0;
  float X1=0, X2=0, X3=0, X4=0;
  int j=0,i=0,control=0;

  for(int cont=0;cont<=6;cont++)
  {
       if ((i==0)&&(ativadas[i]!=0))
       {
           X1=-1;
           X2=((ativadas[i]-b[0])/a[0]);
           X3=((0-b[0])/a[0]);

           areatriangulo=(X3-X2)*ativadas[i];
           arearetangulo=((X2-X1)*ativadas[i]);
           areatrapezio=areatriangulo+arearetangulo;
           centro=(X3+1)/2;

           AREACENTRO=AREACENTRO+(centro*areatrapezio);
           SOMAAREA=SOMAAREA+areatrapezio;

           control=1;

       }
       if ((i>=1)&&(i<=5)&&(ativadas[i]!=0))
       {
           X1=((0-b[j])/a[j]);
           X2=((ativadas[i]-b[j])/a[j]);
           X3=((ativadas[i]-b[j+1])/a[j+1]);
           X4=((0-b[j+1])/a[j+1]);

            areatriangulo=((((X2-X1)*ativadas[i])/2)+
                           (((X4-X3)*ativadas[i])/2));
           arearetangulo=((X3-X2)*ativadas[i]);
           areatrapezio=areatriangulo+arearetangulo;
           centro=(X4+X1)/2;

           AREACENTRO=AREACENTRO+(centro*areatrapezio);
           SOMAAREA=SOMAAREA+areatrapezio;

           control=1;

       }
```

```
        if ((ativadas[i+1]!=0)&&(control==1)&&(cont<=5))
        {
            X1=0, X2=0, X3=0, X4=0;
            arearetangulo=0, areatriangulo=0, areatrapezio=0;

            X1=((0-b[j+2])/a[j+2]);
            X2=((ativadas[i+1]-b[j+2])/a[j+2]);
            X3=((ativadas[i+1]-b[j+3])/a[j+3]);
            X4=((0-b[j+3])/a[j+3]);

            areatriangulo=((((X2-X1)*ativadas[i+1])/2)+
                           (((X4-X3)*ativadas[i+1])/2));
            arearetangulo=((X3-X2)*ativadas[i+1]);
            areatrapezio=areatriangulo+arearetangulo;
            centro=(X4+X1)/2;
            AREACENTRO=AREACENTRO+(centro*areatrapezio);
            SOMAAREA=SOMAAREA+areatrapezio;
            if (ativadas[i+2]==0)
            {
                cont=10;
             }
            else
            {
                X1=0, X2=0, X3=0, X4=0;
                arearetangulo=0, areatriangulo=0, areatrapezio=0;

                X1=((0-b[j+4])/a[j+4]);
                X2=((ativadas[i+2]-b[j+4])/a[j+4]);
                X3=((ativadas[i+2]-b[j+5])/a[j+5]);
                X4=((0-b[j+5])/a[j+5]);

                areatriangulo=((((X2-X1)*ativadas[i+2])/2)+
                               (((X4-X3)*ativadas[i+2])/2));
                arearetangulo=((X3-X2)*ativadas[i+2]);
                areatrapezio=areatriangulo+arearetangulo;
                centro=(X4+X1)/2;
                AREACENTRO=AREACENTRO+(centro*areatrapezio);
                SOMAAREA=SOMAAREA+areatrapezio;
                cont=10;
               }
        }
        if ((i==6)&&(ativadas[i]!=0))
        {
            X1=((0-b[11])/a[11]);
            X2=((ativadas[i]-b[11])/a[11]);
            X3=1;
            areatriangulo=(((X2-X1)*ativadas[i])/2);
            arearetangulo=((X3-X2)*ativadas[i]);
            areatrapezio=areatriangulo+arearetangulo;
            centro=(1-X1)/2;
            AREACENTRO=AREACENTRO+(centro*areatrapezio);
            SOMAAREA=SOMAAREA+areatrapezio;
        }
        if (i==0)
        { j=1; }
        else
        { j=j+2; }
        i=i+1;
    }
    saida=AREACENTRO/SOMAAREA;
    return saida;
}
```

ANEXO XIV – Código fonte da função "*writePID*"

```
void Fuzzy::writePID(const char *pid, float ganho)
{
    int cont=0;
    int setOk1=1;
    if(ganho<0)
      {
           ganho=0-ganho;
      }
    if(strcmp(pid,"P")==0) //Escreve ganho do proporcional (P161)
    {
        if (ganho>=64)
        {
            ganho=63.9;
        }
        int inteiro = (int)floor(ganho);
        float decimaltemp = ganho-((float)inteiro);
        decimaltemp = decimaltemp*10;
        int decimal = (int)decimaltemp;
        while (setOk1!=0){
            char buffdata[15];
            buffdata[0]=0x30;
            buffdata[1]=0x32;
            buffdata[2]=0x38;
            buffdata[3]=0x36;
            buffdata[4]=0x31;
            buffdata[5]=0x33;
            buffdata[6]=0x30;
            buffdata[7]=(inteiro >> 3);
            buffdata[8]=inteiro & 7;
            buffdata[9]=decimal;
            buffdata[10]=0x03;

            char BCC = buffdata[0]^buffdata[1]^buffdata[2]^buffdata[3]\
^buffdata[4]^buffdata[5]^buffdata[6]^buffdata[7]^buffdata[8]^buffdata[9]\
^buffdata[10];
            char p161[] = {0x04,0x41,0x02,buffdata[0],buffdata[1],\
buffdata[2],buffdata[3],buffdata[4],buffdata[5],buffdata[6],buffdata[7],\
buffdata[8],buffdata[9],buffdata[10],BCC};

            int i = CommPort->writeData(p161,sizeof(p161));
            sleep(1);
            char buff[2];
            int numBytes = CommPort->bytesAvailable();
            if(numBytes > 0) {
                i = CommPort->readData(buff, 2);
                if (((buff[0]==0x41)&&(buff[1]==0x06))||(cont>=5)){
                    setOk1=0;
                }
            }

            cont++;

            if (cont>5)
            {
                setOk1=0;
            }
        }
    }
```

```
if(strcmp(pid,"I")==0)
    {   if (ganho>=10) { ganho=9.999; }
        int inteiro = (int)floor(ganho);
        float decimaltemp = ganho-((float)inteiro);
        decimaltemp = decimaltemp*1000;
        int decimal = (int)decimaltemp;
        while (setOk1!=0){
            char buffdata[15];
            buffdata[0]=0x30;\ buffdata[1]=0x32;\ buffdata[2]=0x38;
            buffdata[3]=0x36;\ buffdata[4]=0x32;\ buffdata[5]=0x33;
            buffdata[6]=(inteiro);\ buffdata[7]=(decimal >> 6);
            buffdata[8]=(decimal >> 3);\ buffdata[9]=decimal & 15;
            buffdata[10]=0x03;
            char BCC = buffdata[0]^buffdata[1]^buffdata[2]^buffdata[3]^\
buffdata[4]^buffdata[5]^buffdata[6]^buffdata[7]^buffdata[8]^buffdata[9]^\
buffdata[10];
            char p162[]={0x04,0x41,0x02,buffdata[0],buffdata[1],\
buffdata[2],buffdata[3],buffdata[4],buffdata[5],buffdata[6],buffdata[7],\
buffdata[8],buffdata[9],buffdata[10],BCC};
            int i = CommPort->writeData(p162,sizeof(p162));
            char buff[2];
            int numBytes = CommPort->bytesAvailable();
            if(numBytes > 0) {
                i = CommPort->readData(buff, 2);
                if (((buff[0]==0x41)&&(buff[1]==0x06))||(cont>=5))
                   { setOk1=0; } }
            cont++;
            if (cont>5)
            { setOk1=0; } } }
    if(strcmp(pid,"D")==0)
    {   if(ganho>=10);
        { ganho=7.99; }
        int inteiro = (int)floor(ganho);
        float decimaltemp = ganho-((float)inteiro);
        decimaltemp = decimaltemp*100;
        int decimal = (int)decimaltemp;
        while (setOk1!=0){
            char buffdata[15];
            buffdata[0]=0x30;\ buffdata[1]=0x32;\ buffdata[2]=0x38;
            buffdata[3]=0x36;\ buffdata[4]=0x36;\ buffdata[5]=0x33;
            buffdata[6]=0x30;\ buffdata[7]=inteiro;\
           buffdata[8]=(decimal>>3);\ buffdata[9]=decimal & 15;
            buffdata[10]=0x03;
            char BCC = buffdata[0]^buffdata[1]^buffdata[2]^buffdata[3]^\
buffdata[4]^[5]^buffdata[6]^buffdata[7]^buffdata[8]^buffdata[9]^\
buffdata[10];
            char p166[]={0x04,0x41,0x02,buffdata[0],buffdata[1],\
buffdata[2],buffdata[3],buffdata[4],buffdata[5],buffdata[6],buffdata[7],\
buffdata[8],buffdata[9],buffdata[10],BCC};
            int i = CommPort->writeData(p166,sizeof(p166));
            qDebug("trasmitted : %x (P166)", i);
            char buff[2];
            int numBytes = CommPort->bytesAvailable();
            if(numBytes > 0) {
                i = CommPort->readData(buff, 2);
                if (((buff[0]==0x41)&&(buff[1]==0x06))||(cont>=5))
                { setOk1=0; } }
            cont++;
            if (cont>5)
            { setOk1=0; } } } }
```