TRABALHO DE GRADUAÇÃO

Desenvolvimento de Interface de Comunicação Baseada no Protocolo Modbus para Conexão de um Computador (PC) a um Inversor de Frequência

Matheus Henrique Dinato Menezes

Victor Yuji Sato

Brasília, Julho 2019

ENGENHARIA
MECATRÔNICA
UNIVERSIDADE DE BRASÍLIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia
Curso de Graduação em Engenharia de Controle e Automação

TRABALHO DE GRADUAÇÃO

Desenvolvimento de Interface de Comunicação Baseada no Protocolo Modbus para Conexão de um Computador (PC) a um Inversor de Frequência

Matheus Henrique Dinato Menezes

Victor Yuji Sato

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

Banca Examinadora

Professor Lélio Ribeiro Soares Júnior, ENE/UnB
Orientador _____

Professor Alex Reis, Dr. - UnB
Examinador Interno _____

Professor Pedro Henrique Franco Moraes, MSc. - IFG
Examinador Interno _____

Brasília, Julho 2019

FICHA CATALOGRÁFICA

MENEZES, M.H.D.; SATO, V.Y.
Desenvolvimento de uma interface de comunicação baseada no protocolo Modbus para conexão de um computador a um inversor de frequências. / Matheus Henrique Dinato Menezes; Victor Yuji Sato; Orientador: Lélio Ribeiro Soares Júnior

[Distrito Federal] 2019.

XI, 46p, 210x297 mm (FT/UnB, Engenheiro, Controle e Automação, 2019). Trabalho de Graduação – Universidade de Brasília. Faculdade de Tecnologia.

1.Protocolo Modbus 2.Comunicação Serial
3.Controle 4.Emulação

I. Mecatrônica/FT/UnB II. Desenvolvimento de Interface de Comunicação Baseada no Protocolo Modbus para Conexão de um Computador a um Inversor de Frequência

REFERÊNCIA BIBLIOGRÁFICA

MENEZES, M.H.D., SATO, V.Y. (2019) Desenvolvimento de uma interface de comunicação baseada no protocolo Modbus para conexão de um computador a um inversor de frequências. Trabalho de Graduação em Engenharia de Controle e Automação, Publicação FT. TG-$n°$13, Faculdade de Tecnologia, Universidade de Brasília, Brasília, DF, 57p.

CESSÃO DE DIREITOS

AUTOR: Matheus Henrique Dinato Menezes e Victor Yuji Sato

TÍTULO DO TRABALHO DE GRADUAÇÃO: Desenvolvimento de uma interface de comunicação baseada no protocolo Modbus para conexão de um computador a um inversor de frequências.

GRAU: Engenheiro ANO: 2019

Matheus Henrique Dinato Menezes e Victor Yuji Sato

Campus Darcy Ribeiro, FT, Universidade de Brasília.

70919-970 Brasília – DF – Brasil.

Dedicatórias

Dedico à minha família, principalmente meus pais, por toda assistência necessária para chegar tão longe.

Victor Yuji Sato

Dedico à minha família e professores que me guiaram e me ensinaram muito durante toda a minha jornada.

Matheus Henrique Dinato Menezes

Agradecimentos

Agradeço aos meus queridos amigos Fred, Primo, Saulo, Yuji, Zago e Duas, pelo apoio manifestado nessa reta final. E em especial a minha namorada Letícia por toda ajuda. E a turma 30 de engenharia mecatrônica.

Matheus Henrique Dinato Menezes

Agradeço à minha querida namorada Geovana e a muitos de meus queridos amigos, principalmente Alexandre, Anderson, Aramiz, Clara, Ciro, Fred, Guilherme Victor, Matheus, Saulo por todo apoio fornecido durante minha jornada. Agradeço também a todas as pessoas que participaram de alguma forma de minha vida acadêmica. Considero que todos tiveram um papel essencial para o meu crescimento, como pessoa e como futuro engenheiro.

Victor Yuji Sato

RESUMO

O crescente aproveitamento de energias consideradas renováveis é notável nos últimos anos. Dentre as energias renováveis destaca-se a energia eólica, uma alternativa limpa e inesgotável. O número de instalações de parques eólicos brasileiros vem aumentando significantemente, indicando o quão forte é o potencial eólico existente no nosso país. Isto indica, também, que estudos de viabilidade de novas instalações são de extrema importância e devem ser feitos a fim de alcançar a geração de energia elétrica máxima

Este trabalho apresenta uma solução para estudos de viabilidade de instalações de usinas eólicas através da emulação de turbinas eólicas. Presente no Laboratório de Qualidade de Energia Elétrica, na Universidade de Brasília, há uma bancada capaz de emular turbinas eólicas de velocidade variável. A bancada é composta por um gerador síncrono de imã permanente, um motor de indução trifásico e o inversor de frequência CFW 11, da fabricante WEG. A emulação é executada utilizando uma interface gráfica, executada em um computador, onde um usuário configura as variações de velocidades desejadas. Protocolo de comunicação serial Modbus é utilizado para troca de mensagens entre computador e inversor.

Palavras Chave: Modbus, Comunicação Serial, Controle, Emulação

ABSTRACT

The rising use of energy considered renewable is notable in the last years. Among the types of renewable energy we can highlight the wind energy, a clean and inexhaustible alternative. The number of wind parks installations in Brazil is increasing significantly, indicating how strong is the wind energy potential in our country. In addition, viability studies of future installations is extremely important and have to be done in order to generate electricity in a more optimized way.

This study presents a solution for viability studies of future wind parks through wind turbine emulation. The Quality of Electrical Energy Laboratory, in University of Brasília, have an workbench designed to emulate variable speed wind turbines. The workbench is composed by a permanent magnet synchronous generator, a three phase induction motor and the frequency inverter CFW 11, from WEG. The emulation is executed using a GUI in a computer, where the user is able to configure the speed variations. Serial communication is made by using Modbus protocol in order to exchange message between computer and inverter.

Keywords: Modbus,Serial Communication, Controlling, Emulation

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Acrônimos

PC	Personal Computer
PMSG	Permanent Magnet Synchronous Generator
SIM	Sistemas Integrados de Manufatura
TSR	Tip Speed Ratio
CPU	Central Processing Unit
GUI	Graphical User Interface
IHM	Interface Homem-Máquina
MIT	Motor de Indução Trifásico
IGBT	Insulated Gate Bipolar Transistor
RO	Read-only
CRC	Cyclic Redundancy Check

Capítulo 1

Introducão

A energia eólica é denominada como a energia cinética fornecida pela movimentação de massas de ar ou, em outras palavras, o próprio vento. É possível coletar este tipo de energia através de usinas eólicas, compostas por diversas turbinas capazes de gerar energia elétrica através da rotação de suas pás causadas pelos ventos da região em que está instalada. Determina-se os locais mais adequados e ótimos para instalações de usinas eólicas através do estudo dos perfis de ventos de diversas regiões [1].

Apesar da energia eólica corresponder a apenas 1,7% da capacidade instalada no Brasil, sua participação na matriz energética brasileira dobrou desde 2011 [2]. Por se tratar de uma energia renovável, a sua utilização é de suma importância para o futuro, tendo em vista que o vento é um fenômeno da natureza de ocorrência constante e de disponibilidade ilimitada.

Boletins mensais, como o de setembro de 2018 [3], indicam que a geração média diária de energia elétrica por turbinas eólicas bateram recordes em diversas regiões, dentre as quais o Nordeste se destaca como a região com maior geração.

É também considerada uma forma de energia sustentável, devido à baixa ou nenhuma geração de dióxido de carbono e outros gases do efeito estufa, portanto é uma importante forma de obter energia sem agredir o meio ambiente. Atualmente, as fontes de energias sustentáveis são cada vez mais estudadas a medida que mais atenção é dada aos impactos ambientais causados pelo homem [4].

O melhor aproveitamento, desse tipo de energia, deve ser realizado em países que possuem grande potencial para geração de energia eólica. China, Estados Unidos e Alemanha dominam nesse setor, enquanto o Brasil aparece em $8^{\underline{o}}$ (oitavo) lugar [5], esse levantamento mostra a quantidade de energia produzida pelos países. Dentre esse e outros motivos, a energia eólica brasileira deve ser melhor aproveitada.

Figura 1.1: Parque eólico em Galinhos-RN [5]

Levantamentos recentes mostram que o Nordeste é a região com maior números de parques eólicos no Brasil [5], por contar com uma vasta região litorânea que promove ventos o ano inteiro. Na figura 1.1 pode-se ver um parque éolico na cidade de Galinhos, no Rio Grande do Norte. Aproximadamente 400 (quatrocentos) parques eólicos funcionam em todo o Nordeste, e cerca de 96 (noventa e seis) parques funcionam no Sul, São Paulo e Rio de Janeiro. Esse número só tende a crescer nos próximos anos.

1.1 Contextualização

Com o passar do tempo, mais estudos de viabilidade são realizados para a melhor utilização da energia eólica como fonte limpa de energia. Nota-se que as grandes potências econômicas mundiais estão investindo cada vez mais em uma fonte limpa e renovável, a fim de trazer melhores condições para seu país. E é neste intuito, que esse trabalho foi realizado. A realização de uma emulação de uma turbina eólica pode demonstrar o quão eficiente pode ser o vento na geração de energia.

A Universidade de Brasília dispõe de uma bancada de emulação de geração elétrica a partir de energia mecânica eólica. A bancada consiste de um motor de indução acoplado a um gerador síncrono de imã permanente através de um sistema de transmissão mecânico. Há também um inversor de frequência que consegue controlar o motor através da configuração de parâmetros.

1.2 Definição do Problema

Bibliotecas de comunicação que utilizam protocolo ModBus já existem e muitas estão disponíveis gratuitamente na internet. Esta biblioteca, em conjunto com o programa desenvolvido especificamente para esta aplicação, é responsável por transmitir os dados ao inversor. O programa fica então responsável pela parametrização do inversor de frequência que, por sua vez, controla o motor de maneira adequada.

Uma vez que a comunicação entre computador e inversor de frequência é estabelecida, deve-se executar o programa para que sejam enviados parâmetros necessários para o funcionamento do motor. O motor é acionado com os parâmetros desejados e deve alcançar velocidades de rotação semelhantes às velocidades alcançadas por motores em turbinas eólicas. A velocidade de rotação, neste trabalho, varia dentro do tempo de execução do programa, para que seja simulado um perfil de vento de uma região. Por meio dessa emulação, os dados obtidos podem ser analisados ao final.

1.3 Objetivo do trabalho

O objetivo deste trabalho é emular de forma mais próxima possível a geração de energia elétrica que uma turbina eólica é capaz. Tendo em mãos os estudos realizados sobre as condições climáticas, frequência de velocidade de vento e perfis de vento, a emulação pode ser feita e os resultados da mesma indicam o potencial e a adequabilidade do local que se planeja instalar.

Um objetivo paralelo do trabalho é fazer o controle do inversor de frequência de forma simples e amigável para o usuário. Após conectar o cabo *USB* do inversor no computador, a interface é responsável pela comunicação feita entre as duas máquinas e, através dela, é feito o controle de maneira simples e que o usuário desejar.

Capítulo 2

Referencial Teórico

2.1 Turbina Eólica

Os ventos se dão graças à diferença de temperatura do ar em diferentes regiões, o que causa uma diferença de pressão entre as massas de ar e, logo, o movimento da massa de ar ocorre. Turbinas eólicas, também chamadas de aerogeradores, aproveitam-se deste fenômeno natural para produzir energia elétrica. A Figura 2.1 mostra um desenho esquemático do modelo de turbina mais tradicional e utilizado em instalações de produção de energia elétrica de maior porte.

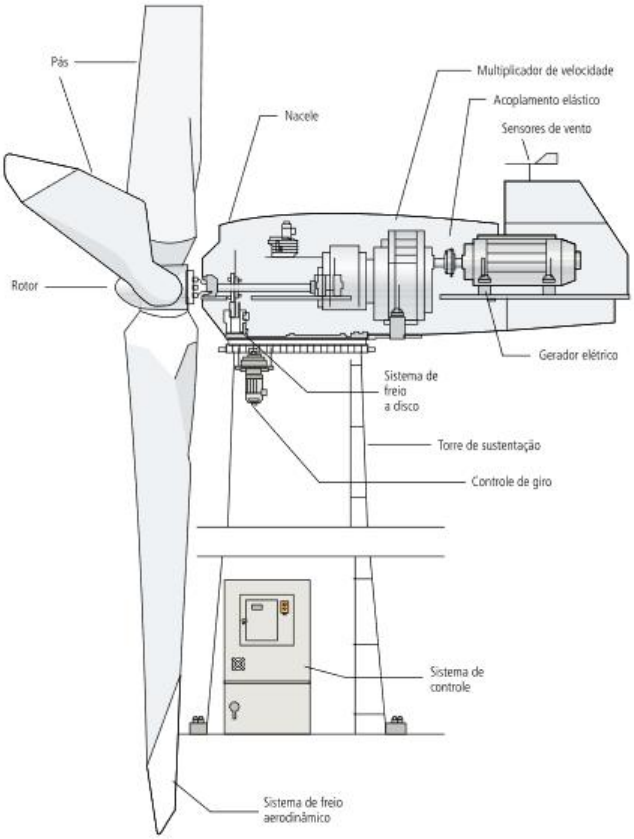


Figura 2.1: Desenho esquemático de uma turbina eólica [1].

Um maior interesse por alternativas limpas e renováveis para produção de energia elétrica irrompeu a partir dos anos 90, principalmente devido às necessidades da época. Houve, então, um grande impulso no uso do vento como forma de produzir energia elétrica, resultando num significativo desenvolvimento tecnológico de métodos e equipamentos utilizados na área [6].

Diversos fabricantes de aerogeradores surgiram e houve também um aumento na variedade de modelos. Dentre os diversos modelos existentes, pode-se classificá-los como turbinas de eixo vertical (Savonius, Giromill, Darrieus) e horizontal, sendo as turbinas de eixo horizontal as mais tradicionais especialmente em instalações de maior potência para produção de energia elétrica [6].

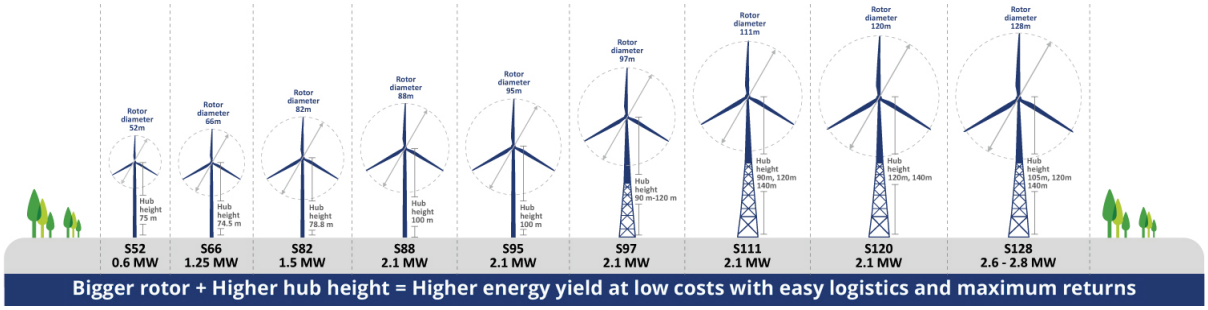


Figura 2.2: Evolução das turbinas eólicas da fabricante Suzlon [7].

A fim de otimizar a produção de energia elétrica em usinas eólicas, diversos cálculos precisos, decisões e adaptações devem ser levados em consideração: aerodinâmica ideal das pás, estudo probabilístico do vento e das condições climáticas da região, modelo de gerador a ser utilizado, controle de ângulo de passo, entre outros fatores. Porém, para este projeto, o escopo se limita a dois aspectos dados como importantes para o desenvolvimento da interface: energia eólica e velocidade de ponta.

2.1.1 Energia Eólica

A energia eólica é a energia cinética de uma massa de ar em movimento. Devido a sua característica aleatória, é preciso fazer um estudo do comportamento espacial e temporal do vento.

Como demonstrado em [6], a potência disponível pelo vento que passa por uma seção transversal de área A é dada por:

$$P = \frac{1}{2}\rho A v^3 \tag{2.1}$$

onde:

- P = potência do vento [W]
- ρ = massa específica do ar [kg/m^3]
- A = área da seção transversal [m^2]
- v = velocidade do vento [m/s]

É possível reescrever a Equação 2.1 de forma que ela passe a representar a potência que o sistema mecânico consegue aproveitar. Basta multiplicar a equação pelo coeficiente de eficiência C_p:

$$P_e = \frac{1}{2}\rho A v^3 C_p \tag{2.2}$$

É importante ressaltar que C_p, apesar de ser um coeficiente de eficiência, não possui valor máximo de 1. Este coeficiente tem valor máximo de $\frac{16}{27}$ ou aproximadamente 59.26%, chamado de Máximo de Betz, como demonstrado em [6]. Este valor abaixo do ideal se dá por ineficiências e perdas ocorridas durante a conversão de energia. Outros fatores como números de pás e projeto do aerogerador também influenciam no valor final do coeficiente. Na prática, é um valor que costuma flutuar próximo de 45% [8].

2.1.2 Velocidade de Ponta (*Tip Speed Ratio*)

Existem aerogeradores de velocidade fixa e de velocidade variável. Este projeto tem como foco a emulação de aerogeradores do segundo tipo, cuja velocidade de rotação das pás varia de acordo com a velocidade do vento. É importante demonstrar como é possível relacionar velocidade de rotação do rotor com a velocidade do vento com alguns poucos dados, que podem ser disponibilizados por fabricantes de diversos modelos de turbinas.

Primeiramente, temos a relação entre velocidade de ponta de pá e velocidade do vento dada por:

$$\lambda = \frac{\omega R}{V_{vento}} \tag{2.3}$$

onde:

- λ = velocidade de ponta (*Tip Speed Ratio*) [adimensional]
- ω = velocidade de rotação do rotor [rad/s]
- R = comprimento da pá [m]
- V_{vento} = velocidade do vento [m/s]

A Equação 2.3 é essencial para o cálculo da velocidade de rotação do rotor eólico sabendo apenas o comprimento da pá do modelo de aerogerador a ser estudado além da velocidade do vento. Para turbinas com velocidade variável, o λ se mantém fixo e a velocidade do rotor varia de acordo com a velocidade do vento, mantendo a máxima eficiência aerodinâmica. Este tipo de turbina tem vantagens como [6]:

- extração de energia do vento mais otimizada;

- pequena variação da potência em relação à potência nominal;
- cargas no rotor menores devido à ação das rajadas;
- baixa velocidade do rotor em caso de ventos de baixa velocidade, reduzindo a emissão de ruído no sistema;

A maioria das grandes turbinas de alta potência utilizadas são de velocidade variável, devido a estas vantagens.

É também possível calcular um TSR (*Tip Speed Ratio*) ótimo em função do número de pás n utilizadas pela turbina. Como demonstrando em [8], o TSR ótimo para extração máxima da potência contida no vento é dado por:

$$\lambda_{opt} \approx \frac{4\pi}{n} \tag{2.4}$$

2.2 Motor de Indução Trifásico

No meio industrial são encontrados diversos tipos de motores. Entretanto, por apresentarem características importantes como custo reduzido de compra e manutenção e uma vida útil mais longa em comparação aos demais [9], os motores de indução trifásico são mais utilizados.

Os motores trifásicos, como o próprio nome diz, são alimentados por um sistema trifásico. As tensões em cada fase, ou em cada fio, encontram-se defasadas em 120^{o}. Existem dois tipos de motores trifásicos, os de indução e os síncronos [9]. O motor utilizado no projeto é o W22 Plus da Weg, um motor de indução trifásico, que será melhor explicado ao longo dessa seção.

2.2.1 Estrutura do Motor de Indução

O motor de indução é constituído de duas partes principais: o estator e o rotor [9]. Entre o estator e o rotor existe um espaço denominado de entre ferro.

- **Estator:** composto pela carcaça, núcleo de chapas magnéticas e bobinas condutoras, o estator é a parte mais externa do motor. É no estator que a corrente da alimentação circula, proporcionando o movimento do rotor.
- **Rotor:** a parte móvel do motor. Basicamente um eixo móvel que também possui chapas magnéticas e bobinas condutoras, para sentir os efeitos do campo gerado pelo estator. A maneira como o rotor é construído define seu tipo, podendo ser: rotor gaiola de esquilo ou rotor bobinado [10].

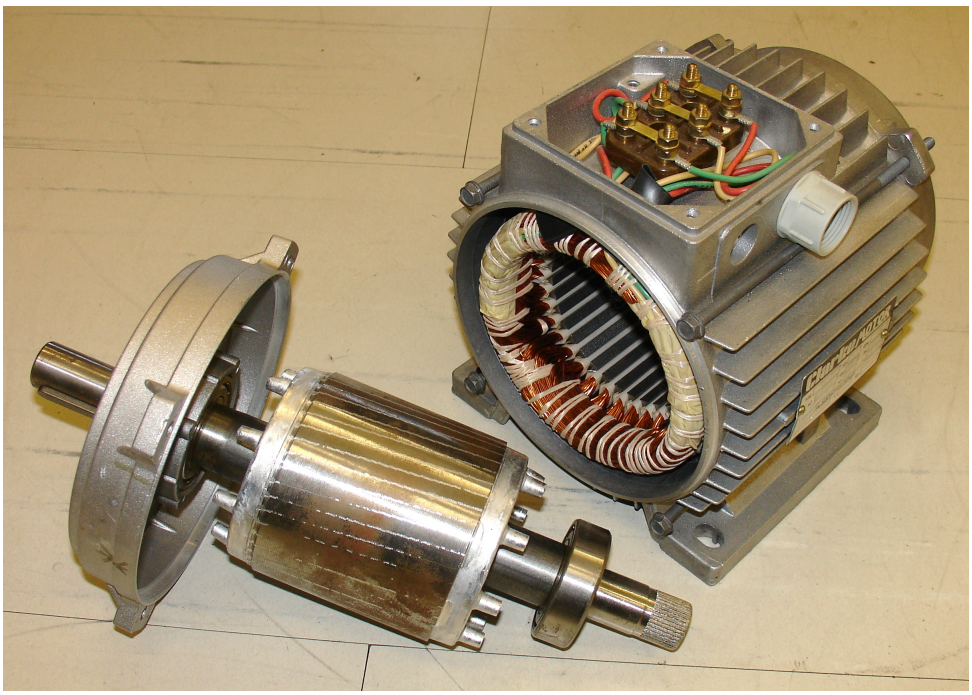


Figura 2.3: Partes de um motor de indução [11].

2.2.2 Princípios do funcionamento do motor de indução

As tensões aplicadas no estator geram correntes que circulam pelas suas bobinas. Estas correntes produzem um campo magnético que irá induzir uma tensão no rotor. Devido a essa indução, surge no rotor uma corrente indutiva. Esta corrente circula pelo material condutor do rotor gerando um fluxo magnético. A partir deste ponto temos a presença de um conjugado no rotor, acelerando-o em um sentido, horário ou anti-horário, dependendo do sentido do fluxo de corrente [10].

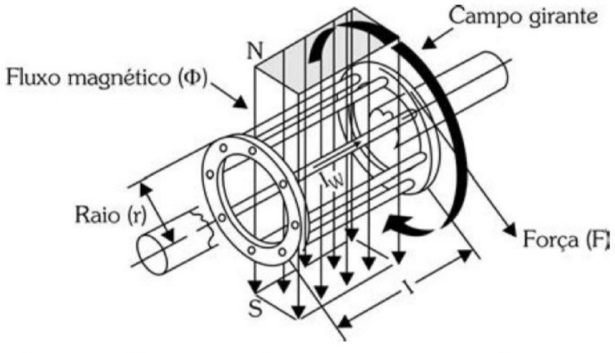


Figura 2.4: Indução de movimento no rotor [12].

É essa indução de corrente e tensão do estator no rotor que faz com que o motor de indução gire. Não necessariamente os dois, estator e rotor, precisam estar na mesma frequência. Se o rotor está bloqueado, este terá a mesma frequência do estator. Ao contrário, caso o rotor esteja livre,

girando na velocidade síncrona, o rotor tem sua frequência nula [10]. A frequência do rotor pode ser escrita pela Equação 2.5 encontrada em [10]:

$$f_{re} = s f_{se} \tag{2.5}$$

onde s é o escorregamento, ou a velocidade relativa percentual, e é definido pela Equação 2.6 também encontrada em [10]:

$$s = \frac{n_{sinc} - n_m}{f_{se}} 100\% \tag{2.6}$$

em que n_{sinc} é velocidade do campo magnético e n_m é velocidade mecânica do eixo do motor. Assim, a frequência fica limitada de acordo com o escorregamento. Caso o escorregamento seja nulo, o rotor se encontra em velocidade síncrona, e para o escorregamento igual a 1 (um), encontra-se na mesma frequência do estator.

2.3 Inversor de Frequência

O controle de velocidade no ambiente industrial é fundamental e indispensável para inúmeras aplicações. Não apenas pelo puro controle da velocidade para se realizar uma tarefa, como em indústrias de papel e celulose, mas também por motivos energéticos e de melhor eficiência, podendo contribuir para redução significativa de energia em certos casos [12]. Por esses e outros motivos, o uso de dispositivos para realização deste controle torna-se cada vez mais frequente e necessário dentro do cenário industrial.

Com o avanço e o surgimento de novas tecnologias, esses dispositivos de controle de velocidade que surgiram como dispositivos mecânicos, hoje em dia encontram-se permeados de circuitos eletrônicos e sistemas embarcados. Um dispositivo que possui tais tecnologias e que será utilizado neste trabalho é o inversor de frequência. Os inversores de frequência são capazes de realizar diferentes tipos de controle apenas alterando suas próprias configurações. Por serem versáteis e dinâmicos os inversores de frequência atendem a diversos requisitos, sendo bastante utilizados no meio industrial [13].

2.3.1 Princípios de funcionamento

Na Figura 2.2 pode-se visualizar uma representação em blocos dos componentes dos inversores de frequência. Seguido da explicação de cada parte, de acordo com [12]:

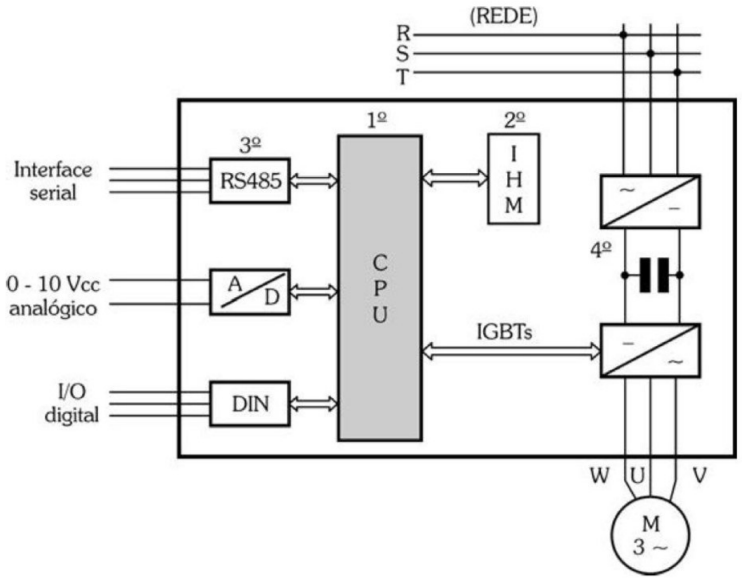


Figura 2.5: Diagrama simplificado dos principais blocos do inversor de frequência [12].

1. **Unidade Central de Processamento (*Central Processing Unit* - - CPU):** Formada por um microprocessador ou por um microcontrolador. É nesse bloco que todas as informações (parâmetros e dados do sistema) estão armazenadas. A CPU também é responsável pela geração dos pulsos de disparo, por meio de uma lógica de controle coerente, para os Transistores Bipolares de Porta Isolada (*Insulated Gate Bipolar Transistor* - IGBT).

2. **Interface Homem/Máquina (IHM):** Responsável por mostrar o que está ocorrendo no inversor e parametrizá-lo de acordo com a aplicação.

3. **Interfaces:** A maioria dos inversores pode ser comandada por dois tipos de sinais: analógicos ou digitais. A interface analógica é a mais utilizada. Porém, além da interface analógica, o inversor possui entradas digitais que podem ser usadas por meio da programação de um parâmetro.

4. **Etapa de potência:** A etapa de potência é constituída por um circuito retificador que por meio de um circuito intermediário denominado 'barramento CC' alimenta o circuito de saída do inversor (módulo IGBT).

2.3.2 CFW-11

O laboratório conta com o inversor de frequência CFW-11 da Weg, responsável por realizar o controle de velocidade do motor e a comunicação com o computador, além de se comunicar com o motor. Escolhido por apresentar excelente desempenho estático e dinâmico, controle preciso de torque e velocidade, e qualidade e economia de energia elétrica [13], o CFW-11 atende à demanda do projeto, realizando o controle do motor W22 Plus, também da Weg. O CFW-11 possui uma vasta lista de parâmetros, que vai do P0000 ao P1052. Alguns desses parâmetros podem ser programados

enquanto outros apenas lidos pelo usuário. A programação e a escolha dos parâmetros muda de acordo com a necessidade do projeto.

2.3.3 Parâmetros

A parametrização do CFW-11 pode ocorrer através da IHM ou por outra interface [14] sendo grande parte *Universal Serial Bus* (USB) ou Serial. Os inúmeros parâmetros disponíveis pelo CFW-11 são separados em grupos para facilitar a sua procura. Podendo acessar um por um, ou recorrer ao grupo no qual o parâmetro pertence. Todavia, não serão destacados os dez grupos criados pelo inversor, mas sim os tipos de parâmetros que podem ser encontrados de acordo com suas características e particularidades [12]:

- Parâmetros de leitura
- Parâmetros de regulação
- Parâmetros de configuração
- Parâmetros do motor

A devida explicação de cada tipo poderá ser encontrada nas subseções seguintes, com exemplos de parâmetros do CFW-11.

2.3.3.1 Parâmetros de leitura

São aqueles que podem ser apenas visualizados pelo usuário, mas não podem ser programados. No manual de programação do CFW-11, estes aparecem com o símbolo RO ao lado. Exemplos de parâmetros de leitura observados durante o projeto são os P0002, P0003 e o P0005:

- **P0002:** Velocidade do motor
- **P0003:** Corrente do motor
- **P0005:** Frequência do motor

2.3.3.2 Parâmetros de regulação

Estes são os programáveis, os quais serão constantemente modificados pela interface para que possa ser realizada a emulação de vento. Exemplos de parâmetros de regulação utilizados durante o projeto são os P0100, P0101, P0133 e P0134:

- **P0100 e P0101:** Tempo de aceleração e tempo de desaceleração, respectivamente.
- **P0133 e P0134:** Limite de referência de velocidade mínima e limite de referência de velocidade máxima, respectivamente.

2.3.3.3 Parâmetros de configuração

Responsáveis pela definição das características do inversor, as funções a serem executadas, por exemplo, parâmetros dos relés de saída e das entradas do inversor. Exemplos de parâmetros de configuração utilizados durante o projeto são os P0202 e do P0220 ao P0228:

- **P0202:** Tipo de controle
- **P0220:** Seleção LOCAL/REMOTO

2.3.3.4 Parâmetros do motor

Armazenam as caracterísitcas nominais do motor. Exemplos de parâmetros do motor utilizados durante o projeto são os P0400 ao P0407:

- **P0400:** Tensão nominal motor
- **P0401:** Corrente nominal motor
- **P0402:** Rotação nominal motor
- **P0403:** Frequência nominal motor

2.3.4 Controle Escalar

O controle escalar é o modo mais simples de controle por tensão/frequência (V/f) imposta. Suas principais características são a regulação de velocidade em malha aberta ou com compensação de escorregamento, e tornar possível a operação multimotor [14]. Nesta aplicação o motor trabalha com fluxo aproximadamente constante [12]. Desta maneira o inversor segue como uma fonte de tensão que gera valores de frequência e tensão de acordo com o modo que foi programado [14].

A necessidade de poucos ajustes nos parâmetros do CFW-11 torna o controle escalar uma maneira simples e, de certa forma, eficaz para o projeto. Outro tipo de controle, como por exemplo o vetorial, também pode ser implementado para melhores resultados.

Para utilizar do controle escalar no CFW-11 os parâmetros de P0137 ao P0141 devem ser programados pelo usuário da maneira que melhor se encaixe no projeto. E o parâmetro P0202 deve estar em 0 (V/f 60Hz) para a aplicação do controle escalar no motor W22 Plus.

A Figura 2.6 mostra um diagrama de blocos para o controle escalar de um inversor de frequência. Enquanto a Figura 2.7 explica o blocodiagrama do controle V/f para o CFW-11, mostrando os parâmetros responsáveis pelo V/f.

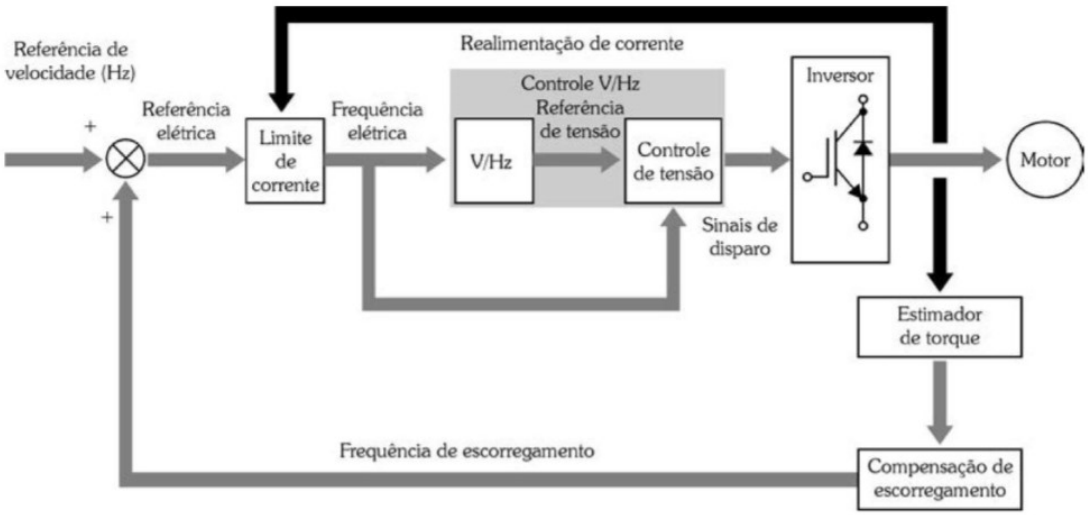


Figura 2.6: Diagrama de blocos para controle escalar [12].

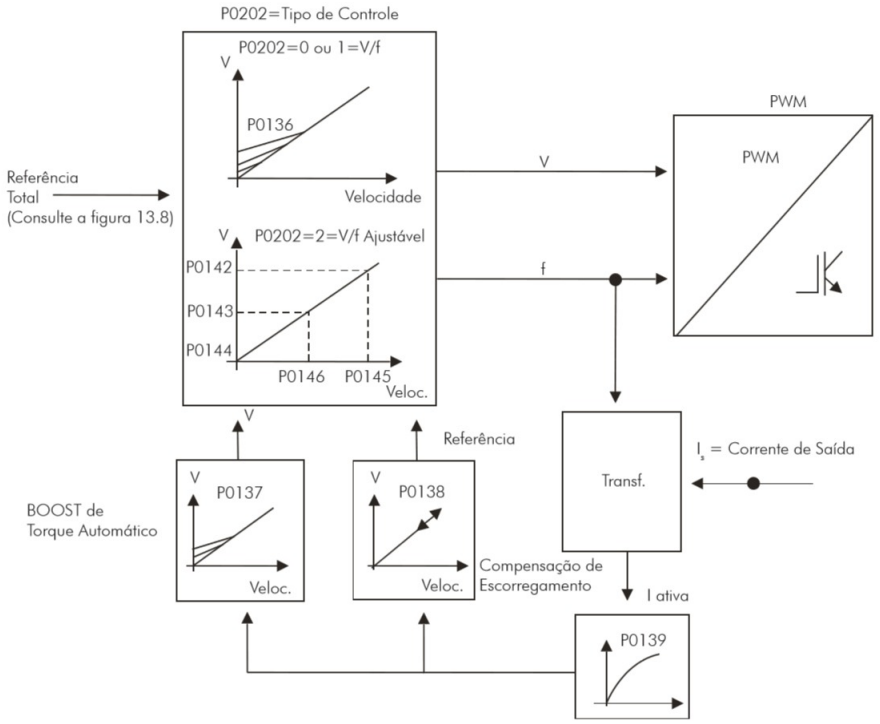


Figura 2.7: Blocodiagrama controle V/f [14].

2.4 Comunicação Serial

Este método de comunicação, como o próprio nome diz, utiliza do envio sequencial de dados bit a bit. Diferente da comunicação paralela, na qual os dados são enviados simultaneamente. Todavia, os dados, em ambos os tipos, são transferidos de um dispositivo ao outro por meio de um canal de comunicação ou de um barramento [15].

De fundamental importância conhecer os fatores da comunicação que serão implementadas ao longo do projeto nos parâmetros P0310 e P0311 do CFW-11. São eles:

- **Taxa de transmissão:** É uma referência ao número de bits por segundo que é transmitido pelo barramento.
- **Bits de dados:** A quantidade de bits reais de uma mensagem.
- **Paridade:** Este fator está ligado diretamente a checagem de erro, ele verifica se a quantidade de números 1's no byte é par ou ímpar, ou nenhum dos dois, caso a paridade seja nula.
- **Bit de parada:** Responsável por indicar o final de uma mensagem. Entretanto iremos ver mais para frente como funciona a comunicação no CFW-11.

Na comunicação serial pode-se notar três tipos de conexões: simplex, half duplex (HDX) e full duplex (FDX). A conexão half duplex permite que a troca de dados possa ocorrer nas duas direções, porém não simultaneamente [16]. Este é o tipo de conexão utilizado no projeto.

Apenas a conexão serial entre dois dispositivos não garante a comunicação entre eles. Alguns padrões devem ser seguidos para que eles se entendam. Por isso, uma série de protocolos de comunicação foram criados para que diferentes dispostivos eletrônicos possam, de certa maneira, conversar entre si. Modbus, Profibus e TCP são exemplos de protocolos desenvolvidos para que ocorra a comunicação. Cada protocolo possui sua particularidade na troca de dados entre os dispositivos, e cada dispositvo deve estar a par do protocolo que está sendo utilizado.

Além disso, existem normas que definem os esquemas de transmissão de dados. Normas como a RS485 e a RS422. Estas normas não são vinculadas a um único tipo de protocolo, podendo ser utilizadas em diferentes protocolos [17].

2.4.1 Protocolo Modbus

Escolhido por ser um protocolo aberto e de bastante uso no cenário industrial, o protocolo Modbus surge em 1979 desenvolvido pela Modicon [18], uma empresa do grupo *Schneider Electric*. O protocolo surge como uma comunicação entre um mestre (*master*), dispositivo que comanda as ações, e um escravo (*slave*), dispositivo que as executa. No protocolo Modbus um mestre pode ter até 247 (duzentos e quarenta e sete) escravos, cada um tendo seu próprio endereço de 1 a 247.

Segundo [18], o protocolo Modbus é uma camada de aplicação de mensagens para mestre e escravos entre dispositivos conectados em diferentes tipos de barramentos ou rede. A Figura 2.8 ilustra tal premissa.

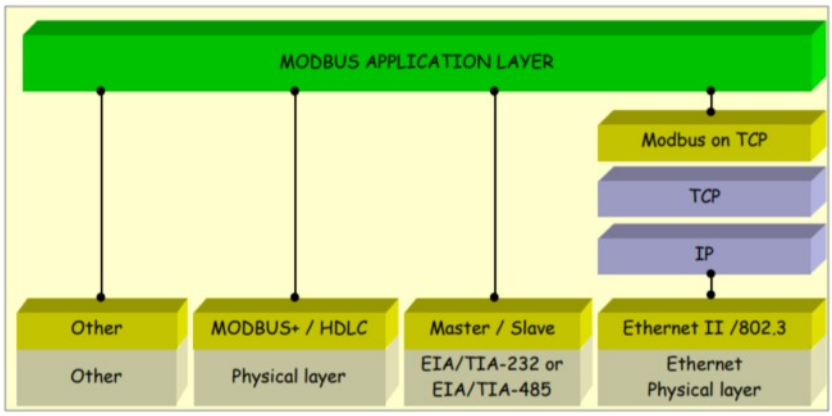


Figura 2.8: Comunicação Modbus [18].

O protocolo Modbus, enquanto modo de transmissão, pode ser divido em dois tipos: Modbus ASCII e Modbus RTU. Cada modo possui sua própria forma de transmissão dos bytes da mensagem, não sendo possível a utilização dos dois modos na mesma rede. O inversor CFW-11 utiliza somente o modo RTU [15].

2.4.1.1 Estrutura da mensagem

No Modbus RTU a transmissão dos dados da mensagem é feita de tal maneira que cada byte, ou sequência de bytes, armazena um dado ou função. Nessa sequência de bytes são encontradas informações como: o endereço do escravo, a função que o mestre está requisitando, os dados da requisição e o CRC. Essa estrutura é válida tanto para o mestre, quanto para o escravo.

Tabela 2.1: Estrutura das mensagens Modbus RTU [15].

Endereço	Função	Dados da requisição	CRC
1 byte	1 byte	n bytes	2 bytes

- **Endereço:** Como mencionado anteriormente, cada escravo possui seu endereço próprio. Dessa maneira o mestre envia no primeiro byte da mensagem o endereço ao qual a mensagem se destina. O escravo, com aquele endereço, então responde ao mestre a requisição.

  O mestre também pode enviar uma mensagem a todos os escravos da rede, basta destinar a mensagem ao endereço 0 (zero). Desta maneira, nenhum escravo irá responder

- **Função:** Contém um único byte que especifica o serviço que o escrevo deverá executar. No modo Modbus RTU existem nove possíveis funções que podem ser implementadas. Ao longo do projeto serão utilizadas as funções *Read Holding Registers* e *Write Single Register*, cujos códigos das funções são 03 (três) e 06 (seis), respectivamente.

- **Dados:** O tamanho do campo varia de acordo com a função exigida. Endereço de registradores e quantidade de registradores que irão realizar uma certa tarefa, são dados que o

mestre deve passar ao escravo. Enquanto o escrevo devolve ao mestre, caso seja uma função de leitura, os dados que estão nos registradores requisitados.

- **CRC:** Por último tem-se o CRC *(Cycling Redundancy Check)*. Este campo é formado por dois bytes, o CRC- (byte menos significativo) e o CRC+ (byte mais significativo). Este campo é calculado dentro da comunicação e serve para verificar se a mensagem foi entendida por ambas as partes.

2.4.1.2 Tempo entre as mensagens

No modo RTU não existe um caractere específico que indique o início ou o fim de um telegrama [15]. O tempo que define a leitura da mensagem.

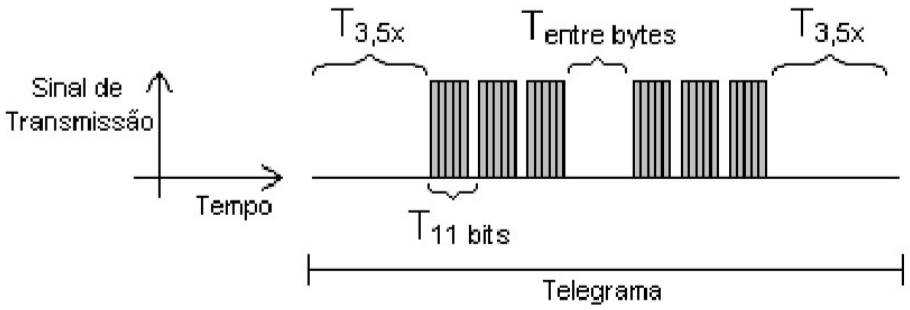


Figura 2.9: Faixas de tempo para o modo RTU [15].

Tabela 2.2: Faixas de tempos da mensagem de acordo com a taxa de comunicação [15].

Taxa de comunicação	T_{11bits}	$T_{3,5x}$
9600 bits/s	1,146 ms	4,010 ms
19200 bit/s	573 μs	2,005 μs
38400 bit/s	573 μs	2,005 μs
57600 bit/s	573 μs	2,005 μs

A Figura 2.9 e a Tabela 2.2, retiradas do manual de comunicação serial da WEG [15], apresentam as faixas de tempo necessárias para a compreensão do funcionamento da comunicação do modo RTU, onde:

$T_{3,5x}$ = Intervalo mínimo para indicar começo e fim de um telegrama

$T_{entrebytes}$ = Tempo entre bytes (não pode ser maior que $T_{3,5x}$)

T_{11bits} = Tempo para transmitir uma palvra do telegrama

Um novo telegrama se inicia, após decorrer um tempo de no mínimo $T_{3,5x}$. Passado esse tempo, qualquer caractere que chegar indicará o início de um novo telegrama. E o final deste telegrama

acontecerá quando passado esse mesmo tempo de $T_{3,5x}$ não existir nenhum novo caracter. Desta maneira, falha de sincronismo entre os dispositivos pode ocasionar uma leitura inválida.

2.4.1.3 RS-485

A norma RS485 descreve uma interface de comunicação operando em linhas diferencias [17], em outras palavras, a RS485 define o comportamento elétrico entre o mestre e escravo. Esse meio de comunicação, geralmente, é realizado por um par trançado. O qual transmite e recebe dados. Essa transmissão de dados é reconhecida pela diferença de tensão aplicada em cada condutor do par trançado. Os condutores são diferenciados em '+' e '-', desse modo, caso a tensão esteja maior em '+' e menor em '-', temos um certo nível de tensão positiva. Caso contrário, se a tensão em '+' for menor que em '-', temos um nível de tensão negativa. Essa tensão positiva e negativa é caracterizada pelos níveis lógicos 1 e 0, respectivamente [17].

Por se tratar de um diferencial, a norma RS485 determina um comprimento máximo de 1200 metros para os cabos de comunicação [17]. Como os dados passam de maneira sequencial pelo cabo, quanto maior o cabo de comunicação, menor é a velocidade da comunicação.

Para que o sinal emitido pelo computador chegue ao inversor de maneira entendível, um adaptador USB para RS-485 é conectado ao computador por meio de um cabo USB. Este adaptador está conectado ao módulo RS-485 do CFW-11 por meio de um cabo blindado de terra comum e configuração DB-9. Essa operação torna possível a comunicação do computador com o CFW-11. A Figura 2.10 mostra o adaptador USB para RS-485 utilizado.

Figura 2.10: Adaptador USB para RS-485 com os devidos cabos.

2.4.2 Python 3.7

Uma das linguagens de programação mais utilizadas, além de ser livre, de código aberto e repleta de documentação na internet, o Python foi a ferramenta utilizada para se criar a interface gráfica. Uma linguagem bastante intuitiva e que possui funções similares às linguagens C/C++, Java e outras.

O código gerado para criação da interface e comunicação serial entre o computador e o CFW-11 foi estruturado todo em Python. Para isso algumas bibliotecas foram importadas. A Tkinter, uma biblioteca padrão da linguagem Python, foi requisitada no processo de criação da interface; e a biblioteca 'minimalmodbus', criada por Jonas Berg [19], proporciona o uso das funções de comunicação vistas na Seção 2.4.1 deste relatório.

2.4.2.1 Tkinter

Presente na instalação padrão do Python, a Tkinter permite desenvolver interfaces gráficas utilizando a linguagem Python. Os diversos recursos fornecidos pela biblioteca e a facilidade para escrever códigos, foram fundamentais para a escolha da Tkinter como ferramente para criação de interface gráfica. A documentação da biblioteca, ensinando como usá-la no código e demais funções podem ser encontradas em [20].

2.4.2.2 Minimalmodbus

Depois de diversas tentativas com outras bibliotecas em Python, ou em outra linguagem, a minimalmodbus trouxe resultados satisfatórios quando utilizada. A sua documentação na internet e a quantidade de comentários úteis no programa, facilita a identificação das funções e na utilização destas.

Assim como no CFW-11, os parâmetros de taxa de comunicação, paridade e bits de parada, devem ser inicializados para estabelecer a comunicação dos dispositivos. Uma vez criado esse meio, funções como *write register* e *read register* podem ser executadas para requisição de dados do CFW-11.

Capítulo 3

Composição e Especificações da Bancada

Serão apresentados os componentes principais da bancada experimental utilizada neste trabalho. Estes componentes já foram especificados anteriormente em [21] e em [22].

3.1 Motor de Indução Trifásico (MIT)

O motor que emula o rotor de um aerogerador foi fabricado pela WEG. O mesmo esteve ligado em formato delta. Suas especificações podem ser encontradas na Tabela (3.1) abaixo:

Tabela 3.1: Especificações do MIT [22].

Motor de Indução Trifásico	
Marca	WEG
Linha do produto	2258/M
Frequência	60Hz
Número de Pólos	8
Potência	30 cv (22kW)
Rotação Nominal	885RPM
Escorregamento	1.67%
Tensão Nominal	380/660 V
Corrente Nominal	45/25,9 A
Corrente à vazio	20,8/12 A
Conjugado Nominal	238 N.m
Momento de Inércia	0,84722 Kg.m^2
Carga	50% / 75% / 100%
Fator de Potência	0,65 / 0,76 / 0,81
Rendimento	91,7 / 91,9 / 91,7 %

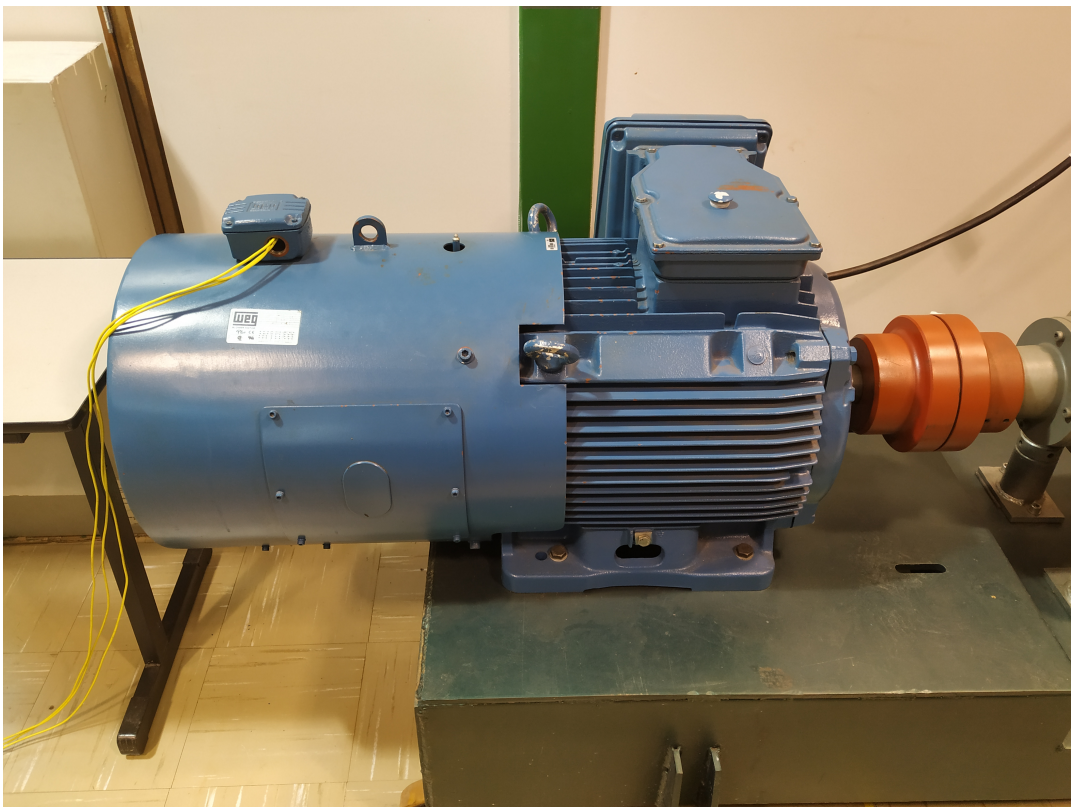


Figura 3.1: MIT com motor de ventilação acoplado.

3.1.1 Motor de Ventilação

Junto ao MIT, está acoplado um motor de ventilação. O mesmo esteve ligado no formato estrela. Especificações na Tabela (3.2) abaixo:

Tabela 3.2: Especificações do Motor de Ventilação.

Motor de Ventilação	
Marca	WEG
Linha do produto	Blower Motor
Frequência	60Hz
Potência	1 cv (0,75kW)
Rotação Nominal	1730 RPM
Fator de Serviço	1.15
Tensão Nominal	220 / 380 / 440 V
Corrente Nominal	2,98 / 1,73 / 1,49 A
Rendimento	82,6%

3.2 Gerador Síncrono de Imãs Permanentes

O gerador usado na bancada possui referências de utilização em centrais eólicas industriais. É acoplado diretamente ao eixo do motor e não necessita de nenhuma caixa de engrenagem no sistema. A Tabela (3.3) contém os dados do gerador em questão:

Tabela 3.3: Especificações do PMSG (*Permanent Magnet Synchronous Generator*) [22].

Gerador Síncrono de Imãs Permanentes	
Marca	Alxion
Modelo	400TK2M
Velocidade Nominal	800 RPM
Potência Nominal	17874 W
Torque Nominal de Entrada	234 N.m%
Rendimento	92%
Corrente Nominal	42 A
Número de Pares de Pólos	12
Resistência por fase	0,15 Ω
Indutância por fase	1,24 mH
Tensão a vazio	305 V
Momento de Inércia	163 g.m^2
Peso	35 Kg
Área de seção do cabo	4x10 mm^2
Diâmetro do cabo	16,7 mm

Figura 3.2: PMSG, ainda não acoplado no eixo do motor [22].

3.3 Caixas Multiplicadoras de Velocidade

Apesar de não haver necessidade de caixa multiplicadora para acoplar o MIT com o PMSG , a bancada possui duas caixas multiplicadoras ligadas em série. A implementação desta caixa se dá para realização de pesquisas neste tipo de equipamento [22] e portanto não tem maiores influências no projeto. Na Figura (3.3), a caixa da esquerda tem relação 36:1 e a caixa da direita tem relação 1:27.

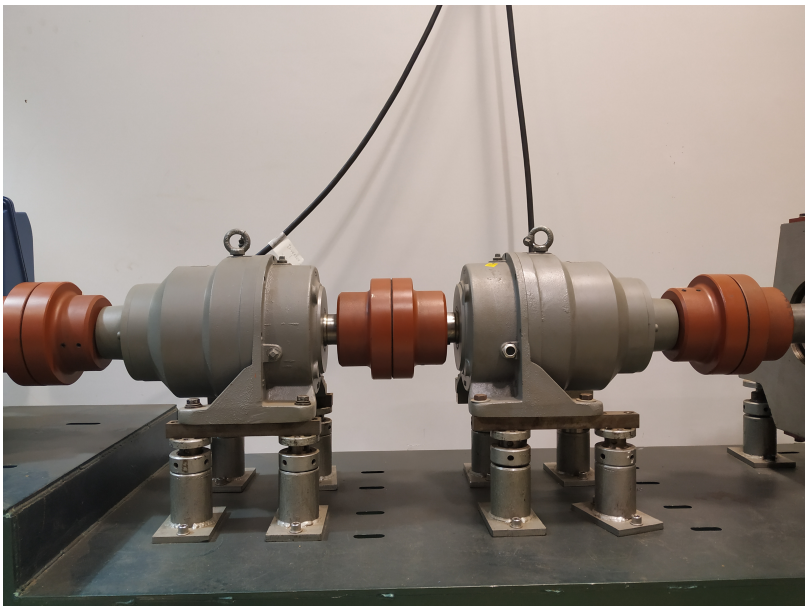


Figura 3.3: A caixa da esquerda tem relação 36:1 e a caixa da direita tem relação 1:27.

Figura 3.4: Foto do conjunto MIT (com motor de ventilação acoplado) com PMSG acoplado utilizando as duas caixas multiplicadoras.

3.4 Inversor de Frequência

O modelo do inversor de frequência utilizado no projeto é o CFW11 modelo 0058 T405Z, que também foi fabricado pela WEG. Ele é responsável por realizar a excitação e o controle do MIT. Informações sobre o inversor se encontram na Tabela (3.4).

Tabela 3.4: Especificações do Inversor [21].

Inversor de Frequência	
Marca	WEG
Modelo	CFW 11 0058 T4054Z
Frequência de Alimentação	50 / 60 Hz
Tensão de Alimentação	380 / 440 V
Tensão de Saída	380 V
Rendimento Típico	Maior ou Igual a 97%
Fator de Potência Típico	0,94

Figura 3.5: Inversor de frequência ligado e pronto para execução.

3.5 Módulo RS-485

É utilizado o módulo de comunicação RS485 que possui um conector plug-in de 4 vias (XC7). Abaixo, a Tabela (3.5) possui a pinagem listada com seus nomes e respectivas funções:

Tabela 3.5: Pinagem do conector de 4 vias para RS485 [23].

Pino	Nome	Função
1	A-Line(-)	RxD/TxD negativo
2	B-Line(+)	RxD/TxD positivo
3	GND	0 V isolado do circuito RS485
4	Ground	Terra (blindagem)

3.6 Adaptador Serial-USB

É necessário um adaptador serial DB9 para USB para conectar o modulo RS485 ao computador. O modelo utilizado é mostrado na Figura (3.1). Abaixo, a Tabela (3.6)

Figura 3.6: Adaptador Serial/USB [24].

Tabela 3.6: Pinagem do conector DB9 para o Adaptador Serial-USB. [24]

Pino	Nome
1	D(-)
2	D(+)
3	-
4	-
5	Ground

Para configurar o adaptador para utilizá-lo em uma comunicação de dois fios é preciso posicionar as chaves seletoras de forma onde a chave número 1 está ligada e a chave número 2 está desligada.

Capítulo 4

Metodologia

Este capítulo descreve detalhadamente as metodologias utilizadas no trabalho.

4.1 Biblioteca de Comunicação

Há diversas bibliotecas de comunicação serial que implementam o protocolo Modbus disponíveis gratuitamente e que podem ser facilmente encontradas na internet. A Modbus Organization, um grupo de usuários independentes e fornecedores de dispositivos de automação que apoiam a evolução e distribuição do protocolo nos diversos segmentos do mercado de automação, disponibiliza uma lista de bibliotecas pagas e gratuitas.

A princípio, o critério para escolha da biblioteca foram: linguagem de programação familiar, como por exemplo C, C++, Java e Python; E documentação bem detalhada. Testes foram feitos utilizando algumas bibliotecas dentre as opções indicadas pela Modbus Organization, porém nenhuma delas foi adequada para aplicação em questão por falta de documentação detalhada. A descoberta da biblioteca utilizada para comunicação da interface com o inversor de frequência foi dada ao procurar por um código feito em Python, uma das linguagens mais utilizadas atualmente por ser versátil e de fácil uso.

Por fim, a escolha pela MinimalModbus [19] foi feita por ser um módulo de uso simplificado, *open-source* e documentação detalhada. O módulo oferece diversas funções básicas e extremamente essenciais, principalmente para o objetivo do trabalho, tais como escrita e leitura de registradores.

4.2 Programação do Código

Esta seção é dividida em duas partes: uma descrevendo as diversas funções disponíveis na interface gráfica e outra que descreve como o programa interpreta as entradas feitas pelo usuário e as envia para o inversor.

4.2.1 Interface Gráfica do Usuário

Apesar de ser possível configurar todos os parâmetros importantes para o funcionamento do motor através da IHM, ela é totalmente limitada e o trabalho se torna cansativo e lento. Então a criação de uma GUI (*Graphical User Interface*) se mostrou essencial para que o usuário pudesse configurar o inversor de frequência de maneira intuitiva e rápida. Utilizando a biblioteca Tkinter, nativa do Python, foi feita uma interface que dispõe de funções importantes para uma emulação da forma que o usuário desejar. A Figura 4.1 mostra a GUI com suas funções destacadas e inumeradas em vermelho.

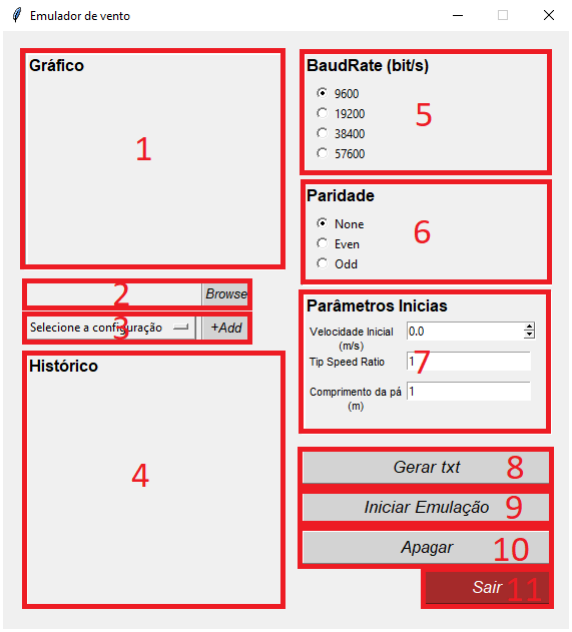


Figura 4.1: GUI com as funções destacadas e inumeradas em vermelho.

1. **Gráfico:** Ainda não implementada.

2. ***Browser*:** Abre o navegador de arquivos para que o usuário possa carregar um arquivo '.txt' com configurações previamente definidas.

3. **Selecionar/Adicionar configuração:** Seleciona um dos tipos dos ventos a ser adicionado na configuração. Uma nova janela será aberta ao selecionar o tipo de vento para configurar os parâmetros para aquele tipo de vento.

4. **Histórico:** Mostra ao usuário os tipos de vento selecionados anteriormente e seus respectivos parâmetros.

5. ***BaudRate*:** Seleção do *BaudRate* (taxa de transmissão) utilizada na comunicação.

6. **Paridade:** Seleção da Paridade da comunicação.

7. **Parâmetros Iniciais:** Configurar parâmetros iniciais. 'Tip Speed Ratio' e 'Comprimento de pá' devem ser diferentes de zero.

8. **Gerar arquivo '.txt':** Gera arquivo texto com as configurações e parâmetros que o usuário cadastrou durante a execução da GUI.

9. **Iniciar emulação:** Envia os parâmetros necessários para o inversor, a fim de começar a emulação.

10. **Apagar:** Limpa as configurações e parâmetros que o usuário cadastrou anteriormente.

11. **Sair:** Encerra a GUI.

4.2.2 Software

O funcionamento do software pode ser facilmente descrito em poucos detalhes. Ao executar o código Python, a GUI mostrado na Figura 4.1 irá abrir e o usuário poderá fazer qualquer uma das funções enumeradas anteriormente. Esta seção irá detalhar partes importantes do software.

4.2.2.1 Tipos de Ventos e Parâmetros

Quando o usuário inicia a GUI, ele pode começar a cadastrar uma configuração. Estão disponíveis quatro tipos de ventos diferentes, cada uma com parâmetros individuais, e mais outros parâmetros. Os quatros tipos são:

1. **Rampa**: Vento de aceleração constante. Este tipo recebe um valor para velocidade final atingida e um valor para duração da rampa.

2. **Rajada**: Vento repentino de alta aceleração e duração relativamente baixa. A velocidade final do vento será igual a velocidade inicial. Este tipo recebe um valor para velocidade máxima atingida e um valor para duração da rajada.

3. **Brisa**: Vento leve de comportamento aleatório e de baixa variação. Este tipo recebe apenas um valor para duração da brisa.

4. ***Ripple***: Vento cuja velocidade se comporta como uma onda. Este tipo recebe um valor para amplitude de onda da velocidade e para a duração da Ripple.

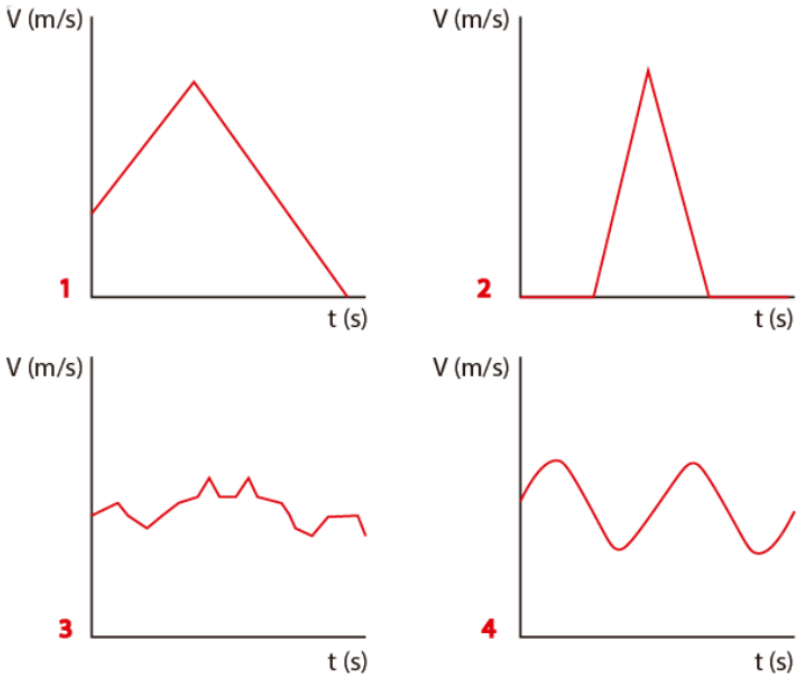


Figura 4.2: Demonstração dos tipos de vento realizados pelo software

É possível cadastrar uma configuração com qualquer combinação de tipos de ventos, onde cada um pode receber parâmetros diferentes. A emulação irá executar a configuração na ordem em que os cadastros foram feitos. Lembrando que as velocidades que deverão ser entradas são todas em [m/s] e os tempos em [s]. Qualquer outra variável que não apresentar especificações sobre a unidade de medida são adimensionais.

A taxa de transmissão e a paridade devem ser configurados de forma em que coincidam com os valores configurados no inversor, para que a comunicação ocorra da forma correta.

Por último, os parâmetros iniciais são: velocidade inicial do vento (em [m/s]), *Tip Speed Ratio* e comprimento da pá (em [m]). A velocidade inicial pode receber valores maiores ou iguais a zero, enquanto os valores de TSR e comprimento de pá devem receber um número maior do que zero. O motivo para essa restrição se dá ao fato de que estes dois parâmetros são essenciais para a conversão da velocidade do vento em velocidade de rotação em [rpm].

4.2.2.2 Troca de Mensagens com Inversor

O ínicio da emulação dará início a comunicação serial utilizando o protocolo Modbus. Lembrando que é necessário já estar com o cabo USB conectado antes mesmo do início da emulação para que a comunicação ocorra sem erros. São utilizados os comandos *instrument.write.register* e *instrument.read.register* para escrita e leitura em registradores, respectivamente. Como cada registrador do CFW11 representa um parâmetro do inversor, é necessário apenas fazer a escrita nos parâmetro para controlar a velocidade do MIT e fazer a leitura nos parâmetros de velocidade de referência.

Os parâmetros alterados para alteração da velocidade são:

- P0001: Parâmetro para leitura de velocidade de referência.
- P0100 e P0101: Tempo de aceleração e deceleração, respectivamente.
- P0133 e P0134: Limite de referência de velocidade mínima e máximam, respectivamente.
- P0682: Palavra de controle via Serial/USB.
- P0683: Referência de Velocidade via Serial/USB.

O parâmetro $P0682$ é utilizado para habilitar o inversor e, logo, a operação do motor. Usa-se valor '13' para colocar o inversor em estado *Ready* e valor '14' para colocá-lo em estado *Run*. Os parâmetros $P0133$ e $P0134$ são utilizados para limitar as velocidades mínimas e máximas e também são usados para acelerar ou desacelerar o motor em conjunto com os parâmetros $P0100$ e $P0101$. Por último, o parâmetro $P0683$ é utilizado para mudar a velocidade do motor.

Como $P0100$ representa o tempo em que o motor deve levar para acelerar de 0 a velocidade máxima, é preciso fazer um pequeno cálculo para saber o valor deste parâmetro para que o motor acelere de uma velocidade inicial V_o para uma velocidade final V_f em um intervalo de tempo t. Este cálculo é feito utilizando as Equações 4.1 e 4.2.

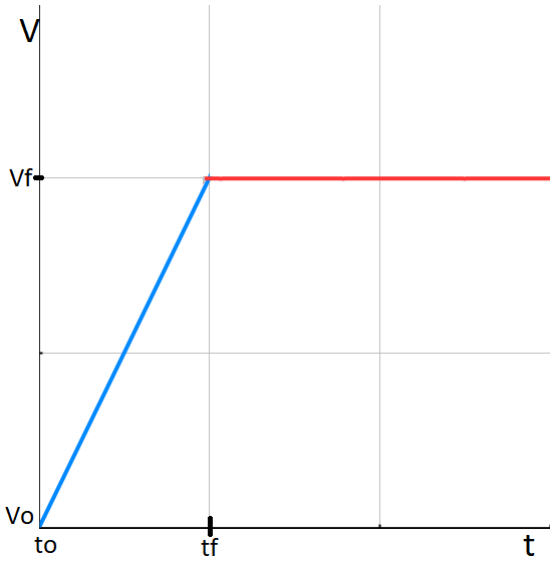


Figura 4.3: Representação gráfica de uma aceleração do tipo rampa, com duração $t = t_f - tf$

$$P0100 = \frac{tV_f}{V_f - V_o} \tag{4.1}$$

$$P0101 = \frac{tV_f}{V_o - V_f} \tag{4.2}$$

Também é utilizada a Equação 2.3 para calcular a velocidade de rotação que o motor deverá receber que corresponderá a velocidade do vento adequada. É importante notar que a velocidade de rotação depende dos parâmetros iniciais citados anteriormente (TSR e comprimento da pá da turbina em [m]), pois estes são utilizados neste cálculo.

*P*0683 recebe a referência da velocidade, a fim de mudar a velocidade de rotação. Este parâmetro deve receber o valor de rotação multiplicado pela fração $\frac{8045}{885}$. A fração foi encontrada por simples regra de três, como pode ser visto em [15], já que o parâmetro recebe uma referência de 13 bits de sinal para representar a rotação síncrona do motor.

Capítulo 5

Resultados e Discussões

Este capítulo apresenta os resultados obtidos no trabalho e discute alguns erros encontrados durante a execução dos códigos.

5.1 Execução do software

A fim de apresentar os resultados neste relatório, o código foi executado no sistema operacional (SO) Windows 10. É importante destacar que o código também foi testado no SO Ubuntu 18.04 com alguns poucos erros, que também ocorreram em ambiente Windows.

Importante ressaltar que antes dos testes feitos, alguns parâmetros foram configurados diretamente pela IHM do CFW11. Foi feito um *Start-up* orientado, que serve para configurar certos parâmetros com as especificações do MIT. Outros parâmetros como formas de comunicação foram definidos todos como 'Remoto' e utilizando protocolo Modbus RTU, assim como definição da taxa de comunicação, pois a escolha da taxa de comunicação através da GUI é válida apenas para o mestre. Estas configurações feitas diretas na IHM são todas salvas no inversor e mesmo após o desligamento os parâmetros permanecerão os mesmos até que sejam alterados, local ou remotamente.

O teste apresentado nas Figuras 5.1, 5.2, 5.3 e 5.4 é um teste onde o usuário cadastrou, em ordem, a seguinte configurações de ventos:

1. Vento tipo 'rampa', com duração de 5 segundos e velocidade final de 10 m/s
2. Vento tipo 'rampa', com duração de 5 segundos e velocidade final de 20 m/s
3. Vento tipo 'rampa', com duração de 5 segundos e velocidade final de 5 m/s
4. Vento tipo 'rampa', com duração de 5 segundos e velocidade final de 0 m/s

Os parâmetros iniciais foram inicializados em:

- Velocidade inicial = 0 m/s

- Tip Speed Ratio = 3
- Comprimento da pá = 1 m

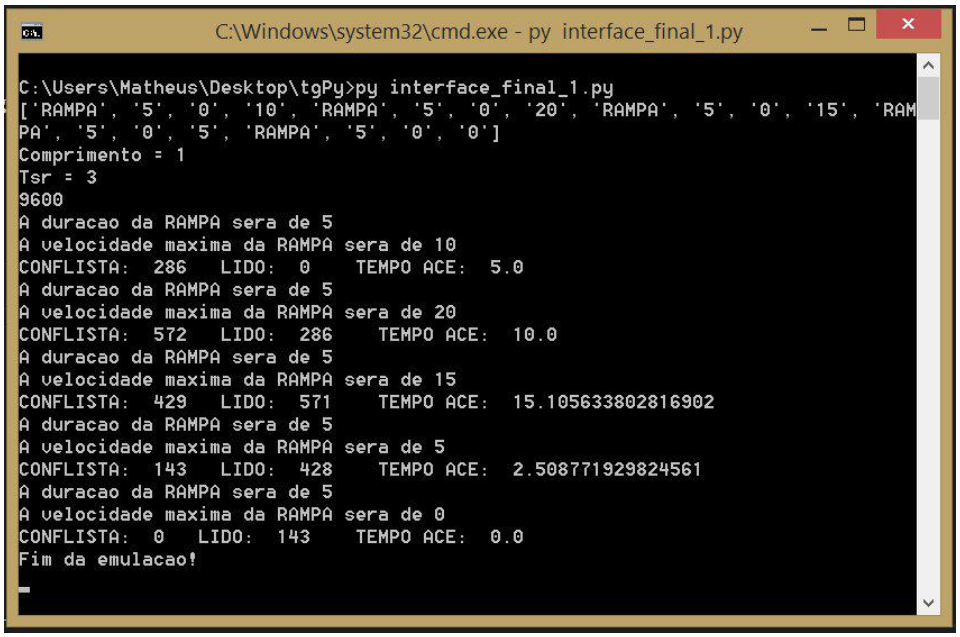
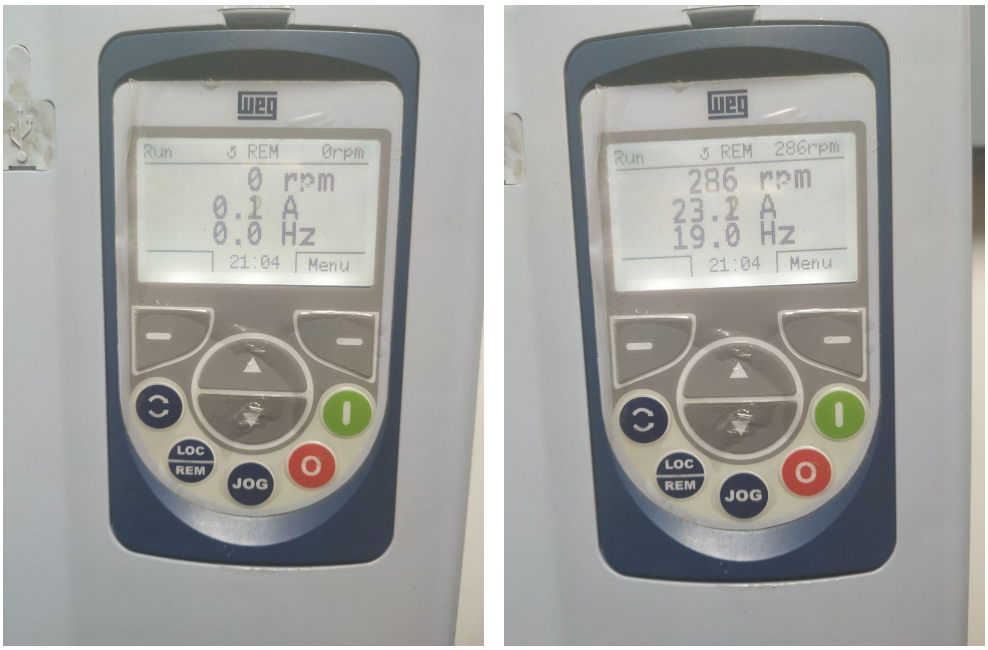
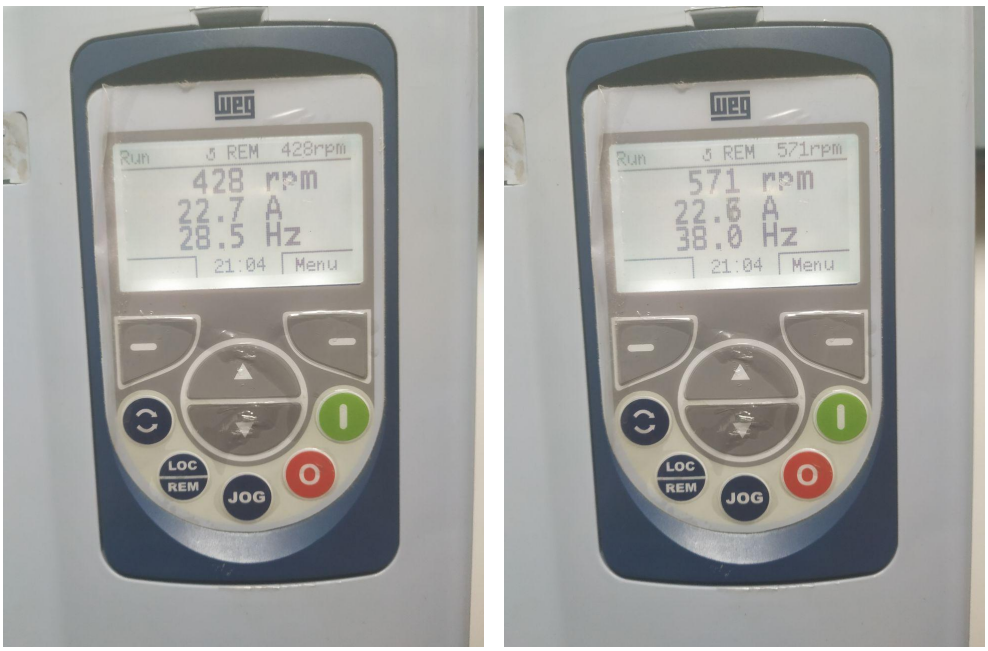


Figura 5.1: Prompt de comando apresentando *prints* conforme a GUI comunica com o inversor. Estes servem apena como forma de depuração do código para que o usuário tenha conhecimento da troca de mensagens.



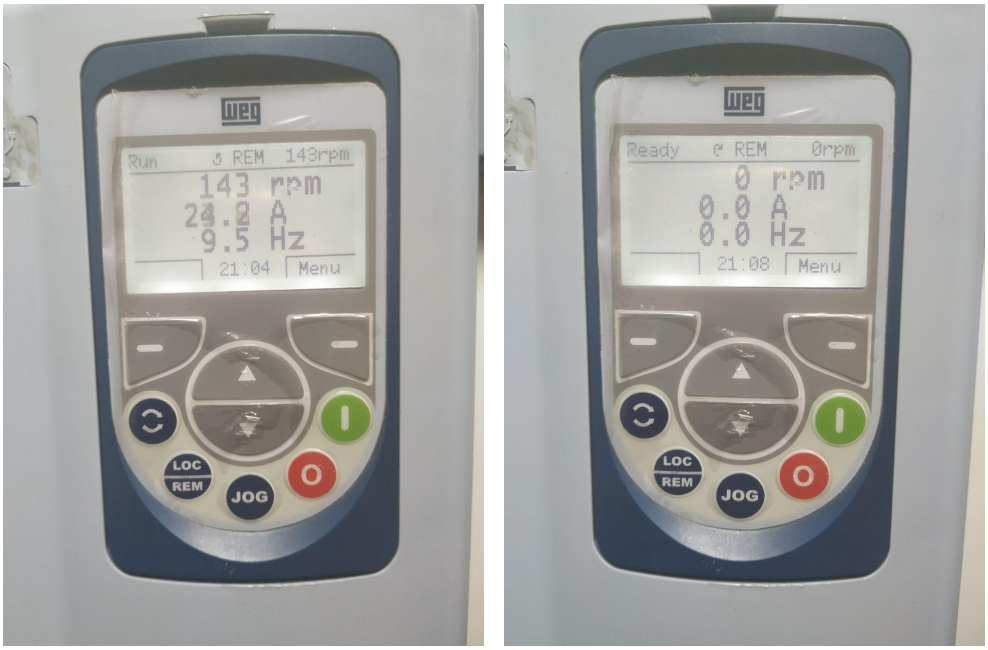
(a) (b)

Figura 5.2: (a) Começo da emulação, estado modificado para *Run*; (b) Primeira rampa concluída, atingida em 5 segundos com velocidade final de 286 rpm.



(a) (b)

Figura 5.3: (a) Segunda rampa com a velocidade acelerando; (b) Segunda rampa concluída, atingida em 5 segundos com velocidade final de 571 rpm.



(a) (b)

Figura 5.4: (a) Terceira rampa concluída, atingida em 5 segundos com velocidade final de 143 rpm; (b) Última rampa concluída, atingida após 5 segundos com velocidade final de 0 rpm. Nota-se também que o estado do inversor foi modificado para **Ready**.

5.2 Erros Ocorridos

Apesar de ser possível realizar emulações bem sucedidas, como demonstrado na Seção 5.1, houveram alguns erros que ocorreram ao longo do desenvolvimento do software. Estes erros ocorrem durante a verificação de soma do CRC e ele pode ocorrer em algumas situações específicas e em outras ocasiões não específicas. Caso o computador seja um notebook e esteja conectado a tomada o erro de *checksum* do CRC irá acontecer. Outro caso em que registramos esse erro foi quando utilizamos algum dispositivo de saída ligado ao notebook, como um monitor. Em casos que o computador portátil não está utilizando um carregador ou um monitor, foram registrados o mesmo tipo de erros porém de forma esporádica.

Uma das hipóteses para os casos em que o notebook está conectado a tomada ou quando se utiliza um monitor é de que haja alguma forma de interferência durante o envio dos bits na comunicação. Já em outros casos, os erros esporádicos podem ser explicados devido à utilização do controle escalar. Segundo [12], o controle escalar não é indicado para motores que rodam a baixas velocidades, que foram o foco de nossas emulações devido à ligação direta do rotor com o gerador. Alguns testes foram feitos para rotações mais altas (por volta de 500 rpm) e estes erros ocorreram com frequência menor.

Capítulo 6

Conclusão

Uma forma de comunicação com o inversor utilizando um computador já existia, como mencionado anteriormente no capítulo introdutório. Apesar da forma de comunicação existir, ela possui alguns empecilhos. Podemos citar alguns deles como ter que utilizar o software LabVIEW da National Instruments, que é pago e, portanto, não é possível utilizá-lo livremente por estudantes ou mesmo professores que não tem acesso a um computador com a licença. A programação feita no LabVIEW apesar de ser uma forma de programação gráfica com objetivo de ser intuitiva e de fácil entendimento, foge um pouco deste objetivo inicial, pois o programa é extenso e não é um *software* utilizado por todas as engenharias.

Com estes obstáculos mencionados, a GUI desenvolvida neste trabalho procurou se tornar mais acessível de fácil entendimento para quaisquer usuários que virão a utilizá-la. A linguagem Python vem sendo cada vez mais utilizada, principalmente nas disciplinas iniciais de programação da Universidade de Brasília. Ela também é multiplataforma, significando que funciona igualmente em SO mais populares, como Windows, Linux e Macintosh. É necessário apenas que o usuário instale os pacotes básicos e alguns específicos para que seja possível a execução do código desenvolvido no trabalho.

Essas facilidades cumpriram os objetivos iniciais citados no Capítulo 1, que torna uma forma intuitiva, de fácil uso, conveniente, rápida e eficaz para qualquer tipo de usuário, com conhecimento ou não de programação em linguagem Python. O Capítulo 5 demonstrou quão rápida pode ser feita uma emulação, mesmo que as configurações precisem ser cadastradas antes.

A biblioteca de comunicação aliada a interface gráfica poderá ajudar pesquisas futuras que utilizarão a bancada de emulação de turbina eólica ou qualquer elemento envolvido nela. Conclui-se, então, que o desenvolvimento deste trabalho será de grande contribuição para o Laboratório de Qualidade de Energia e os pesquisadores que nele trabalham.

6.1 Perspectivas Futuras

A conclusão é extremamente positiva, entretanto o trabalho tem perspectivas futuras e estará sempre aberta a melhoras. O projeto estará disponível e salvo na plataforma GitHub [25], um código *open-source*. O projeto possui arquivo texto com instruções básicas para execução e outras informações básicas. O código principal também foi documentado a fim de detalhar as linhas de código para melhor interpretação do mesmo. Logo, este projeto tem bastante espaço para melhoras, seja na parte da interface gráfica quanto na parte da comunicação.

Mencionados na Seção 5.2, não foi encontrada nenhuma solução para os erros durante o período deste trabalho de graduação, o que pode ser o foco para um projeto de iniciação científica. Como mencionado também nesta seção, a causa dos erros podem estar relacionados ao controle escalar. Uma sugestão para projetos futuros é a implementação de um controle mais robusto, como controle vetorial.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] *Atlas de Energia Eólica*. http://www2.aneel.gov.br/aplicacoes/atlas/pdf/06-energia_eolica(3).pdf. Acessado: 2018-11-09.

[2] *Fontes hidráulicas geram a maior parte da energia elétrica*. http://www.brasil.gov.br/noticias/infraestrutura/2011/12/fontes-hidraulicas-geram-a-maior-parte-da-energia-eletrica. Acessado: 2018-11-09.

[3] *Boletim mensal de geração eólica (Setembro de 2018)*. http://ons.org.br/AcervoDigitalDocumentosEPub BoletimMensaldeGeraÃğÃčoEÃşlica2018-09.pdf. Acessado: 2018-11-09.

[4] *Energia eólica*. https://www.portal-energia.com/energia-eolica/. Acessado: 2019-1-09.

[5] *Brasil atinge 8º lugar em ranking mundial de energia eólica*. https://g1.globo.com/ultimas-noticias/noticia/brasil-atinge-8o-lugar-em-ranking-mundial-de-energia-eolica.ghtml. Acessado: 2019-07-21.

[6] Ronaldo dos Santos Custódio. *Energia Eólica para Produção de Energia Elétrica*. Eletrobrás, 2009.

[7] *Energy Solutions*. https://www.suzlon.com/in-en/energy-solutions. Acessado: 2018-07-05.

[8] *Optimal Rotor Tip Speed Ratio*. https://mragheb.com/NPRE475WindPowerSystems/OptimalRotorTipSpee pdf. Acessado: 2018-07-05.

[9] João Mamede Filho. *Instalações Elétricas Industriais*. 7ª edição. LTC Editora.

[10] Stephen J. Chapman. *Fundamentos de Máquinas Elétricas*. 5ª edição. AMGH Editora Ltda.

[11] *Foto das partes de um motor*. https://pt.wikipedia.org/wiki/Motor_el\%C3\%A9trico. Acessado: 2019-07-07.

[12] Claiton Moro Franchi. *Inversores de frequência Teoria e Aplicações*. 2ª edição. Editora Érika.

[13] *CFW11 Inversor de Frequência*. 2ª edição. Grupo Weg - Unidade Automação Jaraguá do Sul.

[14] *Inversor de Frequçencia CFW-11 V3.1X Manual de Programação*. Grupo Weg - Unidade Automação Jaraguá do Sul, 2012.

[15] *Serial CFW-11 Manual de Comunicação*. Grupo Weg - Unidade Automação Jaraguá do Sul, 2010.

[16] *ComunicaÇÃo serial entre equipamentos*. https://www.novus.com.br/downloads/Arquivos/conceitos\%20\%C3\%A1sicos\%20de\%20rs485\%20e\%20rs422.pdf. Acessado: 2019-07-06.

[17] *Conceitos básicos de RS485 e RS422*. https://docplayer.com.br/15852067-Comunicacao-serial-entre-equipamentos.html. Accessed: 2019-07-06.

[18] *Modbus Organization*. http://www.modbus.org/about_us.php. Acessado: 2019-07-06.

[19] *Welcome to MinimalModbus's documentation!* https://minimalmodbus.readthedocs.io/en/master/. Acessado: 2019-07-01.

[20] *An Introduction to Tkinter*. https://effbot.org/tkinterbook/tkinter-index.htm. Acessado: 2019-07-01.

[21] Bruno Marques Ivo. "Controle de um Motor de Indução trifásico para Emulação de uma Turbina Eólica". Em: *UnB* (jul. de 2017), p. 82.

[22] João Paulo Jorge de Oliveira. "GERADORES SÍNCRONOS A IMãS PERMANENTES APLICADOS A AEROGERADORES: MODELAGEM, OBTENçãO DE PARâMETROS E VALIDAçãO LABORATORIAL". Em: *UnB* (jul. de 2018), p. 203.

[23] *Manual da Comunicação Serial RS232/RS485*. http://www.gigawattsistemas.com.br/file/cfw_411.pdf. Acessado: 2019-07-01.

[24] *Conversor USB para 1 porta serial RS422/485 (DB9M)*. https://www.flexport.com.br/Produtos/23/82/conversor-usb-para-1-porta-serial-rs422485-db9m.html. Acessado: 2019-07-01.

[25] *GitHub - Código da GUI e biblioteca de comunicação disponível em open-source*. https://github.com/yujinmh/WindTurbineEmulationGUI.TG2019-2. Acessado: 2019-07-07.

ANEXOS

I. CÓDIGO EM PYTHON

```
#Code by: Matheus Henrique Dinato Menezes e Victor Yuji Sato
#
#Universidade de Brasília
#
#Trabalho de Graduação 2019/02
#Tema: Desenvolvimento de Interface de Comunicação Baseada no Protocolo Modbus
#para Conexão de um Computador (PC) a um Inversor de Frequência
#
#Orientador: Lélio Ribeiro Soares Júnior

from tkinter import *
from tkinter import filedialog
import minimalmodbus
import serial
import time
import tkinter.messagebox
import random
import math
import numpy as np

#FUNçãO PARA FECHAMENTO DE JANELA
def exitt():
    exit()

#FUNçãO PARA CRIAçãO DE ARQUIVO .TXT COM AS CONFIGURAçõES CRIADAS PELO USUáRIO
def arqtexto():
    global conflista
    f= open("inputdeventos.txt","w+")
    f.write("Tipo:\tTempo:\tAmplitude:\tVelocidade Final\n")
    for i in range (len(conflista)):
        if i%4 == 0:
            f.write( conflista[i] + "\t")
        if i%4 == 1:
            if conflista[i] != 0:
                f.write(conflista[i] + "\t")
        if i%4 == 2:
            if conflista[i] != 0:
                f.write(conflista[i] + "\t\t")
        if i%4 == 3:
            if conflista[i] != 0:
                f.write(conflista[i] + "\n")

    f.close()

#FUNçãO PARA BROWSER, UTILIZADA PARA PROCURAR UM ARQUIVO .TXT COM CONFIGURAçõES Já CRIADAS PREVIAMENTE E
#CARREGAR AS
#CONFIGURAçõES CONTIDAS NELE
def browser():
    global conflista
    aux = []
    string = ''

    # filename = filedialog.askopenfilename(initialdir = "/", title = "Select a File")
    filename = filedialog.askopenfilename(initialdir = "/", title = "Select a File")
    label = Label(root, text=filename, anchor=E, width=24).place(x=23, y=252)
    f = open(filename, "r")
    f.seek(42) # Pular a primeira linha
    msg = f.read()

    # Implementar alguma coisa aqui

    for x in msg:
        i = 0
        if x != '\n' and x != '\t':
            string += x
        elif x == '\n' or x == '\t':
            if string != '':
                aux.append(string)
            string = ''

    conflista.extend(aux)
    f.close()
    print(conflista)
```

```
#FUNçãO PARA ZERAMENTO DE FUNçõES. IMPORTANTE PARA INICIALIZAçãO DE CADA JANELA ABERTA PARA CADA
#CONFIGURAçãO NOVA DE VENTO
def zerar():
    vel_max.set(0)
    ampl.set(0)
    tempo.set(0)

#FUNçãO PARA APAGAR A LISTA COM OS VENTOS SELECIONADOS ANTERIORMENTE
def apagar():
    global conflista
    global count

    count.set(0)
    conflista.clear()
    label = Label(root, anchor=E, width=24).place(x=23, y=252)
    label3 = Label(root,text="Histórico", anchor= NW, bd= 4, relief="groove",
    width=25, height= 13, font=("arial", 12, "bold")).place(x=23, y=320)

#FUNçãO PARA ENCERRAMENTO DE SIMULAçãO
def parar():
    b=Button(root, text="Parar Simulação", width= 22, fg='white', bg= 'brown', relief=GROOVE,
    font=("arial", 13, "italic"), command= b_all)
    b.place(x=300, y=458)

#FUNçãO PARA CRIAçãO DE BOTõES CONTIDOS NA INTERFACE
def b_all():
    button0=Button(root, text="+Add", width= 5, height=1 ,fg='black', bg= 'light gray', justify= CENTER, relief=GROOVE,
    font=("arial", 10, "italic"), command= add)
    button0.place(x=200, y=282)
    button1=Button(root, text="Gerar txt", width= 24, fg='black', bg= 'light gray', relief=GROOVE,
    font=("arial", 13, "italic"), command=arqtexto)
    button1.place(x=300, y=418)
    button2=Button(root, text="Iniciar Emulação", width= 24, fg='black', bg= 'light gray', relief=GROOVE,
    font=("arial", 13, "italic"), command=inversor)
    button2.place(x=300, y=458)
    button3=Button(root, text="Apagar", width= 24, fg='black', bg= 'light gray', relief=GROOVE,
    font=("arial", 13, "italic"), command=apagar)
    button3.place(x=300, y=498)
    button4=Button(root, text="Sair", width= 12, fg='white', bg= 'brown', relief=GROOVE,
    font=("arial", 13, "italic"), command=exitt)
    button4.place(x=420, y=538)
    button5=Button(root, text="Browse", width= 5, height=1 ,fg='black', bg= 'light gray', justify= CENTER, relief=GROOVE,
    font=("arial", 10, "italic"), command= browser)
    button5.place(x=200, y=248)


#FUNçãO PARA ADQUIRIR O TIPO DE VENTO QUE O USUáRIO DESEJAR
def add():
    aux = str(conf.get())
    if aux == 'BRISA':
        brisa()
    elif aux == 'RAMPA':
        rampa()
    elif aux == 'RIPPLE':
        ripple()
    elif aux == 'RAJADA':
        rajada()

#FUNçãO PARA INSERIR NOVAS CONFIGURAçõES DE VENTO EM UMA LISTA. A LISTA é UTILIZADA
#PARA CRIAçãO DE TXT E PARA
#COMEçAR A SIMULAçãO
def peguei():

    global count
    global conflista
    line = int(count.get())
    tipo = str(conf.get())
    t = str(tempo.get())
    a = str(ampl.get())
    v = str(vel_max.get())
    conflista.append(tipo)
    conflista.append(t)
    conflista.append(a)
    conflista.append(v)
    print (conflista)
    count.set(count.get()+1)
    if tipo == "BRISA":
        historico = str(line+1) + ": " + tipo + ": Tempo: " + t + "s"
    elif tipo =="RIPPLE":
        historico = str(line+1) + ": " + tipo + ": Tempo: " + t + " Amplitude: " + a
    elif tipo =="RAJADA":
```

```
        historico = str(line+1) + ": " + tipo + ": Tempo: " + t +"s Velocidade: " + v + "m/s"
    elif tipo =="RAMPA":
        historico = str(line+1) + ": " + tipo + ": Tempo: " + t +"s Velocidade: " + v + "m/s"
    if line < 14:
        label = Label(root,text= historico, font=("arial",7)).place(x=25, y=330 + (count.get()*16))

#FUNçãO RESPONSáVEL POR COMUNICAR COM O INVERSOR. A TROCA DE MENSAGENS OCORRE AQUI
def inversor():
    global conflista
    global pary
    global baud
    global tsr
    global vel_ini
    global comprimento

    instrument = minimalmodbus.Instrument('COM3', 1) # port name, slave address (in decimal)

    if int(baud.get()) == 1:
        instrument.serial.baudrate = 9600   # Baud
    elif int(baud.get()) == 2:
        instrument.serial.baudrate = 19200   # Baud
    elif int(baud.get()) == 3:
        instrument.serial.baudrate = 38400   # Baud
    elif int(baud.get()) == 4:
        instrument.serial.baudrate = 57600   # Baud

    instrument.serial.bytesize = 8

    if int(pary.get()) == 1:
        instrument.serial.parity   = serial.PARITY_NONE
    elif int(pary.get()) == 2:
        instrument.serial.parity   = serial.PARITY_EVEN
    elif int(pary.get()) == 3:
        instrument.serial.parity   = serial.PARITY_ODD

    # instrument.debug = True
    instrument.serial.stopbits = 1
    instrument.serial.timeout  = 0.05 # seconds
    instrument.serial.xonxoff = False
    instrument.precalculate_read_size = False
    instrument.mode = minimalmodbus.MODE_RTU   # rtu or ascii mode

    # INICIAR VALORES NO INVERSOR #


    time.sleep(2)
    instrument.write_register(682,0x0014,functioncode = 6)   # Deixar ele em modo operacional
    # time.sleep(2)
    instrument.write_register(100,0,functioncode = 6)
    instrument.write_register(134,1500,functioncode = 6)      # Setar a velocidade máxima para 1800
    instrument.write_register(133,0,functioncode = 6)         # Setar a velocidade mínima para 0
    instrument.write_register(683,0,functioncode = 6)      # Setar a velocidade de referência para 0
    time.sleep(2)
    instrument.write_register(682,0x0013,functioncode = 6)   # Deixar ele em modo operacional
    ini = (int(tsr.get())*int(vel_ini.get())*9.549)/int(comprimento.get())
    time.sleep(5)
    instrument.write_register(100,5, 1,functioncode = 6)
    time.sleep(0.05)
    instrument.write_register(134,ini,functioncode = 6)      # Setar a velocidade máxima para 1800
    time.sleep(0.05)
    ini = ini*8045/885
    time.sleep(0.05)
    instrument.write_register(683,ini,functioncode = 6)      # Setar a velocidade de referência para 0
    time.sleep(5)
    # instrument.write_register(100,10,functioncode = 6)
    # time.sleep(2)

    i = 0
    # print(i)

    for i in range(len(conflista)):
        if i%4 == 0:

            if conflista[i] == "BRISA":
                print("T brisa = " + conflista[i+1]+ "s")
                j=0
                for j in range (1,int(conflista[i+1])+1):
                    time.sleep(0.011)
                    vel_anterior = int(instrument.read_register(1,0))
                    time.sleep(0.011)
                    a = random.randint(-5, 5)
```

```
            vento =  vel_anterior + a
            print(j,"s")
            if vento <= 0:
                vento = 3
            elif vento > 1000:
                vento = 999
            time.sleep(0.011)
            # instrument.write_register(134,100, functioncode = 6)
            # time.sleep(0.011)
            # instrument.write_register(100,
            (int(vento))/int(vento)-int(vel_anterior),functioncode = 6)
            # instrument.write_register(100,10,functioncode = 6)
            vento = vento*8045/885
            time.sleep(0.011)
            instrument.write_register(683,vento,functioncode = 6)
            time.sleep(1)

    elif conflista[i] == "RAJADA":
        print("T rajada = " + conflista[i+1]+ "s")
        print("Vm rajada = " + conflista[i+3]+ "m/s")

        conflista[i+3] = (int(tsr.get())*int(conflista[i+3])*9.549)/int(comprimento.get())
        v_ini = int(instrument.read_register(1,0))
        print ("VEL_INI = ", str(vel_ini))
        time.sleep(0.05)

        if conflista[i+3] > instrument.read_register(1, 0):
            #SUBIDA
            time.sleep(0.05)
            instrument.write_register(134,int(conflista[i+3]), functioncode=6)
            ace_pos = (int(conflista[i+1])*int(conflista[i+3]))/
            (2*(int(conflista[i+3])-int(instrument.read_register(1,0))))
            time.sleep(0.05)
            instrument.write_register(100,ace_pos,1, functioncode=6)
            time.sleep(0.05)
            vel_pos = int(conflista[i+3])*8045/885
            time.sleep(0.05)
            instrument.write_register(683,vel_pos, functioncode=6)
            t_raj = (int(conflista[i+1])/2) #- 0.2
            time.sleep(t_raj)

            #DESCIDA
            # ace_neg = (int(conflista[i+1])*int(conflista[i+3]))/
            (2*(int(instrument.read_register(1,0))-v_ini))
            ace_neg = (int(conflista[i+1])*int(conflista[i+3]))/(2*(int(conflista[i+3])-v_ini))
            print("ACE = ", ace_neg)
            time.sleep(0.05)
            instrument.write_register(101,ace_neg,1, functioncode=6)
            time.sleep(0.05)
            vel_neg = v_ini*8045/885
            instrument.write_register(683,vel_neg, functioncode=6)
            time.sleep(t_raj)

        elif conflista[i+3] == instrument.read_register(1,0):
            time.sleep(int(conflista[i+1]))

        elif conflista[i+3] < instrument.read_register(1,0):
            #DESCIDA
            ace_neg = (int(conflista[i+1])*int(conflista[i+3]))/
            (2*(int(instrument.read_register(1,0) - int(conflista[i+3]))))
            time.sleep(0.05)
            instrument.write_register(101,ace_neg,1, functioncode=6)
            time.sleep(0.05)
            vel_neg = int(conflista[i+3])*8045/885
            instrument.write_register(683,vel_neg, functioncode=6)
            t_raj = int(conflista[i+1])/2
            time.sleep(t_raj)

            #SUBIDA
            ace_pos = (int(conflista[i+1])*int(conflista[i+3]))/
            (2*(int(conflista[i+3])-int(instrument.read_register(1,0))))
            time.sleep(0.05)
            instrument.write_register(100,ace_pos,1, functioncode=6)
            time.sleep(0.05)
            vel_pos = v_ini*8045/885
            instrument.write_register(683,vel_pos, functioncode=6)
            time.sleep(t_raj)

    elif conflista[i] == "RIPPLE":
        print("T ripple = " + conflista[i+1]+ "s")
        print("Amp ripple = " + conflista[i+2])
```

```
                t_amostragem = int(int(conflista[i+1])/0.5)
                time.sleep(0.05)
                V_inic = int(instrument.read_register(1,0))
                amp = int((int(tsr.get())*int(conflista[i+2])*9.549)/2*int(comprimento.get()))
                v_max = V_inic + amp + amp
                time.sleep(0.05)
                instrument.write_register(134,v_max, functioncode=6)
                time.sleep(0.05)
                instrument.write_register(100,1, functioncode=6)
                time.sleep(0.05)
                instrument.write_register(101,1, functioncode=6)
                time.sleep(0.05)
                for i in range (1, t_amostragem):
                    V_ripple = V_inic + amp*(np.sin((np.pi*i/10)+(3*np.pi/2))) + amp
                    V_ripple = math.floor(V_ripple*8045/885)
                    time.sleep(0.05)
                    instrument.write_register(683, V_ripple, functioncode=6)
                    # print (V_ripple)
                    time.sleep(0.5)

            elif conflista[i] == "RAMPA":
                print("T rampa = " + str(conflista[i+1]) + "s")
                print("Vm rampa = " + str(conflista[i+3]) + "m/s")

                conflista[i+3] = (int(tsr.get())*int(conflista[i+3])*9.549)/int(comprimento.get())
                conflista[i+3] = math.floor(conflista[i+3])
                time.sleep(0.05)

                if int(conflista[i+3]) > int(instrument.read_register(1, 0,functioncode = 3)):
                    time.sleep(0.05)
                    instrument.write_register(134,int(conflista[i+3]), functioncode = 6)
                    escrita = (int(conflista[i+1])*int(conflista[i+3]))/
                    (int(conflista[i+3])-int(instrument.read_register(1,0)))
                    escrita = math.floor(escrita)
                    time.sleep(0.05)
                    instrument.write_register(100,escrita,1, functioncode = 6)
                    escrita2 = int(conflista[i+3])*8045/885
                    escrita2 = math.floor(escrita2)
                    time.sleep(0.05)
                    instrument.write_register(683,escrita2, functioncode = 6)
                    time.sleep(int(conflista[i+1]))

                elif conflista[i+3] == instrument.read_register(1,0):
                    time.sleep(int(conflista[i+1]))

                elif int(conflista[i+3]) < int(instrument.read_register(1, 0)):
                    escrita = (int(conflista[i+1])*int(conflista[i+3]))/
                    (int(instrument.read_register(1,0) - int(conflista[i+3])))
                    escrita = math.floor(escrita)
                    if escrita == 0:
                        escrita = int(conflista[i+1])
                    time.sleep(0.05)
                    instrument.write_register(101,escrita,1,functioncode = 6)
                    escrita2 = int(conflista[i+3])*8045/885
                    escrita2 = math.floor(escrita2)
                    time.sleep(0.05)
                    instrument.write_register(683,escrita2, functioncode = 6)
                    time.sleep(int(conflista[i+1]))
                    instrument.write_register(134,int(conflista[i+3]), functioncode = 6)

    print('Fim da emulacao!')
    apagar()

    instrument.write_register(682,0x0014, functioncode=6)   # Deixar ele em modo operacional


# VENTOS #
#FUNçõES ('brisa', 'ripple', 'rajada' e 'rampa') PARA CRIAçãO DE JANELAS
#PARA CADA VEZ QUE O USUáRIO DESEJAR ADICIONAR
#UMA NOVA CONFIGURAçãO DE VENTO
def brisa():
    zerar()
    wBrisa = Toplevel()
    wBrisa.title("BRISA")
    wBrisa.geometry("300x120+500+300")
    wBrisa.resizable(width=False, height=False)
    l1 = Label(wBrisa, text= "Duração (s): ").place(x=20, y=20)
    e1 = Entry(wBrisa, textvar= tempo).place(x=150, y=20)

    b0=Button(wBrisa, text="Salvar", width= 8, height=1 ,fg='black', bg= 'light blue', relief=GROOVE,
```

```
    font=("arial", 13, "italic"), command=peguei)
    b0.place(x=85, y=70)
    b1=Button(wBrisa, text="Sair", width= 8, height=1 ,fg='black', bg= 'light blue', relief=GROOVE,
    font=("arial", 13, "italic"), command=wBrisa.destroy)
    b1.place(x=185, y=70)

def ripple():
    zerar()
    wRipple = Toplevel()
    wRipple.title("RIPPLE")
    wRipple.geometry("300x160+500+300")
    wRipple.resizable(width=False, height=False)
    l1 = Label(wRipple, text= "Amplitude (m/s): ").place(x=20,y=20)
    l2 = Label(wRipple, text= "Duração (s): ").place(x=20, y=60)
    e1 = Entry(wRipple, textvar= ampl).place(x=150, y=20)
    e2 = Entry(wRipple, textvar= tempo).place(x=150, y=60)
    b0=Button(wRipple, text="Ok", width= 8, height=1 ,fg='black', bg= 'light blue', relief=GROOVE,
    font=("arial", 13, "italic"), command=peguei)
    b0.place(x=85, y=100)
    b1=Button(wRipple, text="Cancel", width= 8, height=1 ,fg='black', bg= 'light blue', relief=GROOVE,
    font=("arial", 13, "italic"), command=wRipple.destroy)
    b1.place(x=185, y=100)

def rajada():
    zerar()
    wRajada = Toplevel()
    wRajada.title("RAJADA")
    wRajada.geometry("300x160+500+300")
    wRajada.resizable(width=False, height=False)
    l1 = Label(wRajada, text= "Velocidade final (m/s): ").place(x=20,y=20)
    l2 = Label(wRajada, text= "Duração (s): ").place(x=20, y=60)
    e1 = Entry(wRajada, textvar= vel_max).place(x=150, y=20)
    e2 = Entry(wRajada, textvar= tempo).place(x=150, y=60)
    b0=Button(wRajada, text="Ok", width= 8, height=1 ,fg='black', bg= 'light blue', relief=GROOVE,
    font=("arial", 13, "italic"), command = peguei)
    b0.place(x=85, y=100)
    b1=Button(wRajada, text="Cancel", width= 8, height=1 ,fg='black', bg= 'light blue', relief=GROOVE,
    font=("arial", 13, "italic"), command=wRajada.destroy)
    b1.place(x=185, y=100)

def rampa():
    zerar()
    wRampa = Toplevel()
    wRampa.title("RAMPA")
    wRampa.geometry("300x160+500+300")
    wRampa.resizable(width=False, height=False)
    l1 = Label(wRampa, text= "Velocidade final (m/s): ").place(x=20,y=20)
    l2 = Label(wRampa, text= "Duração (s): ").place(x=20, y=60)
    e1 = Entry(wRampa, textvar= vel_max).place(x=150, y=20)
    e2 = Entry(wRampa, textvar= tempo).place(x=150, y=60)
    b0=Button(wRampa, text="Ok", width= 8, height=1 ,fg='black', bg= 'light blue', relief=GROOVE,
    font=("arial", 13, "italic"), command=peguei)
    b0.place(x=85, y=100)
    b1=Button(wRampa, text="Cancel", width= 8, height=1 ,fg='black', bg= 'light blue', relief=GROOVE,
    font=("arial", 13, "italic"), command=wRampa.destroy)
    b1.place(x=185, y=100)

########################################## INíCIO DA MAIN ##########################################

root=Tk()

w_soft = 570
h_soft = 600
screen_w = root.winfo_screenwidth()
screen_h = root.winfo_screenheight()
x_pos = (screen_w/2)-(w_soft/2)
y_pos = (screen_h/2)-(h_soft/2)

root.geometry("%dx%d+%d+%d" % (w_soft, h_soft, x_pos, y_pos))
root.title("Emulador de vento")
root.resizable(width=False, height=False)
# root.configure(bg = "#94d42b")

# VARIABLES #
conflista = []
count = IntVar()
baud = IntVar()
pary = IntVar()
conf = StringVar()
vel_ini = IntVar()
vel_max = IntVar()
```

```
ampl = StringVar()
tempo = StringVar()
tsr = IntVar()
comprimento = IntVar()

# INICIAR VARIáVEIS #

tsr.set(1)
comprimento.set(1)
pary.set(1)
baud.set(1)

# LABELS #
label1 = Label(root,text="Gráfico", anchor= NW, bd= 4, relief="groove", width=25, height= 11,
font=("arial", 12, "bold")).place(x=23, y=20)
label2 = Label(root,text="BaudRate (bit/s)", anchor= NW, bd= 4, relief="groove", width=24, height= 6,
font=("arial", 12, "bold")).place(x=300, y=20)
label3 = Label(root,text="Histórico", anchor= NW, bd= 4, relief="groove", width=25, height= 13,
font=("arial", 12, "bold")).place(x=23, y=320)
label4 = Label(root,text="Paridade", anchor= NW, bd= 4, relief="groove", width=24, height= 5,
font=("arial", 12, "bold")).place(x=300, y=150)
label5 = Label(root,text="Par metros Inicias", anchor= NW, bd= 4, relief="groove", width=24, height= 7,
font=("arial", 12, "bold")).place(x=300, y=260)
label6 = Label(root,text="Velocidade Inicial\n(m/s)", font=("arial", 8)).place(x=305, y=290)
label7 = Label(root,text="Tip Speed Ratio", font=("arial", 8)).place(x=305, y=320)
label8 = Label(root,text="Comprimento da pá\n(m)", font=("arial", 8)).place(x=305, y=350)

# ENTRY #
e1 = Entry(root, textvar= vel_ini).place(x= 405, y=290)
e2 = Entry(root, textvar= tsr).place(x=405, y=320)
e3 = Entry(root, textvar= comprimento).place(x=405, y=350)

# COMBO BOX #
list1 = ['BRISA', 'RAMPA', 'RIPPLE', 'RAJADA']
droplist = OptionMenu(root, conf, *list1)
conf.set("Selecione a configuração")
droplist.config(width=22)
droplist.place(x=20, y = 280)

# RADIO BUTTONS #
b1 = Radiobutton(root, text="9600", variable= baud, value= 1).place(x=310, y=50)
b2 = Radiobutton(root, text="19200", variable= baud, value= 2).place(x=310, y=70)
b3 = Radiobutton(root, text="38400", variable= baud, value= 3).place(x=310, y=90)
b4 = Radiobutton(root, text="57600", variable= baud, value= 4).place(x=310, y=110)

p1 = Radiobutton(root, text="None", variable= pary, value= 1).place(x=310, y=180)
p2 = Radiobutton(root, text="Even", variable= pary, value= 2).place(x=310, y=200)
p3 = Radiobutton(root, text="Odd", variable= pary, value= 3).place(x=310, y=220)

b_all()

root.mainloop()
```