UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

FABRICIO CASSANHO TEODORO

AUTOMATIZAÇÃO DE UM TRIBÔMETRO PIN ON DISC

DISSERTAÇÃO DE MESTRADO

CORNÉLIO PROCÓPIO
2023

FABRICIO CASSANHO TEODORO

AUTOMAÇÃO DE UM TRIBÔMETRO PIN ON DISC

Automation of a tribometer pin on disc

Dissertação apresentada como requisito para obtenção do título de Mestre em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná (UTFPR).
Orientador: Alessandro do Nascimento Vargas

Cornélio Procópio

2023

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Campus Cornélio Procópio

FABRICIO CASSANHO TEODORO

AUTOMATIZAÇÃO DE UM TRIBÔMETRO PIN ON DISC

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestre Em Engenharia Elétrica da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Sistemas Eletrônicos Industriais.

Data de aprovação: 02 de Outubro de 2023

Dr. Alessandro Do Nascimento Vargas, Doutorado - Universidade Tecnológica Federal do Paraná

Dr. Marcio Aurelio Furtado Montezuma, Doutorado - Universidade Tecnológica Federal do Paraná

Dr. Ricardo Breganon, Doutorado - Instituto Federal do Paraná

Documento gerado pelo Sistema Acadêmico da UTFPR a partir dos dados da Ata de Defesa em 02/10/2023.

AGRADECIMENTOS

Agradeço primeiramente a DEUS por permitir a realização desse trabalho inspirada nas palavras sagradas contidas em $Proverbios\ 16:3$.

"Confia ao Senhor as tuas obras, e teus pensamentos serão estabelecidos."

Agradeço a minha esposa Lísia, aos meus pais José e Maria e a minha irmã Fernanda pelo esforço e base familiar que me forneceram durante essa caminhada.

Agradeço também aos professores Vargas e Montezuma que me ofertaram uma chance de crescimento profissional, acadêmico e pessoal. E estendo a todos coloboradores da UTFPR-CP que participaram de forma direta ou indireta para construção desse trabalho.

RESUMO

TEODORO, Fabricio. Automatização de um Tribômetro *PIN ON DISC*. 2023. 61 f. Dissertação de Mestrado – Programa De Pós-Graduação Em Engenharia Elétrica, Universidade Tecnológica Federal do Paraná. Cornélio Procópio, 2023.

O presente trabalho tem por objetivo a automatização/controle de um tribômetro, este equipamento é utilizado para pesquisa e estudo sobre fenômenos ligados ao atrito. Para isso foi efetuado uma revisão bibliográfica de conceitos relacionado aos componentes que compõe este equipamento, tais como: acionamento elétrico (motor, inversor de frequência e circuitos de comando), componentes eletrônicos dedicados a leitura de variáveis (sensores de temperatura, rotação e célula de carga), processamento de dados (microcontrolador) e interface computacional. Na sequência foi construído um painel elétrico que pudesse agrupar todos os componentes eletrônicos para atender as necessidades do experimento tribológico, também fora desenvolvido uma lógica de funcionamento do equipamento e traduzida em linguagem de programação para ser utilizado no microcontrolador. Posteriormente todo conjunto foi ligado e testado, operando de forma satisfatória dentro das condições pré-estabelecidas.

Palavras-chave: Tribômetro, automatização, controle e eletrônica.

ABSTRACT

TEODORO, Fabricio. Automation of a PIN ON DISC Tribometer. 2023. 60 f. Dissertation Master's Degree – Postgraduate Program in Electrical Engineering, Technological University Federal of Parana. Cornelio Procopio, 2023.

The objective of this work is the automation/control of a tribometer, this equipment is used for research and study on phenomena related to friction. For this, a bibliographic review of concepts related to the components that make up this equipment was carried out, such as: electric drive (motor, frequency inverter and control circuits), electronic components dedicated to reading variables (temperature, rotation and load cell sensors), data processing (microcontroller) and computational interface. Next, an electrical panel was built that could group all the electronic components to meet the needs of the tribological experiment. An operating logic for the equipment was also developed and translated into programming language to be used in the microcontroller. Subsequently, the entire set was connected and tested, operating satisfactorily within the pre-established conditions.

Keywords: Tribometer, automation, control and eletronics.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADC	*Analog to digital converter*
ARM	*Advanced RISC machine*
AHB	*Advanced high performance bus*
APB1	*Advanced peripheral bus 1*
APB2	*Advanced peripheral bus 2*
bps	*Bits per second*
COM	Interface de comunicação serial
CPU	*Central process unit*
CV	Cavalo vapor
DAC	*Digital to analog converter*
DMA	Direct memory access controller
GCC	Conjunto de compiladores multi plataforma
GDB	*GNU project debugger*
GND	*Ground*
GNU	*GNU compile collection*
GPIO	*General purpose input output*
IGBT	*Insulated-gate bipolar transistor*
Hz	*Hertz*
I2C	*Inter integrated circuit*
I/O	*Input/output*
K	*Kelvin*
LCD	*Liquid crystal display*
LPT	*Local printer terminal*
MOSFET	*Metal Oxide Semiconductor Field Effect Transistor*

NTC	*Negative Temperature Coefficient*
PC	*Personal computer*
PLL	*Phase-locked Loop*
PWM	Modulação por largura de pulso
RAM	*Random access memory*
RCC	*Reset and Clock Control*
RS	*Recommended standard*
RTC	*Real time clock*
SD	*Secure digital*
SPI	*Serial peripheral interface*
SPWM	*Sinusoidal pulse width modulation*
TIA	*Telecommunication industry association*
TTL	*Transistor-transistor logic*
UART	*Universal asynchronous receiver/transmitter*
USART	*Universal synchronous-asynchronous receiver/transmitter*
USB	*Universal serial bus*
Vac	*Voltage alternate current*
Vdc	*Voltage direct current*

LISTA DE SÍMBOLOS

A	Coeficiente obtido pela medição de resitência (Ω) em relação a temperatura de 0°C e aplicado a equação de *Steinhart-Hart*
B	Coeficiente obtido pela medição de resitência (Ω) em relação a temperatura de 25°C e aplicado a equação de *Steinhart-Hart*
C	Coeficiente obtido pela medição de resitência (Ω) em relação a temperatura de 100°C e aplicado a equação de *Steinhart-Hart*
D_{or}	Registrador em *bits* para o DAC
n_{esc}	Velocidade de escorregamento (RPM)
n_m	Velocidade do rotor (RPM)
n_{sinc}	Velocidade do campo magnético girante (RPM)
R_1	Resistência do extensômetro 1 (Ω)
R_2	Resistência do extensômetro 2 (Ω)
R_3	Resistência do extensômetro 3 (Ω)
R_4	Resistência do extensômetro 4 (Ω)
s	Escorregamento em porcentagem ($\%$)
V_i	Tensão de entrada da ponte *Wheatstone* (V)
V_{PB0}	Tensão de entrada ADC (V)
V_{ADC}	Tensão no divisor resistivo (V)
V_o	Tensão de saída da ponte *Wheatstone* (V)
V_{out}	Tensão de saída do DAC do microcontrolador (V)
V_{ref}	Tensão de referência do microcontrolador (V)
T	Temperatura do termistor (°K)
α	Coeficiente de variação da corrente de curto circuito pela temperatura (A/°K)

SUMÁRIO

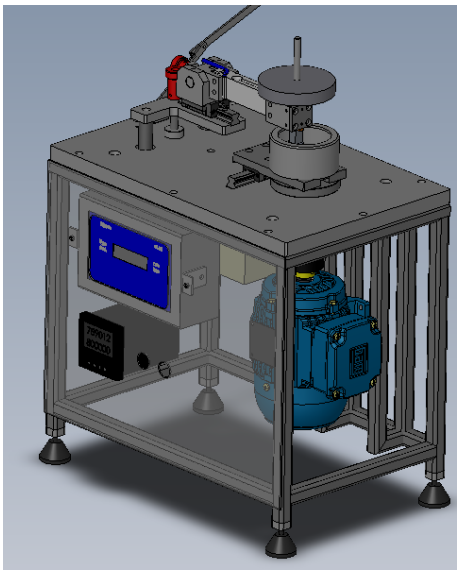
1 INTRODUÇÃO

A ciência que estuda os fenômenos relativos ao atrito, desgaste e lubrificação é denominada tribologia, esse vocábulo tem origem do grego "tribo", que significa esfregar, e "logos" estudo, esse foi utilizado pela primeira vez em 1966 durante a apresentação ao comitê do departamento inglês de educação, sendo definido e utilizado até hoje, como o ramo da ciência e tecnologia interessada no estudo da interação entre as superfícies em movimento (JOST, 1990).

Leonardo da Vinci é considerado o pai da tribologia moderna, ele realizou importantes contribuições para o entendimento do atrito e do desgaste, da Vinci mediu através de experimentos as forças de atrito em planos horizontais e inclinados, demonstrando que essas forças são dependentes da força normal ao deslizamento dos corpos e independentes da área de contato. Também foi proposta uma diferenciação entre atritos de escorregamento e rolamento, e introduziu o coeficiente de atrito proporcional à força normal, logo esse equipamento construído por Leonardo da Vinci foi considerado o primeiro tribômetro documentado na história (JOST, 1990).

O tribômetro consiste em um sistema mecânico, elétrico e eletrônico empregado na medição de atrito, desgaste e lubrificação de superfície. Em linhas gerais, o equipamento pode ser empregado para: a simulação de contato para aplicações específicas, avaliação das propriedades de atrito e desgaste dos materiais e o desempenho dos lubrificantes bem como sua propriedade antidesgaste. O tribômetro pino sobre disco, mais conhecido como *pin on disc* é demonstrado conforme a Figura 1.

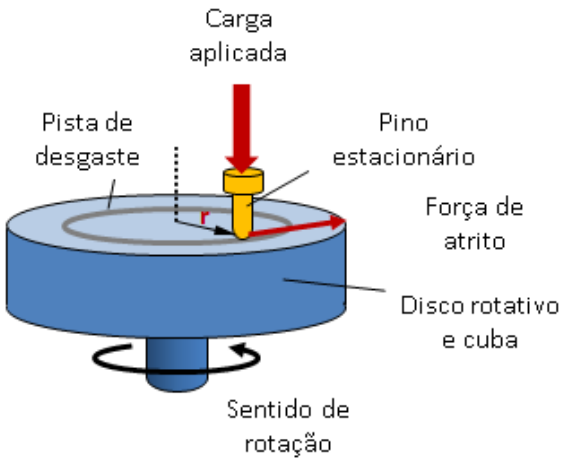
Figura 1 – Projeto de um tribômetro *pin on disc*



Fonte: (SOARES, 2014)

Este equipamento consiste em um disco giratório, cuba para retenção do óleo lubrificante e um pino estacionário, em que uma carga é aplicada no pino estacionário e as leituras de atrito são registradas em uma célula de carga, de acordo com a Figura 2.

Figura 2 – Esquema básico de um tribômetro *pin on disc*



Fonte: Autoria própria

A necessidade de submeter materiais à análises tribológicas torna-se mais indispensável na indústria, para validação de propriedades mecânicas, definição de regimes de lubrificação ou até mesmo para auxílio no planejamento de rotinas de manutenções (ACCADROLLI; VERNEY, 2017). Estima-se que quase um quarto dos recursos energéticos do mundo sejam usados para vencer o atrito de forma direta ou indireta (HOLMBERG; ERDEMIR, 2017). Na indústria, setor que compreende maior número de máquinas e equipamentos, a perda energética por atrito e seus relacionados chega a trinta porcento (DOWSON, 1979). Logo, máquinas com melhores soluções tribológicas aumentam a eficiência e produção, e também menor será a frequência de paradas para manutenções corretivas (SANTOS, 2021).

Visando trazer uma contribuição para o conhecimento em tribologia, esse trabalho apresenta um sistema eletrônico embarcado que proporcionará o aperfeiçoamento das técnicas/medições tribológicas contidas no padrão (ASTM-G99-17, 2020). Assim sendo um sistema embarcado é determinado por um sistema computacional, conjunto de *hardware* e *software*, projetado para executar uma tarefa específica em um sistema maior (tribômetro), propondo controlar ou monitorar uma determinada função, ou processo.

Os sistemas embarcados são projetados para serem compactos, confiáveis e eficientes em termos de energia. Eles também precisam lidar com restrições de espaço e recursos, como memória e processamento limitados (LEE; SESHIA, 2016). No caso desse projeto, foi construído um sistema eletrônico embarcado e integrado a um tribômetro *pin on disc*, capaz de realizar as seguintes tarefas:

- Leitura de parâmetros - Sensores eletrônicos que executam a leitura de variáveis específicas como a temperatura do experimento e rotação do disco, estes componentes foram ligados

ao microcontrolador para a aquisição dessas informações, já que são exigidas pelo padrão (ASTM-G99-17, 2020).

- Indicações - Instalados sinalizadores LCD que informam o status operacional do equipamento (operando, parado, habilitado e local/remoto), bem como os parâmetros lidos dos sensores no *Display* LCD. Todos esses foram ligados ao microcontrolador e disponibilizados ao usuário.
- Canais de comunicação - Foram configurados duas interface seriais no microcontrolador, uma dedicada ao enlace com o *display* da célula de carga, e a outra com o computador. Estes canais são responsáveis pela coleta de informação provininete do sensor *strain gage* e a disponibilização desta informação ao PC.
- Seleção de comandos - O conjunto possui dois tipos de seleção de operação, a primeira em local onde os comandos (partida, parada, *reset* e habilitado) podem ser efetuados diretamente no painel que compõe o equipamento, a outra forma seria execução desses mesmos comandos remotamente entre o microcontrolador e um computador.
- Controle e automatização - O controle do experimento será efetuado pela manipulação da rotação do disco, essa tarefa é realizada pelo microcontrolador que envia um sinal ao conjunto tribômetro, de forma a manter uma rotação específica de acordo com ensaio. Para automatização é executada uma lógica programada também no microcontrolador que possibilita a execução de rotinas temporizadas, tratamento de dados, canais de comunicação e modos operacionais.
- Armazemamento de dados - Ao ser iniciado o experimento, as medições efetuadas pelo sensor *strain gage* serão armazenadas eletronicamente no microcontrolador e computador, afim de que possa ser analisada para avaliação do ensaio.

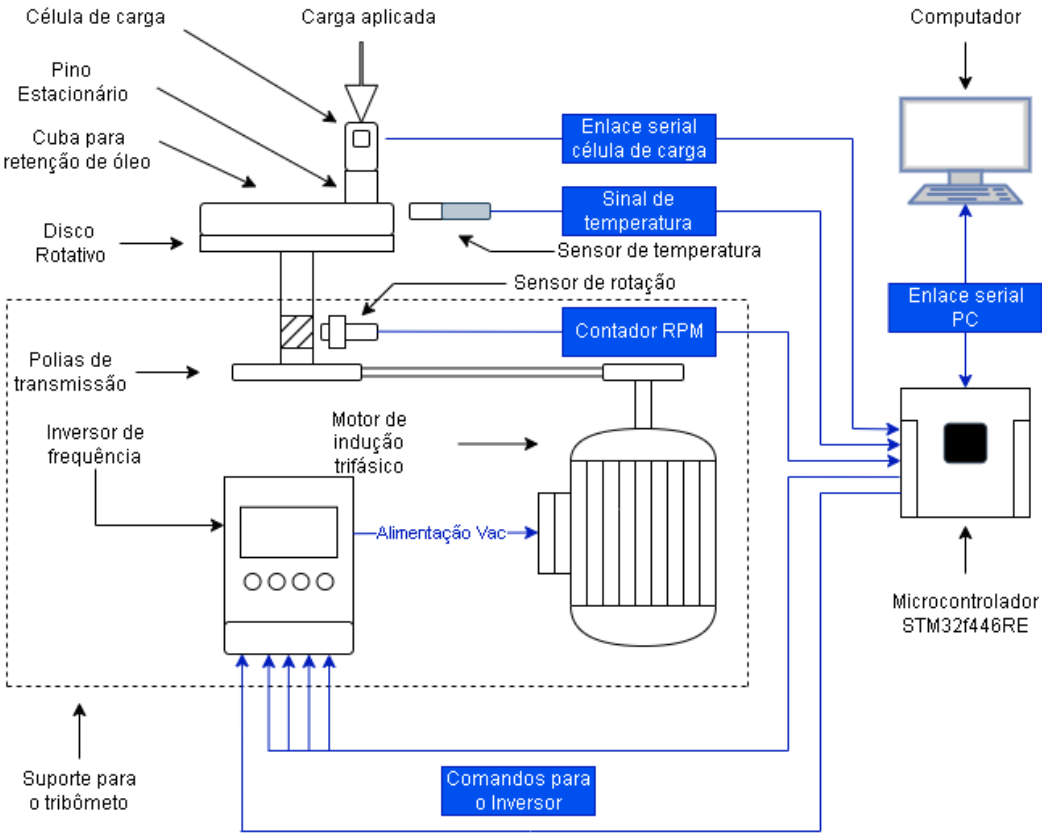
As tarefas supracitadas contemplam o funcionamento e operação do conjunto tribômetro onde são demonstradas as implementações de controle e monitoração.

Permitindo que os sistemas eletrônicos embarcados aumentem a confiabilidade do processo em questão, contribuindo na melhor obtenção de resultados. Os componentes eletrônicos empregados no conjunto tribômetro serão descritos com maiores detalhes na sequência do trabalho, observando que não foi encontrado nenhuma solução similar disponível no mercado, portanto este projeto representa um protótipo construido e dedicado a pesquisa em laboratório. Assim, objetivo desse trabalho foi automatizar um tribômetro pin on disc através de um sistema eletro eletronico embarcado.

2 MATERIAIS E MÉTODOS

Esta seção descreve os componentes eletroeletrônicos utilizados na automatização de um tribômetro *pin on disc* segundo a norma ASTM G99 que preconiza a aquisição de parâmetros como: carga, velocidade angular, temperatura e tempo de experimento. Conforme a Figura 3 mostra uma forma mais detalhada a construção mecânica, equipamentos elétricos e componentes eletrônicos.

Figura 3 – Esquemático detalhado dos equipamentos



Fonte: Autoria própria

A estrutura do tribômetro foi segmentada em partes como: conjunto motor elétrico, inversor de frequência, célula de carga, sensores e componentes eletrônicos, elaboração de *firmware* e *software* de aplicação, aquisição de dados, acionamentos eletromecânicos e circuitos elétricos para alimentação.

2.1 Conjunto motor elétrico, inversor de frequência e célula de carga

2.1.1 Conjunto motor elétrico

Os motores elétricos tem como objetivo a transformação de energia elétrica em mecânica, e nesse trabalho será tratado motores de corrente alternada. Usualmente, são

classificados em dois grupos, os assíncrono e síncrono.

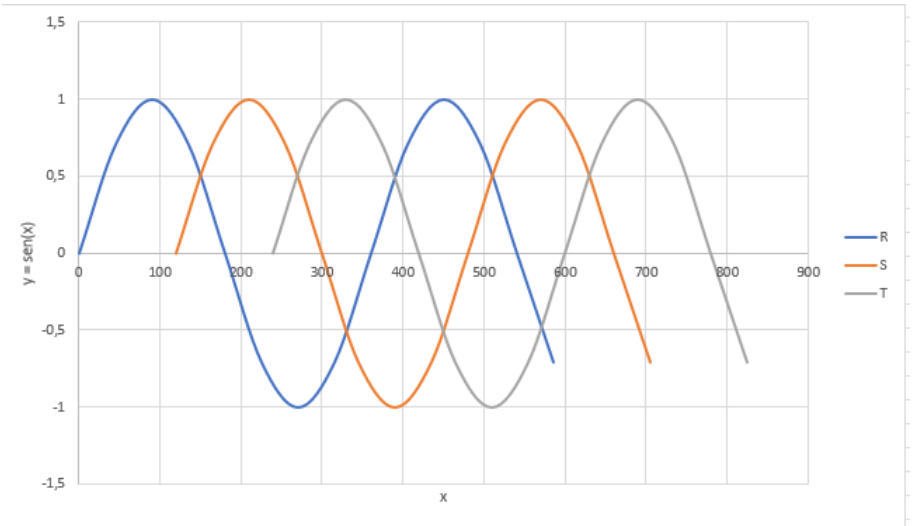
O motor síncrono possui um enrolamento no rotor que circula uma corrente contínua em um enrolamento trifásico no estator. Para o seu funcionamento aplica-se uma tensão contínua no rotor, esta tensão irá produzir um campo magnético fixo em torno do rotor. Já no estator aplica-se uma tensão alternada trifásica, logo um campo magnético girante será produzido. Estes dois campos magnéticos irão interagir de forma que o campo magnético fixo acompanha sincronizadamente o campo magnético girante fazendo que o rotor gire na mesma velocidade do campo magnético girante (JÚNIOR, 2011).

Já o motor assíncrono possui um enrolamento trifásico no estator que também cria um campo magnético girante, mas a tensão aplicada em seu rotor é induzida pelo campo girante. Portanto, o motor trifásico assíncrono tem seu funcionamento dependente do campo girante (JÚNIOR, 2011). O campo girante ocorre a partir de uma aplicação de uma tensão trifásica no enrolamento do estator que irá produzir um campo magnético conjunto trifásico de correntes circulando, sendo que essas correntes irão produzir um campo magnético que está girando a uma velocidade dada pela equação (1), o campo magnético produz uma força eletromotriz capaz de produzir um torque no rotor (CHAPMAN, 2013).

$$n_{sinc} = (120 f_{se})/P \tag{1}$$

Onde P é quantidade de polos que o motor possui e f_{se} é frequência de tensão em hertz(Hz). As tensões que percorrem os três enrolamentos do motor são defasadas em 120 graus demonstrada na Figura 4.

Figura 4 – Defasagem das tensões no motor elétrico



Fonte: Autoria própria

O seu rotor nunca poderá girar na mesma velocidade que o campo magnético girante, se isso ocorrer as barras do rotor estariam em um estado de repouso em relação a este campo, logo o motor não funcionará. Assim, a velocidade de giro do rotor de um motor assíncrono nunca deverá ser a velocidade síncrona, mas próxima a ela. A diferença entre a velocidade do

rotor e o campo magnético girante é definida por velocidade de escorregamento e sua relação é dada pela equação 2. (CHAPMAN, 2013).

$$n_{esc} = (n_{sinc} - n_m) \tag{2}$$

Onde n_{esc} é a velocidade de escorregamento, n_{sinc} é a velocidade do campo magnético girante e n_m é a velocidade do eixo do motor. Logo pode ser definido o escorregamento s como a porcentagem da velocidade de escorregamento pela velocidade dos campos magnéticos. A equação (3) representa isto (CHAPMAN, 2013).

$$s = (n_{nesc}/n_{sinc})100 \tag{3}$$

É possível expressar a equação (3) por meio das velocidades angulares como mostra a equação (4) (CHAPMAN, 2013).

$$s = (w_{sinc} - w_m/w_{sinc})100 \tag{4}$$

No qual w_{sinc} é a velocidade angular do campo magnético girante e w_m é a velocidade angular do eixo do motor. O motor elétrico dedicado a este projeto é do fabricante WEG, sendo este trifásico com potência de 0,33 CV, com rotação nominal de 1730 rpm, e pode ser ligado em 220 ou 380V, como apresentado na Figura 5. Este motor fornecerá a rotação para o disco por meio de um conjunto de polias ligadas ao eixo do disco.

Figura 5 – Motor instalado no tribômetro

Fonte: Autoria própria

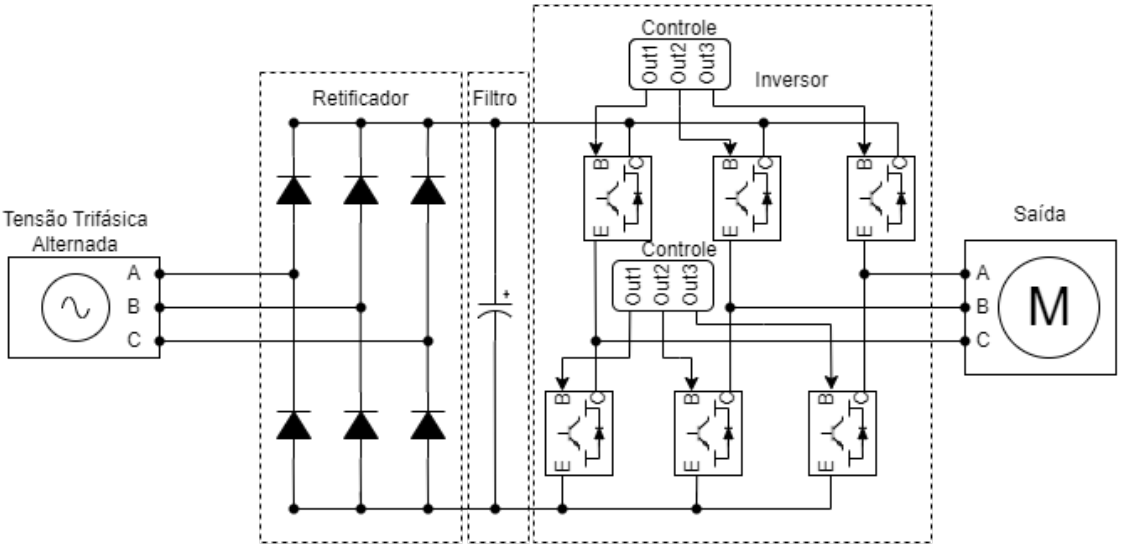
2.1.2 Inversor de frequência

Uma das formas mais utilizadas para variar a velocidade de um motor assíncrono é variar a frequência das fases da tensão de alimentação. Pode-se dizer que o inversor de

frequência converte a tensão e frequência de entrada para uma tensão e frequência variável para controlar motores de indução Vca. Ele consiste em dispositivos eletrônicos de potência IGBT ou MOSFET, unidade de controle de velocidade como um microprocessador (FRANCHI, 2013).

As duas principais características do conversor de frequência são as velocidades ajustáveis e os recursos de partida/parada suave. Esses dois recursos tornam o inversor um controlador para controlar os motores. O inversor de frequência consiste principalmente em quatro seções, Figura 6, compreendidas em: retificador, filtro, inversor e circuito de controle:

Figura 6 – Componentes de um inversor de frequência

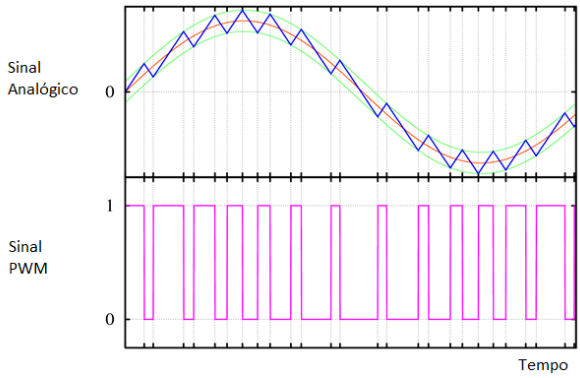


Fonte: Autoria própria

- Retificador - É o primeiro estágio de um inversor de frequência, onde é convertida a tensão de corrente alternada da rede elétrica em tensão corrente contínua, podendo serem empregados os componentes eletrônicos, tais como: diodos, SCRs, transistores e outros dispositivos de comutação eletrônica que possam retificar a corrente.
- Filtro - A tensão de corrente contínua vindo do retificador é entregue ao filtro, que consiste em capacitores e indutores destinados absorver as variações de tensão do circuito anterior, linearizando máximo possível as ondulações e perturbações que possam ocorrer.
- Inversor - Esta seção é composta por chaves eletrônicas de potência como transistores, tiristores, IGBT, MOSFETs. Esses componentes recebem a tensão de corrente conínua dos filtros e converte em tensão de corrente alternada que é fornecida ao motor. Usando técnicas de modulação como modulação por largura de pulso (*PWM*) para variar a frequência de saída para controlar a velocidade do motor de indução, Figura 7.
- Controle - Consiste em uma unidade microprocessada que executa várias funções como controle, ajuste de configurações do inversor, condições de falha e interfaces de comunicação utilizando protocolos industriais. Esta unidade pode receber um sinal de *feedback* do motor como referência de velocidade atual e, consequentemente, regular a relação

entre tensão e frequência para controlar a velocidade do motor, ou controlar a velocidade em malha aberta dispensando o uso sensor de *feedback* como foi aplicada nesse projeto.

Figura 7 – Sinal de entrada *versus* sinal *PWM*



Fonte: Autoria própria

Após a contextualização do funcionamento do inversor de frequência, é apresentado o equipamento empregado modelo WEG CFW08, Figura 8.

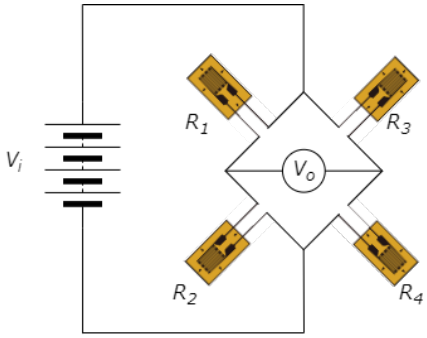
Figura 8 – Inversor de frequência instalado no tribômetro

Fonte: Autoria própria

2.1.3 Célula de carga

A célula de carga é um sensor usado para medir força. Nesse trabalho, foi usado o sensor baseado em deformação do material que altera a resistência dos extensômetros simbolizados $R1, R2, R3$ e $R4$, ligados no formato de ponte *Wheatstone* (DALLY; RILEY; MCCONNELL, 1993). O circuito de ponte *Wheatstone* pode ser usado de várias maneiras tanto para medir resistência elétrica quanto para determinar do valor absoluto de uma resistência por comparação com uma resistência conhecida, e a determinação de alterações diminutas na resistência. O último método é aplicado para o sensor *Strain Gage*. Ele permite que mudanças relativas de resistência possam ser medidas com grande precisão. A Figura 9 demonstra o funcionamento deste circuito esta baseado na variação das resistências internas e pode ser observado de acordo com a equação (5) abaixo:

Figura 9 – Esquema elétrico de uma ponte de *Wheatstone* utilizando extensômetros



Fonte: Autoria própria

$$V_o = V_i((R_1R_3 + R_2R_4)/(R_1 + R_2)(R_3 + R_4)) \tag{5}$$

Quando os valores das resistências obedecem a equação (6):

$$(R_1R_3) = (R_2R_4) \tag{6}$$

Ou os valores das resistências dos extensômetros forem todos iguais, isto é, $R1 = R2 = R3 = R4$, o valor de $V_o = 0$, diz-se então que a ponte está equilibrada ou em balanço. Caso contrário, se houver um desequilíbrio entre as cargas resistivas estará ocorrendo algum tipo de deformação no material.

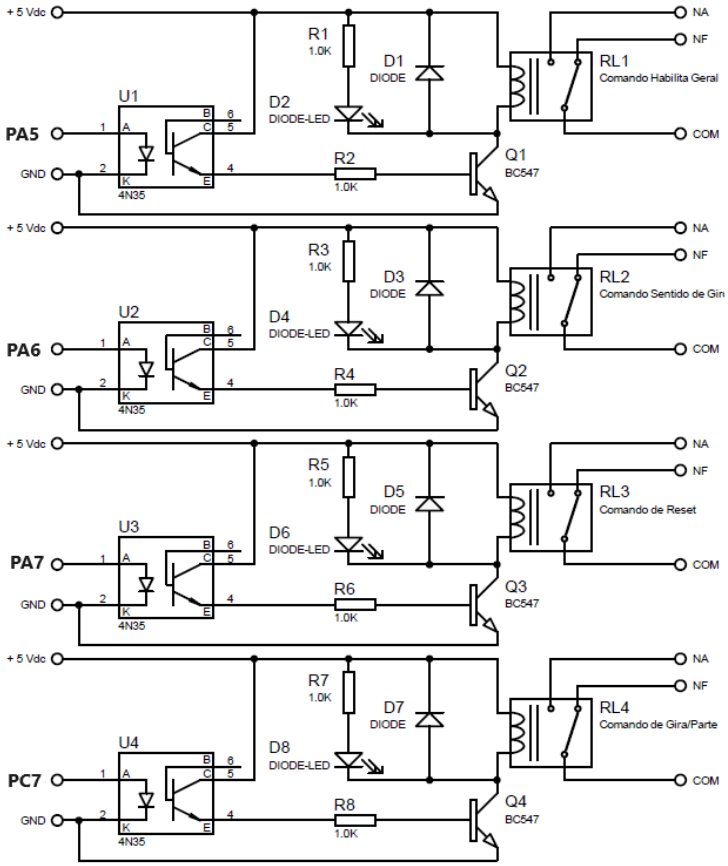
2.2 Sensores e componentes eletrônicos

Os componentes instalados foram selecionados de acordo com cada aplicação, por exemplo, microcontrolador foi utilizado a placa de desenvolvimento NUCLEO-F446RE para o controle e automatização, a placa SD *Card* para *backup* dos dados lidos da célula de carga, o módulo conversor serial USB/RS232-TTL para os enlaces entre PC e célula de carga e a microcontrolador. Sensores de temperatura e rotação para acompanhamentos dessas variáveis, *display* LCD para indicação local, um módulo de quatro relés para enviar comandos discretos ao inversor de frequência e um conversor de tensão para corrente aplicado ao controle de rotação.

2.2.1 Microcontrolador

O microcontrolador escolhido para esse projeto foi baseado na tecnologia ARM, modelo STM32F446RE montado em uma placa de desenvolvimento NUCLEO-F446RE, em que explorou-se os recursos de *hardware* e *software* utilizando as funcionalidades: GPIO, RTC, TIM, UART, e SPI.

Figura 10 – Esquema elétrico placa *four relay module*



Fonte: Autoria própria

2.2.1.1 GPIO - *General Purpose Input and Output*

Nome dado aos terminais que podem assumir a função de entradas ou saídas, com uso livre pelo projetista conforme sua necessidade. Os terminais do microncontrolador como os GPIO podem assumir funções especiais de acordo com as caracteristicas que estão apresentadas no *datasheet* do microcontrolador, assim demonstradas nos subitens abaixo:

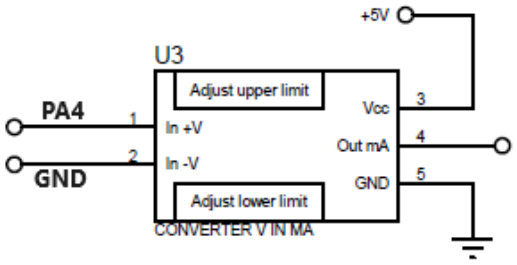
- Saídas digitais - As saídas digitais foram usadas para comandar o inversor remotamente por meio do enlace serial entre PC e microcontrolador, disponibilizado na tela do computador quatro comandos discretos como: habilitar o sistema, inverter o sentido de rotação do motor, reset do inversor e partida/parada do equipamento (WEG,2008). Para isso, dentro do programa do microcontrolador, foi configurado um vetor que monitora os comandos provenientes da interface homem máquina e repassa os seus estados lógicos aos registradores GPIO PA5, PA6, PA7 e PC4, que são os pinos de saídas e estão ligados a placa de relés, Figura 10. Assim enviando o comando desejado ao equipamento.
- DAC - *Digital-to-Analog Converter* ou conversor digital para analágico, é um circuito eletrônico cuja função é converter uma grandeza digital, por exemplo uma sequência de *bits*, em uma grandeza analógica, esta grandeza analógica pode ser dada em tensão ou corrente. No caso de sistemas microcontrolados, a sua saída é dada de 0 a $3{,}3V$ é a

resuloção segundo o fabricante é de 12 *bits*, o pino usado foi o PA4, já que é o único pino disponível nessa versão de microcontrolador, portanto o sinal de saída V_{out} pode ser dado pela multiplicação da tensão de referência $V_{ref} = 3{,}3V$ pelo registador de saída D_{or}, divido pela resolução máxima em *bits* (4095 *bits*). Assim expressa pela equação (7).

$$V_{out} = V_{ref}(D_{or}/4095) \tag{7}$$

De acordo com o resultado da equação (7) um nível de tensão será fornecido ao conversor de tensão para corrente, Figura 11, a utilização desse conversor foi necessária devida a entrada analógica do inversor de frequência ser configurada para um *loop* de corrente (4 a 20 mA).

Figura 11 – Esquema elétrico conversor tensão x corrente



Fonte: Autoria própria

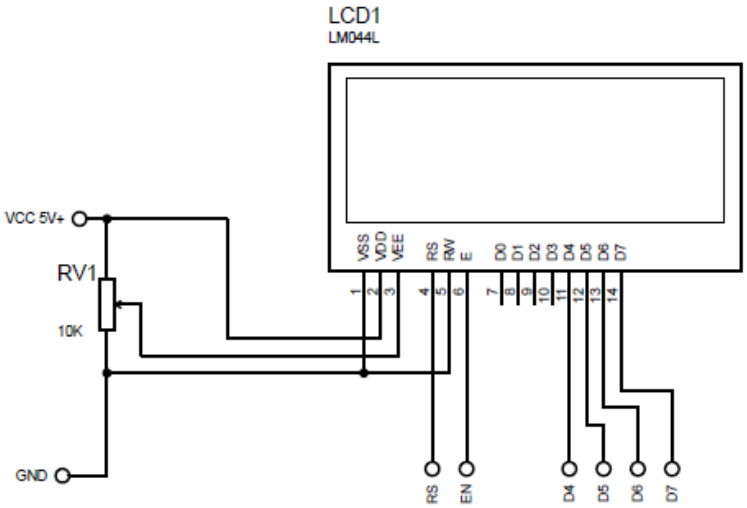
2.2.2 *Display* LCD

Um *display* é um dispositivo optoeletrônico que pode mostrar números, dígitos hexadecimais ou qualquer combinação de letras, números e sinais de pontuação, ou gráfico que pode ser representado por uma matriz de pontos (HOROWITZ; HILL, 2017). Nesse projeto, optou-se pelo *display* LCD de 20x4, ou seja, 4 linhas por 20 colunas contendo informações sobre a rotação medida no eixo, temperatura local, status do conjunto e o tempo gasto durante o experimento. As definições de *software* bem como a ligação do *hardware* entre o *display* e microcontrolador adotada foi para uma conexão de 4 *bits* de dados configurados na biblioteca LCD.h com os pinos de saída, utilizando os pinos D4, D5, D6 e D7 para recepção dos dados, RW colocado em nível baixo já que este *display* efetua somente leitura, conforme Figura 12.

2.2.3 Termistor NTC 10kΩ

Este componente é dedicado a medição de temperatura durante a execução do experimento, o termistor NTC (*Negative Temperature Coefficient*) é definido como um resistor sensível a variação de temperatura, ou seja, a resistência apresentada entre seus terminais está relacionada com a temperatura local que o componente está submetido (BOYLESTAD; NASHELSKY, 2013). No caso do NTC a resistência diminui conforme a temperatura aumenta,

Figura 12 – Esquema elétrico *display* LCD



Fonte: Autoria própria

e sua temperatura pode ser determinada usando a relação de *Steinhart-Hart* , que é dada pela equação (8).

$$1/T = A + B\ln(R_{ntc}) + C(\ln(R_{ntc}))^3 \tag{8}$$

Na equação acima, a variável R é o valor da resistência do NTC medido em Ω, já a variável T e temperatura em graus K (*Kelvin*). Logo para utilizar esta relação no microcontrolador, é necessário a determinação dos os coeficientes A, B e C, estes coeficientes podem serem calculados de acordo com a Tabela 1 de resistência à temperatura do componente NTC 10kΩ.

Tabela 1 – Tabela de resistência NTC 10kΩ

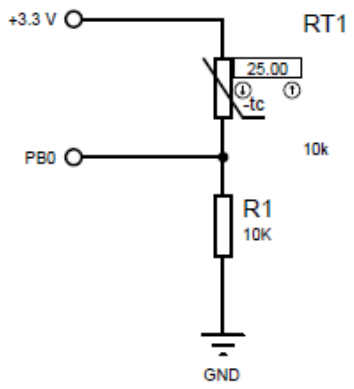
Temperatura T °C	Resistência R Ω
0,0	35563,0
25,0	10000,0
100,0	549,4

Escolhendo três diferentes temperaturas de medição na Tabela 1 e escrevendo os valores de resistência correspondentes as suas temperaturas na equação (8), obtêm-se 3 equações com três incógnitas. Portanto os coeficientes A, B e C poderam ser encontrados e incorporados no microcontrolador.

Para a conexão entre STM32F446RE e o sensor NTC 10kΩ foi usado o pino PB0 que corresponde ao conversor ADC 1 do microcontrolador, juntamente com um resistor de 10kΩ para fazer um circuito *pull-down* de acordo com o esquema elétrico, Figura 13.

Ao observar a equação (8) é necessário o conhecimento da resistência do sensor de temperatura para calcular o valor da temperatura. Aplicando o conceito do divisor resistivo

Figura 13 – Esquema elétrico do sensor de temperatura



Fonte: Autoria própria

no circuito do sensor de temperatura, a tensão em V_{PB0} pode ser descrita da seguinte forma, equação (9).

$$V_{PB0} = 3{,}3(10000/(10000 + R_{ntc})) \tag{9}$$

Mas V_{PB0} também pode ser escrito como um divisor de tensão, para isso deve ser considerado a tensão de referência de 3,3V e dividir pela resolução de 12 *bits* do ADC (4095), e multiplicado pelo valor em tensão lido no ADC V_{ADC}, que é escrito pela equação (10).

$$V_{PB0} = V_{ADC}(3{,}3/4095) \tag{10}$$

Igualando as equações (9) e (10), de forma a reescrever em função da resistência R_{ntc}, onde é encontrada a seguinte relação:

$$R_{ntc} = (40950000/V_{ADC}) - 10000 \tag{11}$$

Após determinação da resistência R_{ntc}, este valor deverá ser aplicado novamente na equação (8) para a determinação da temperatura T.

2.2.4 *Enconder*

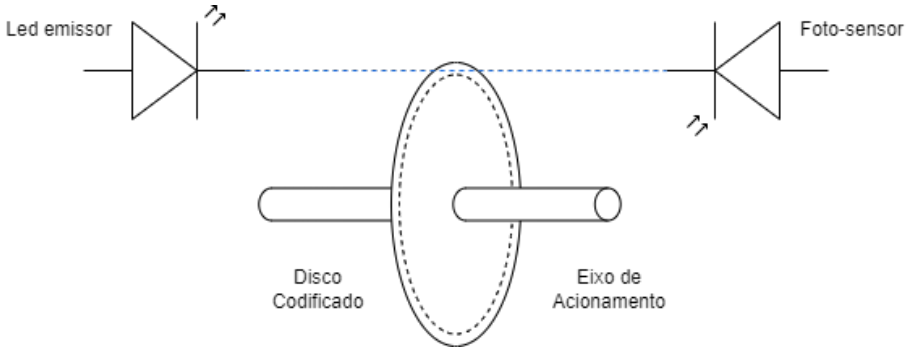
Conhecidos como transdutores de posição angular os *encoders* são dispositivos eletromecânicos capazes de converter movimentos lineares ou angulares em pulsos elétricos. A maioria dos encoders são do tipo ópticos, ou seja, utilizam um par transmissor receptor óptico para a produção dos pulsos digitais como saída (detecção fotoelétrica) (POP, 2022). Quanto à diversidade pode-se classificá-los em encoder regular, encoder regular defasado (ou incremental) e os encoders absolutos. Os encoders regulares são os modelos mais simples, enviando à saída um trem de pulsos digitais toda vez que o eixo do encoder gira. A partir da contagem de pulsos

do sinal de saída pode-se determinar a velocidade do eixo ou ainda a posição do mesmo a partir de um referencial.

Já os encoders incrementais possuem duas faixas de áreas claras e escuras defasadas de modo que os sinais de saída gerados também são defasados. Sua vantagem é que, além de ser possível determinar a velocidade de um eixo, também é possível determinar o sentido de rotação. O último tipo são os encoders absolutos capazes de fornecer informações sobre velocidade, sentido de rotação e posição real do eixo girante. Eles são dotados de um circuito de codificação em binário onde todas as possíveis posições de parada do encoder tem um código único. Dessa forma é possível saber o ponto exato onde o eixo está posicionado bem como o curso final do mesmo (POP, 2022).

Basicamente os encoders ópticos são constituídos por um disco codificado acoplado à um eixo, um ou mais transmissores e um ou mais receptores como mostra a Figura 14. O disco codificado é feito de plástico transparente onde estão gravadas tiras escuras que correspondem à codificação digital de cada posição. A leitura do movimento do disco codificado é feita por um par transmissor-receptor acoplado de forma que o disco fique entre esse conjunto. Assim as partes claras e escuras passam diante do par transmissor-receptor, hora impedindo que o seu sinal chegue ao receptor hora permitindo a passagem desse sinal.

Figura 14 – Montagem básica de *encoder* fotoelétrico

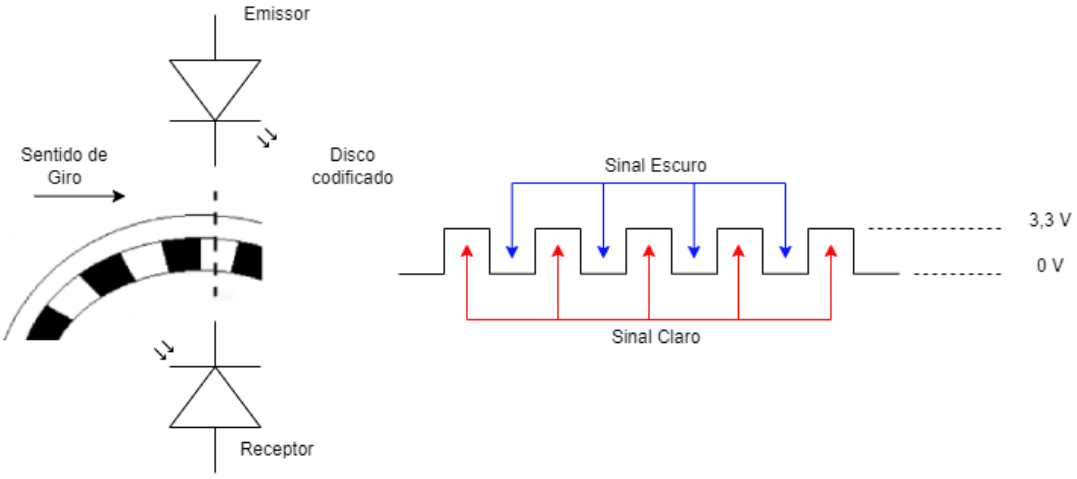


Fonte: Autoria própria

O eixo do encoder é acoplado à alguma peça móvel do equipamento que se deseja monitorar seu movimento (tal como um eixo de motor, por exemplo). A movimentação do eixo é convertida em um trem de pulsos elétricos através da detecção fotoelétrica, onde uma série de pulsos são gerados pela passagem de luz ou sinal infravermelho pelas "partes claras"do disco codificado. O receptor detecta o sinal enviado pelo emissor e também a ausência desse sinal gerando assim pulsos digitais (0 e 1) como mostra a Figura 15. Os pulsos gerados se assemelham à uma onda quadrada e normalmente sua amplitude varia entre 0 e 3,3V.

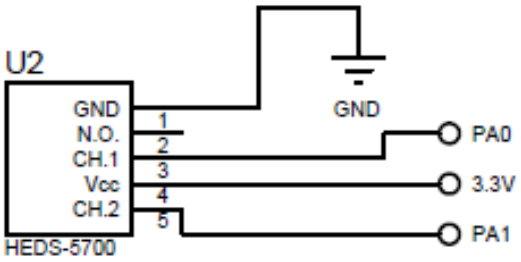
O sinal de saída dos encoders ópticos incrementais deve ser convertido em informação, para realizar essa tarefa foi necessário a conexão com o *encoder HEDS-5701*, alimentado com 3,3 V e ligados aos pinos PA0 e PA1 do microcontrolador para as leituras dos canais 1 e 2 segundo a Figura 16.

Figura 15 – Funcionamento de um *encoder*



Fonte: Autoria própria

Figura 16 – Esquema elétrico de ligação do *encoder*



Fonte: Autoria própria

Lembrando que os pinos PA0 e PA1 foram configurados de acordo com o *Timer 5* canal 1 e 2 para devidos tratamentos dos sinais capturados.

2.2.5 UART – RS 232

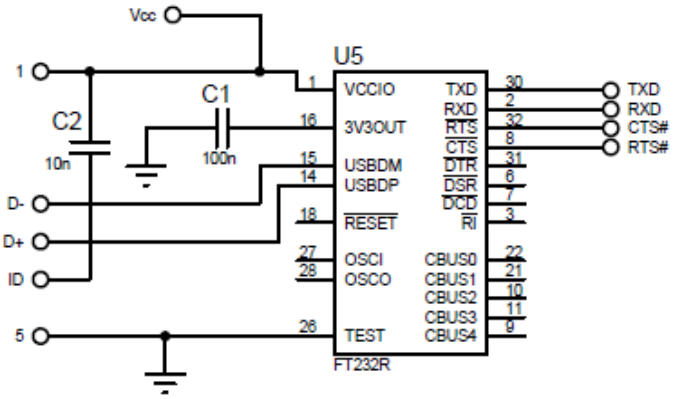
Este é um formato de sinalização que remonta à década de 1960 (HOROWITZ; HILL, 2017), destinado originalmente para enlaces seriais de baixa velocidade (menor que 19,2 k*bps*) entre terminais alfanuméricos. Os dados normalmente são enviados em quadro de bytes seriais contento 8 *bits*, com um bit de início e um ou dois *bits* de fim. Os *bits* extras permitem a sincronização entre receptor e emissor. E as taxas de velocidade de rede são múltiplos 300 *bps* TIA -232.

Este projeto utiliza duas portas de comunicação serial do microcontrolador, ambas foram configuradas com a velocidade em 9600 *bps*, 8 *bits* de dados, sem paridade, 1 *bit* de parada e sem nenhum controle de fluxo de comunicação.

Para o enlace entre o microcontrolador e PC foram utilizados os pinos PA0 e PA1

do STM32F446RE e identificado pela porta *UART4*. O circuito eletrônico empregado foi composto pelo componente FT232R representado pelo esquema elétrico, Figura 17, que realiza a conversão de nível *TTL* para o padrão USB.

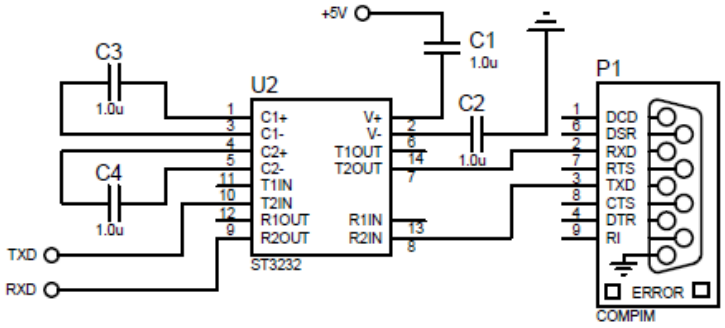
Figura 17 – Esquema elétrico placa para interface UART/USB



Fonte: Autoria própria

O segundo enlace foi dedicado entre o microcontrolador e indicador de pesagem WT21. Este canal de comunicação foi montado através dos pinos PC7 e PC8 do STM32F446RE simbolizado pela porta *UART6* e o circuito eletrônico usado foi representado pelo componente MAX232 e exemplificado no esquema elétrico, Figura 18.

Figura 18 – Esquema elétrico do conversor UART/Serial



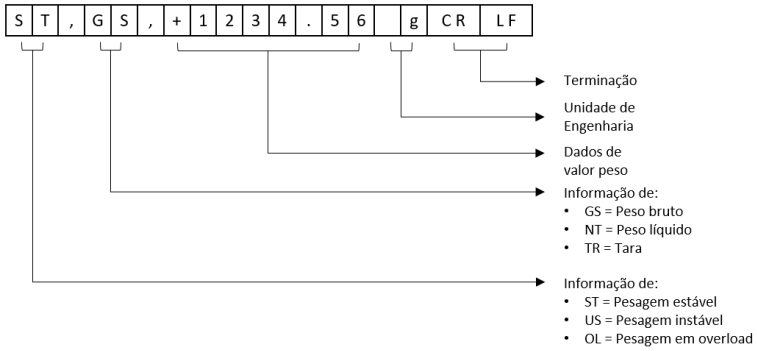
Fonte: Autoria própria

Basicamente o enlace com indicador de pesagem WT21 obedece a um protocolo definido pelo fabricante *Weightech*, onde especifica o *frame* de cada mensagem que é composto por quadros de 18 bytes de acordo com a Figura 19.

Do primeiro a terceiro *byte* se referem ao tipo de pesagem se estável ou instável, do quarto ao sexto é a informação sobre o peso, líquido ou bruto, do sétimo ao décimo quarto é o valor do peso, décimo quinto ao décimo sexto é a unidade de engenharia, e os dois últimos *bytes* são definidos como finalização da mensagem.

Após a leitura do frame supracitado o microntrolador separa o *byte* responsável pelo peso líquido e concatena o *time stamp* gerado pelo RTC interno, envia uma *string* para o PC

Figura 19 – Protocolo do indicador de pesagem WT21



Fonte: Autoria própria

contendo o dado de pesagem mais o horário da amostragem, essa informação será armazenada para fim de estudo e análise do experimento tribológico.

2.2.6 SD *Card*

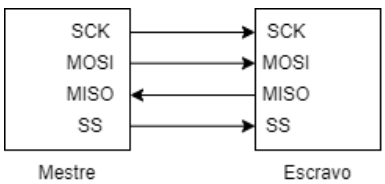
Para a utilização do cartão de memória foi empregado o protocolo SPI, que é uma interface serial amplamente utilizada para se comunicar com dispositivos de *hardwares* comuns, incluindo *displays*, cartões de memória e sensores. Os processadores STM32 tem várias interfaces SPI e cada uma dessas interfaces pode se comunicar com vários dispositivos (BROWN, 2012).

A interface SPI é um tipo de comunicação *full duplex* (envia e recebe os dados ao mesmo tempo), para que isso ocorra são exigidas quatro vias para seu funcionamento, de acordo com a Figura 20. Esta interface possui uma linha de *clock*, ou seja, sua comunicação é síncrona entre o mestre e escravo, as nomenclaturas para esta interface são padronizadas e mostradas na Tabela 2,

Tabela 2 – Vias de comunicação SPI

Nome Padrão	Significado	Função
MOSI	*Master Output Slave Input*	Mestre para escravo
MISO	*Master Input Slave Output*	Escravo para mestre
SCK	*Serial Clock*	Temporização
SS	*Slave Selec*	Seleção do escravo

Figura 20 – Troca de informações entre mestre e escravo



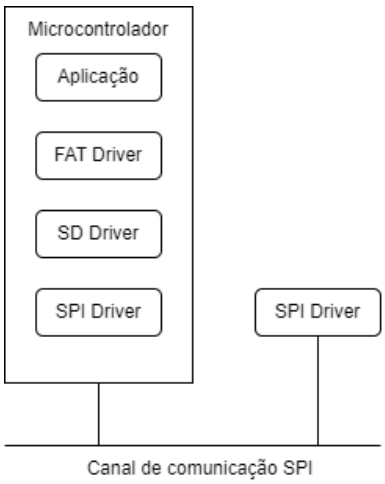
Fonte: Autoria própria

A seleção de um escravo ocorre quando a tensão da via *SS* cai ao nível *low*, quando nível volta para *high*, o escravo fica ocioso. Esta via também pode ser utilizada para o sincronismo, marcando o início e o final da transferência do pacote de dados (BROWN, 2012).

Quando o pino *SS* é mantido em nível *high*, a saída do escravo, isto é o pino MISO, é colocado em alta impedância para não interferir na transmissão de outro escravo. A troca de dados acontece nas duas direções, isto é, se o mestre enviar um *bit* ele precisa receber um *bit* do escravo, isso é uma característica da interface SPI, pois há um registrador de deslocamento, baseado na técnica *shift register*, que é uma cascata de *flip flops* ligadas no mesmo sinal de *clock*, onde cada *flip flops* é ligado em série. Assim o mestre recebe os dados armazenados no escravo completando o ciclo de troca de informações.

Os dados em um SD *card* são organizados como um sistema de arquivos FAT32. Os primeiros blocos de armazenamento são usados para manter os dados sobre o sistema de arquivos, por exemplo em tabelas de alocação, enquanto os blocos restantes são usados para armazenar o conteúdo de arquivos e diretórios. Considere a Figura 21, que mostra a pilha de *software* necessária. Como a aplicação necessita acessar os dados no SD *card*, utiliza comandos de nível de arquivo, como *open*, *read*, e *write* para acessar arquivos específicos dentro do sistema de arquivos do SD *card*. Estes comandos são fornecidos pelo *driver* do sistema de arquivos FAT. Este sistema emite comandos ao nível de leitura e escrita de blocos, logo o *driver* SD separado implementa estes comandos. Finalmente, o *driver* do SD repassa os comandos desejados a interface SPI para se comunicar com o SD *card*.

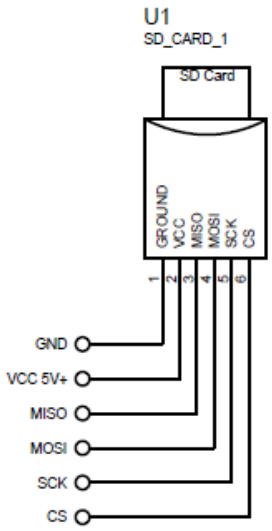
Figura 21 – Pilha de *software* SD *Card*



Fonte: Autoria própria

Assim sendo a placa SD *card* foi conectada aos pinos do microcontrolador da seguinte forma, *MOSI* pino PC1, *MISO* pino PC2, *SCK* pino PB13 e *CS* pino PB14, e demonstrada na Figura 22.

Figura 22 – Esquema elétrico do *software* SD *Card*



Fonte: Autoria própria

2.3 Elaboração de *firmware* e *software* de aplicação

Para a confecção do *firmware* do microcontrolador foi utilizado dois *softwares* de edição, o primeiro denominado STM32CubeMX e o segundo STM32CubeIDE, que é um ambiente de desenvolvimento integrado (IDE) desenvolvido pela STMicroelectronics (STMICROELECTRONICS, 2017).

2.3.1 STM32CubeMX

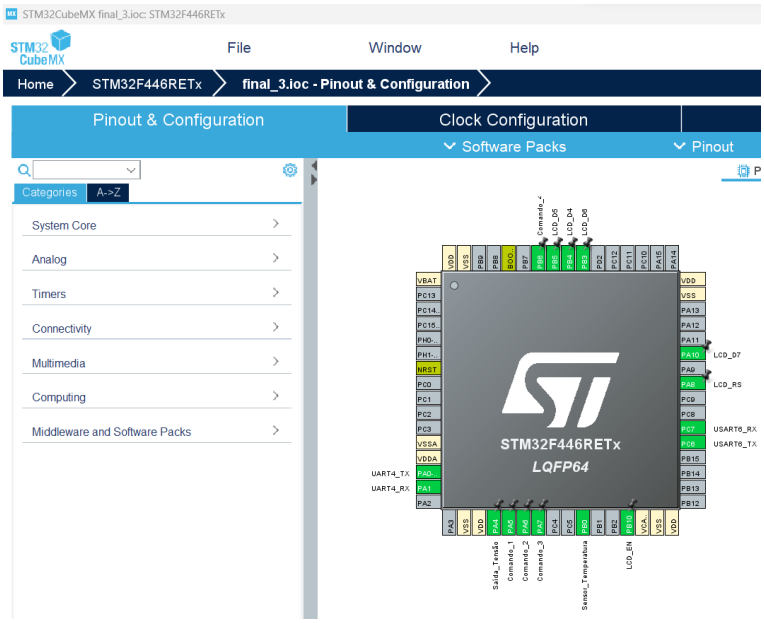
Nesta IDE é possível configurar de forma gráfica todos os pinos de entrada/saída, temporizadores, conectividade, e pacotes de drivers que pode ser associado ao microcontrolador. Este *software* gera o código de configuração em linguagem C da plataforma escolhida pelo desenvolvedor (STMICROELECTRONICS, 2017).

Ao abrir o programa a primeira aba a ser vista é *Pinout* & *Configuration*, Figura 23, onde cada pino utilizado do microcontrolador é destacado na cor verde e sua função é atribuida após a compilação do projeto.

Prosseguindo na aba o *Clock Configuration*, será determinada a frequência de operação do microcontrolador, sabendo que o STM32 tem uma rede de distribuição de *clock* complexa que garante que somente os periféricos que são utilizados na aplicação serão utilizados, Figura 24. Esse artificio é chamado de *Reset and Clock Control (RCC)* é controlado pelo módulo de firmware stm32f4xx_rcc.h.

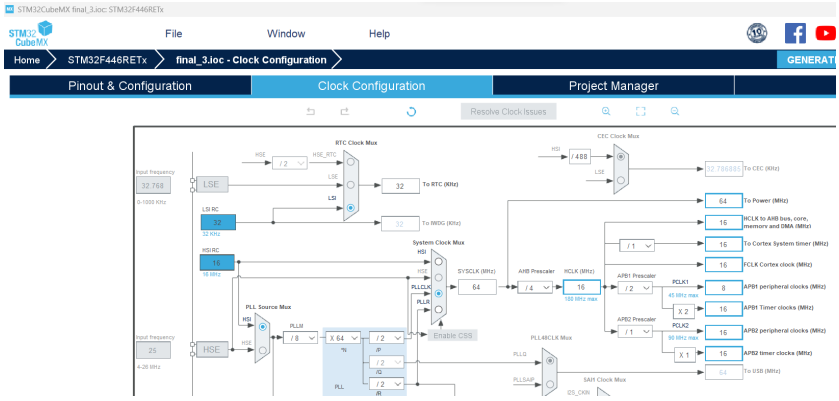
Ao habilitar o *clock* dos periféricos é necessário o conhecimento dos dispositivos, onde foram organizados em três grupos diferentes chamados APB1, APB2 e AHB. Periféricos APB1 incluem os dispositivos I2C, USARTs, e dispositivos SPI; dispositivos APB2 incluem as portas GPIO, controladores ADC e a USART. Dispositivos AHB são primariamente orientados a memória incluindo os controladores DMA e interfaces de memória externa (para alguns

Figura 23 – Aba*Pinout & Configuration* STMCubeMX



Fonte: Autoria própria

Figura 24 – Aba Configuração do *Clock* STMCubeMX



Fonte: Autoria própria

dispositivos). Para diversos periféricos o *clock* pode ser controlado com três rotinas de *firmware*::

```
1 RCC_APB1PeriphClockCmd(uint32_t RCC_APB1PERIPH , FunctionalState NewState)
2 RCC_APB2PeriphClockCmd(uint32_t RCC_APB2PERIPH , FunctionalState NewState)
3 RCC_AHBPeriphClockCmduint32_t RCC_AHBPERIPH , FunctionalState NewState)
```

Cada rotina pode receber dois parâmetros(*ENABLE* ou *DISABLE)*, por meio de um vetor de *bits* do periférico a qual o estado será modificado. Por exemplo, as portas do GPIO A e B podem ser habilitadas com o seguinte chamado:

```
1 RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOA |
2 RCC_APB2Periph_GPIOB , ENABLE);
```

As constantes apropriadas são definidas na biblioteca stm32f4xx_rcc.h; o nome dos dispositivos e periféricos são demonstrada nas Tabelas 3, 4 e 5. Vale lembrar que as características dos dispositvos estão detalhadas no manual do STM32F446RE.

Tabela 3 – Dispositivo APB1 e seus periféricos

Dispositivo	Periféricos
$APB1$	$RCC\_APB1Periph_BKP$
	$RCC\_APB1Periph_CEC$
	$RCC\_APB1Periph_DAC$
	$RCC\_APB1Periph_I2C1$
	$RCC\_APB1Periph_I2C2$
	$RCC\_APB1Periph_PWR$
	$RCC\_APB1Periph_SPI2$
	$RCC\_APB1Periph_TIM2$
	$RCC\_APB1Periph_TIM3$
	$RCC\_APB1Periph_TIM4$
	$RCC\_APB1Periph_TIM5$
	$RCC\_APB1Periph_TIM6$
	$RCC\_APB1Periph_TIM7$
	$RCC\_APB1Periph_USART2$
	$RCC\_APB1Periph_USART3$
	$RCC\_APB1Periph_WWDG$

Tabela 4 – Dispositivo APB2 e seus periféricos

Dispositivo	Periféricos
$APB2$	$RCC\_APB2Periph_ADC1$
	$RCC\_APB2Periph_AFIO$
	$RCC\_APB2Periph_GPIOA$
	$RCC\_APB2Periph_GPIOB$
	$RCC\_APB2Periph_GPIOC$
	$RCC\_APB2Periph_GPIOD$
	$RCC\_APB2Periph_GPIOE$
	$RCC\_APB2Periph_SPI1$
	$RCC\_APB2Periph_TIM1$
	$RCC\_APB2Periph_TIM15$
	$RCC\_APB2Periph_TIM16$
	$RCC\_APB2Periph_TIM17$
	$RCC\_APB2Periph_USART1$

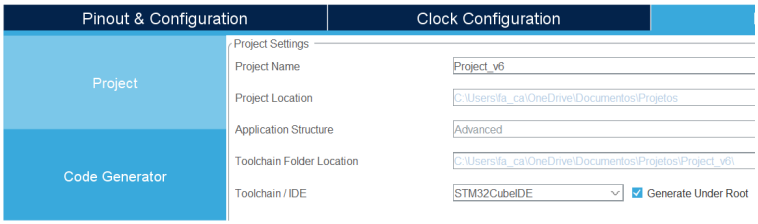
Tabela 5 – Dispositivo AHB e seus periféricos

Dispositivo	Periféricos
AHB	$RCC\_AHBPeriph_CRC$
	$RCC\_AHBPeriph_DMA$

Neste projeto foram empregados todos os dispositivos presentes e deixados com o *clock* de 50 Mhz padrão para que haja atualização de escrita e leitura adequada de todos os registradores do programa.

E na aba *Project Manager* o projeto será salvo com nome atribuído ao arquivo compilado, e o mais importante, é a atribuição da IDE que será aberto para a configuração dos dispositivos que foram escolhidos no STMCubeMX finalizando o procedimento neste *software*.

Figura 25 – Configuração do *Project Manager*



Fonte: Autoria própria

2.3.2 STM32CubeIDE

É a plataforma de desenvolvimento C/C++ avançada com configuração periférica, geração de código, compilação de código e recursos de depuração para microcontroladores e microprocessadores STM32. Ele é baseado na estrutura Eclipse /CDT e na cadeia de ferramentas de compilação *GNU Compiler Collection* (GCC) para o desenvolvimento e *debugger* - GNU *Project Debugger* (GDB) para a depuração. Permite a integração das centenas de *plugins* existentes que completam as funcionalidades do Eclipse IDE. O STM32CubeIDE integra a configuração do STM32 e as funcionalidades do STM32CubeMX. O STM32CubeIDE inclui ferramentas de análise de compilação e uma pilha que fornece informações sobre o status do projeto e os requisitos de memória. Estes, por sua vez, incluem os recursos: de depuração padrão e avançado; visualizações de registros da CPU; memórias e registros periféricos; observação de variável *on line*; e, interface *Serial Wire Viewer* ou analisador de falhas. Como descrito na seção de componentes eletrônicos, a placa usada foi a NUCLEO-F446RE, o *firmware* desenvolvido para essa placa foi concebido para realização de tarefas relativas ao tribômetro e organizado em rotinas de inicialização, *loop* infinito e de interrupção.

2.3.2.1 Rotina de inicialização

O arquivo startup_stm32f446.c define o vetor de interrupções, que deve ficar localizado na memória a partir do endereço 0. Portanto o código de inicialização foi executado logo após o *reset* e as ações de inicialização incluíram:

- Inicialização de variáveis estáticas (incluindo as globais) com os valores definidos no código.
- Inicialização de variáveis estáticas (incluindo as globais) que não tiveram o valor inicial definido com 0, por exemplo os vetores de caracteres.
- Inicialização da rotina System Init que foi definida como padrão do próprio programa.
- Inicialização da rotina main.c , que geralmente que dá inicio propriamente dito ao arquivo configurado pelo projetista carregando; as bibliotecas declaradas, funções e drivers.

Observação: para as bibliotecas relacionadas com o componente eletrônico *display* LCD, houve a necessidade de customização para esta aplicação nos arquivos, conforme o trecho de código abaixo, contido no arquivo lcd_v1.1.c.

```
1 #define Output_RS_PIN(state) { HAL_GPIO_WritePin(LCD_RS_GPIO_Port, LCD_RS_Pin, state); }
2 #define Output_EN_PIN(state) { HAL_GPIO_WritePin(LCD_EN_GPIO_Port, LCD_EN_Pin, state); }
3 #define Output_D4_PIN(state) { HAL_GPIO_WritePin(LCD_D4_GPIO_Port, LCD_D4_Pin, state); }
4 #define Output_D5_PIN(state) { HAL_GPIO_WritePin(LCD_D5_GPIO_Port, LCD_D5_Pin, state); }
5 #define Output_D6_PIN(state) { HAL_GPIO_WritePin(LCD_D6_GPIO_Port, LCD_D6_Pin, state); }
6 #define Output_D7_PIN(state) { HAL_GPIO_WritePin(LCD_D7_GPIO_Port, LCD_D7_Pin, state); }
```

2.3.2.2 Rotina de *loop* infinito

O *loop* infinito é definido pela repetição do código em questão (citado no Apêndice A), sendo que neste caso o bloco de operações foi executado enquanto uma dada condição for sempre verdadeira. No código desenvolvido essa condição pode ser observado no emprego da sintaxe *while* de acordo com o trecho de código abaixo.

```
1 while (1)
2 {
3 "User Code"
4 }
```

A execução desse *loop* somente será cessado, caso haja uma interrupção externa ou interna, apresentada no fluxograma, Figura 26.

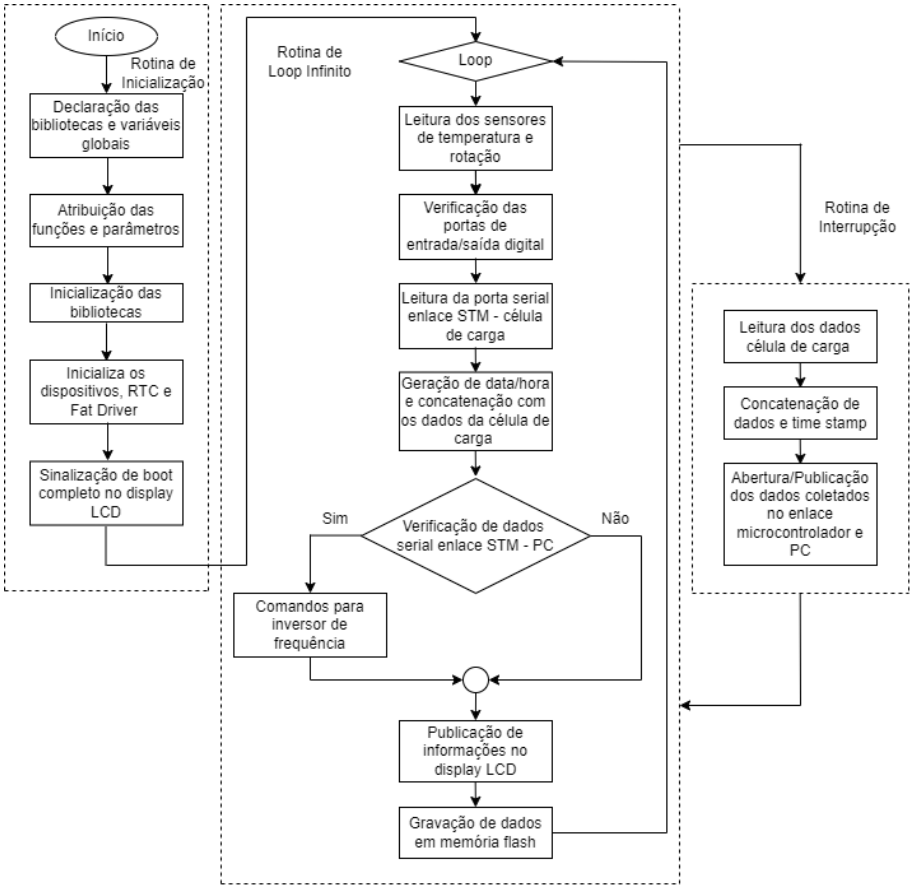
2.3.2.3 Rotina de interrupção

Interrupção é um evento que obriga o microprocessador a suspender suas atividades temporariamente, para atender exclusivamente uma rotina indicada pelo evento que a interrompeu. A interrupção utilizada nesse código foi provocada por um *timer* configurado para atuação em um segundo. Basicamente, essa abre a porta serial entre o PC e microcontrolador, e escreve uma mensagem contendo valor lido da célula de carga concatenado com o *time stamp* gerado pelo RTC, Figura 26.O conteúdo da mensagem é usado para arquivamento no cartão de memória e para o armazenamento de dados no PC.

2.3.3 *Software* de Aplicação

O *software* de aplicação foi desenvolvido com ferramenta MS Visual Studio 2022, que é um ambiente de desenvolvimento integrado (IDE) da Microsoft dedicado ao .NET Framework e linguagem de programação como o C, C++ e o C# (código desenvolvido está no Apêncide B). Nessa aplicação .NET possui uma biblioteca que contem todos os controles necessários para desenvolver as aplicações no Windows, sendo os mais comuns as janelas, campos de entrada, barras de rolagem, botões de opção e modos e exibição.

Figura 26 – Fluxograma de funcionamento do *firmware*



Fonte: Autoria própria

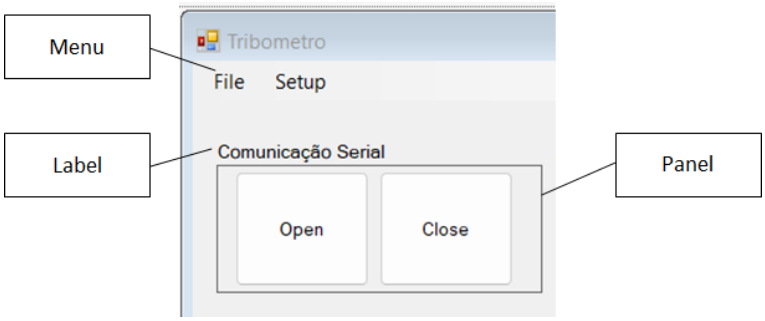
Diante do ensaio do tribômetro, alguns controles foram empregados para o desenvolvimento do aplicativo, tais como:*Button, CheckBox, DataGrid, Forms, Label, MenuStrip, Panel, SerialPort Timer*.

Iniciando a configuração do aplicativo obrigatoriamente deverá conter uma janela principal de interação, este controle é denominado como *Windows.Forms*. E esta é a janela definida como a base de todo o aplicativo desenvolvido. Todos os demais controles supracitados deverão estar inseridos dentro desse formulário. Para o texto estático foi atribuído o controle *Label* utilizado na indicação das opções disponíveis e no caso dos contornos internos, foi utilizado o controle *Panel* para o melhor agrupamento das opções, Figura 27. O *MenuStrip* foi usado para a configuração da porta serial a ser utilizada, Figura 28.

Para a comunicação serial, o controle *SerialPort* foi selecionado, juntamente com o controle *Timer*. Isso se deve aos comandos em C# atribuir a função de abertura da porta COM e sua atualização em milissegundos, de acordo com o trecho de código:

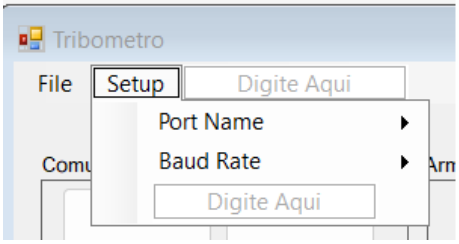
```
1  private void timer1_Tick(object sender, EventArgs e)
2          {
3          // Declaracao de variaveis locais
4              string tempVal;
5               if (serialPort1.IsOpen)
6               {
7                   tempVal = serialPort1.ReadExisting();
```

Figura 27 – Controles *Label* e *Panel*



Fonte: Autoria própria

Figura 28 – Controle *MenuStrip*



Fonte: Autoria própria

```
8                   if (tempVal != "")
9                   {
10                      cont++;
11                      Grid.Rows.Add(tempVal, cont);
12                  }
13              }
```

A publicação dos dados em tela é feita pelo controle *DataGrid*, onde monta os dados lidos pela porta serial em forma de tabela, e esta tabela pode ser armazenada em arquivos .csv através do *checkbox* Backup Automático ou pelo botão Salvar.

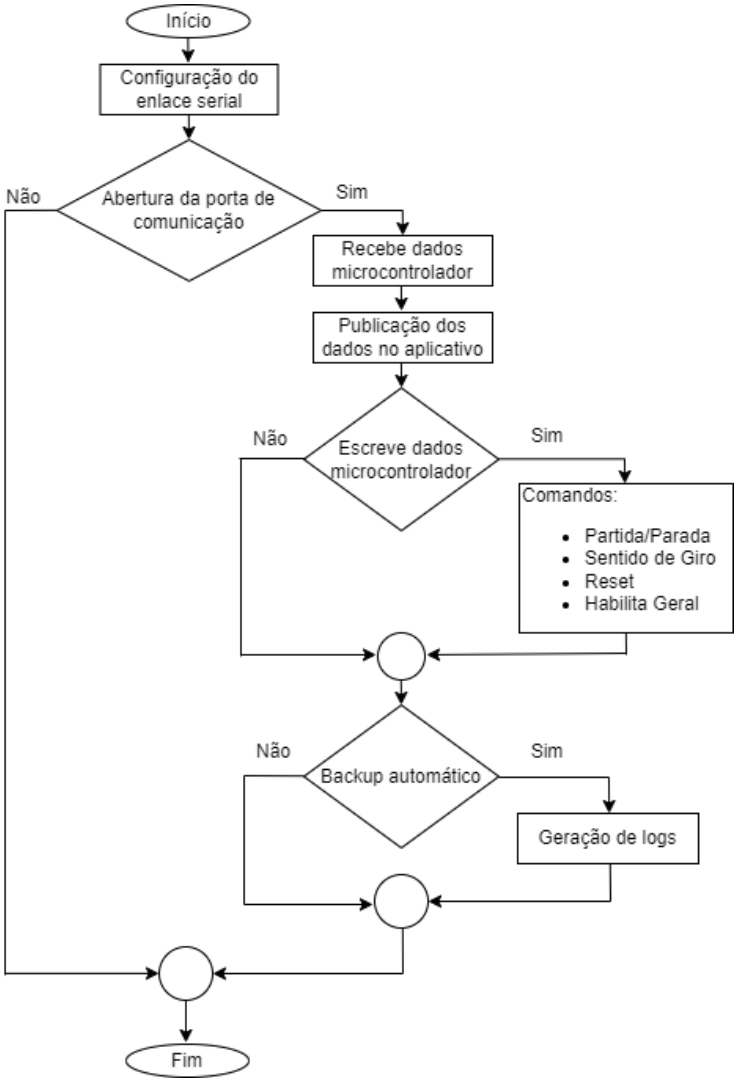
E o controle *Button* que é representado pelos botões, que foram configurados para enviarem comandos ao inversor através do envio de *string* pela porta serial aberta e também para comandos de abertura e fechamento da porta serial.

O funcionamento do aplicativo obedeceu a seguinte lógica de eventos, primeiramente é aberto o canal de comunicação serial, uma rotina verifica os dados para leitura, e na sequência esses dados são mostrados na tela. Após a publicação dos dados, os comandos efetuados pelo usuário são escritos na porta serial perante alguma interação do usuário. E finalizando a rotina de *backup* que é executada automaticamente, essas ações podem ser vistas no fluxograma, Figura 29.

2.4 Aquisição de dados

Foi constituído um canal de comunicação entre PC e microcontrolador, no qual envia comandos e coleta de dados relativos ao ensaio de tribologia. Para isso, foi criada uma interface

Figura 29 – Fluxograma de funcionamento da IHM



Fonte: Autoria própria

gráfica em que o usuário possa configurar sua conexão serial, executar salvamento de dados e acompanhar o experimento de forma remota.

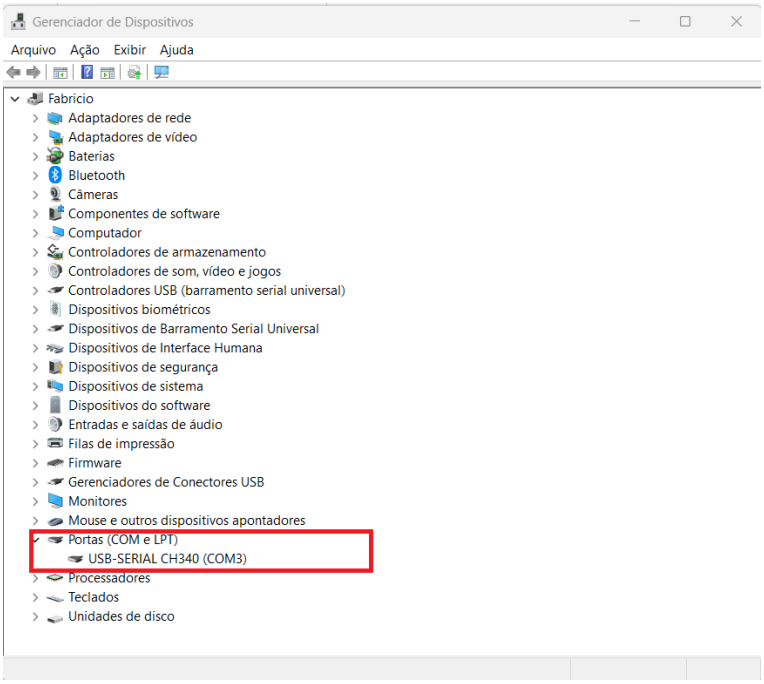
Antes de abrir o aplicativo denominado como "Tribômetro", o usuário deverá consultar qual é porta de comunicação serial disponível no PC, para isso basta clicar pressionar simultaneamente as teclas "Windows e X", selecionar a opção "Gerenciador de Dispositivos", Figura 30. Clicar no item "Portas(COM e LPT)", e o resultado será as portas de comunicação seriais disponíveis na máquina.

Após a identificação da porta "COM", o aplicativo "Tribômetro", poderá ser aberto e iniciar o procedimento de configuração para a realização do experimento tribológico, Figura 31.

Deve-se atribuir ao aplicativo a porta COM observada na etapa anterior e localizar o menu Setup; opção Port Name e clicar nas opções disponíveis (COM1, COM2, ... e COM5), Figura 32 .

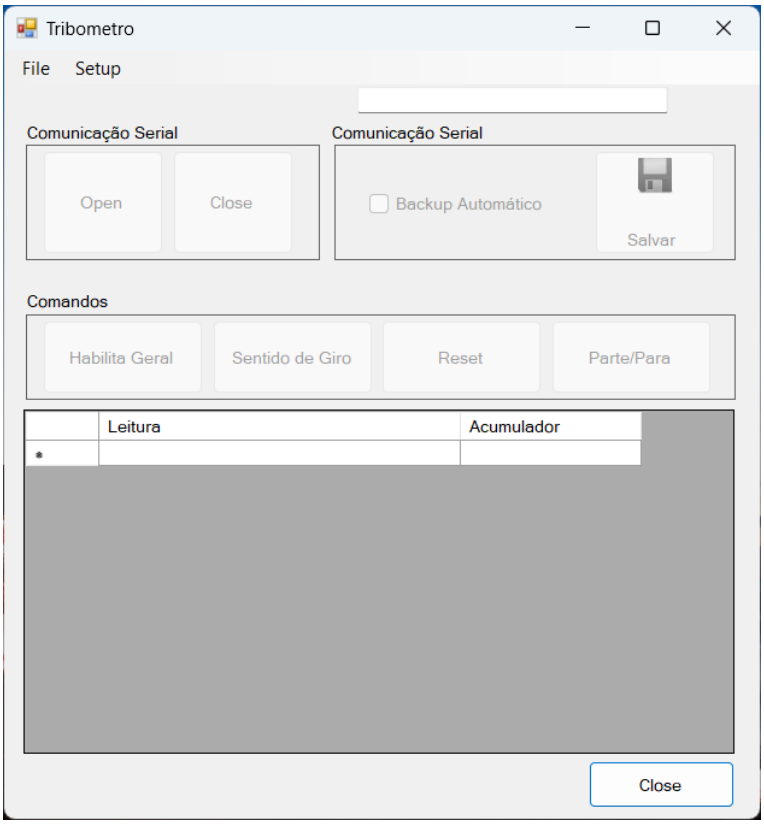
No menu Setup, também deverá ser feito a seleção de velocidade da porta. Neste

Figura 30 – Porta serial instalada



Fonte: Autoria própria

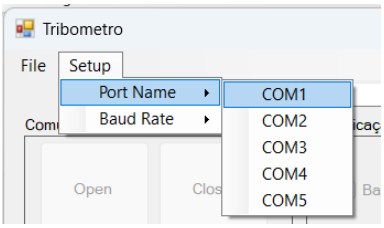
Figura 31 – Abrindo o Aplicativo "Tribômetro"



Fonte: Autoria própria

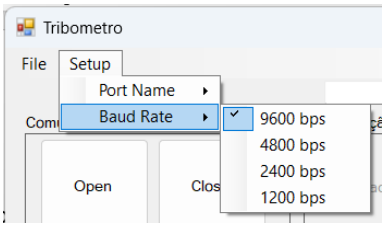
caso foi definido como padrão deste enlace em 9600 *bps* que foi configurado previamente no microcontrolador assim finalizando a configuração do canal de comunicação, Figura 33.

Figura 32 – Menu seleção de portas seriais



Fonte: Autoria própria

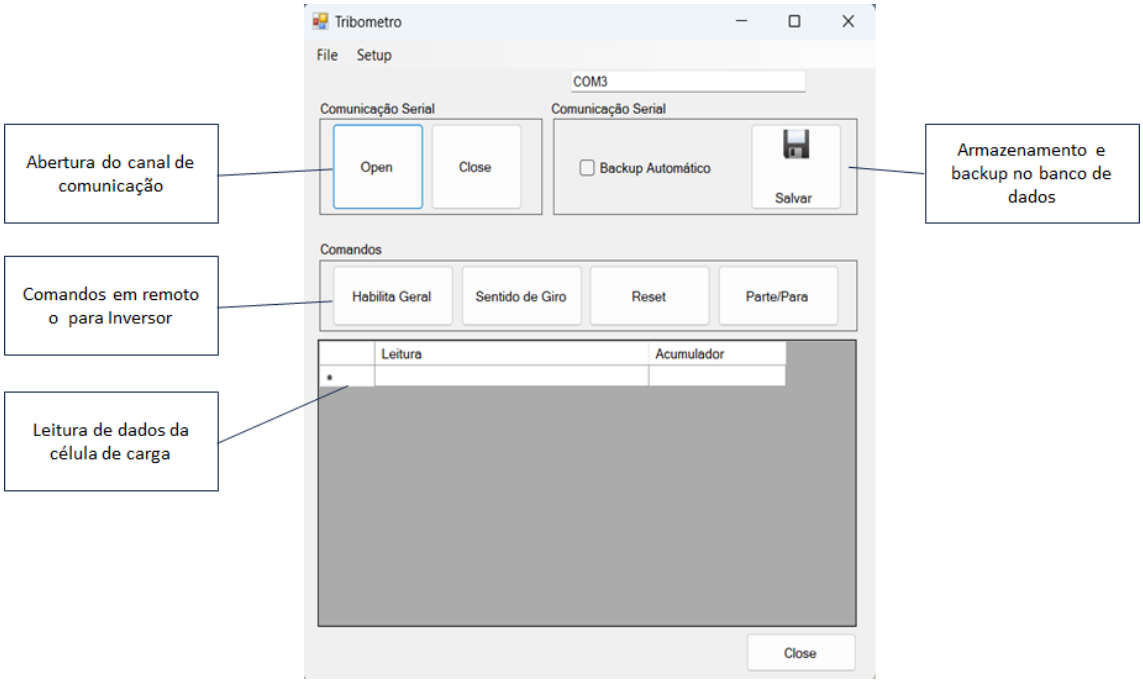
Figura 33 – Menu seleção de *baud rate*



Fonte: Autoria própria

Para dar início a troca de dados o usuário deverá clicar no botão Open e observar a habilitação das opções disponíveis no aplicativo como: comandos para o inversor, leitura dos dados da célula de carga e backup automático para o banco de dados, Figura 34.

Figura 34 – Aplicativo e suas funções



Fonte: Autoria própria

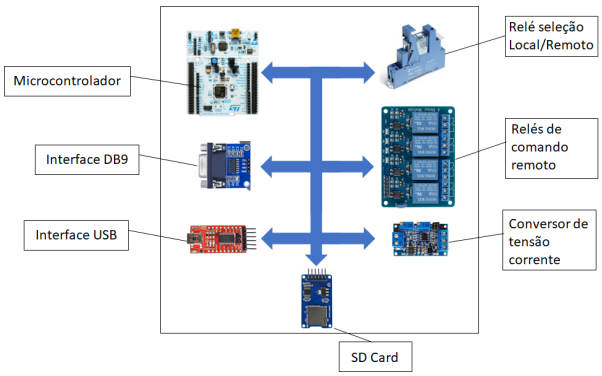
O salvamento no banco de dados, foi previamente configurado para ser armazenado em .csv e depositado na pasta C:\Users\Documentos com o título Log concatenado com a data e horário da realização do experimento, por exemplo \Log _07032023 _135233.csv, este salvamento automático pode ser selecionado através do *check box* "Backup Automático", que

grava os dados em uma frequência pré-determinada de quarenta segundos de forma incremental garantindo a integridade do começo ao final do experimento.

2.5 Acionamentos eletromecânicos e circuitos elétricos para alimentação

Com todo aparato eletrônico supracitado, foi necessário o acondicionamento dos componentes em uma caixa ou painel protegido. No caso, optou-se por uma um painel de dimensões de 30X30X20 cm, devido a quantidade de peças para alocação. A montagem interna do espelho ficou organizada de acordo com a Figura 35.

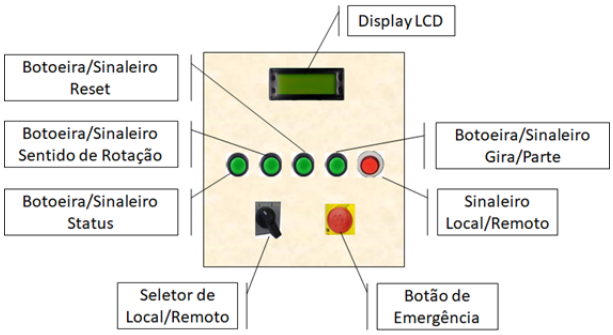
Figura 35 – Montagem do quadro elétrico



Fonte: Autoria própria

Na tampa externa foi montado os componentes eletromecânicos como: sinaleiros, botoeiras e as chaves para a operação em local do tribômetro, da forma disposta na Figura 36.

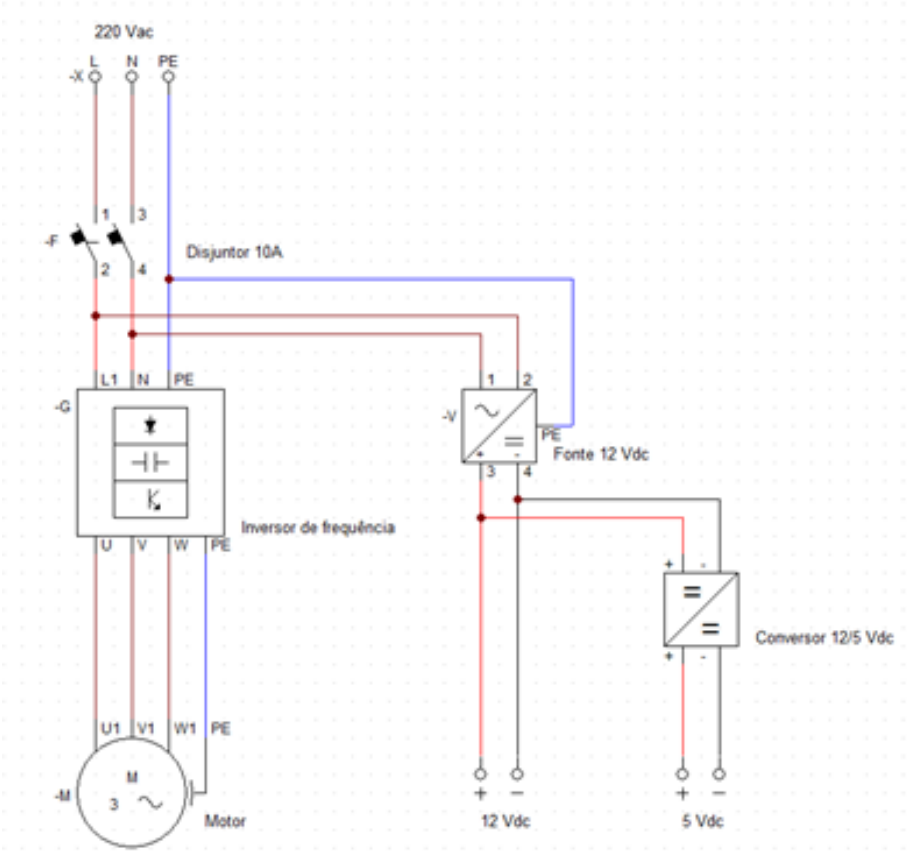
Figura 36 – Montagem do quadro elétrico tampa externa



Fonte: Autoria própria

Para o fornecimento de energia elétrica para o projeto, foi concebida uma entrada de 220 *Vac*, protegida por um disjuntor bipolar de 10 A curva C. Esta entrada é compartilhada pelo painel de componetes eletroeletrônicos e o conjunto inversor de frequência e motor elétrico. No interior do painel foi instalado uma fonte de 12 Vdc destinada aos componentes eletromecânicos, e também um rebaixador de tensão de 12 para 5 Vdc para a alimentação dos componentes eletrônicos, de acordo com a Figura 37.

Figura 37 – Esquema elétrico alimentação 220 Vac

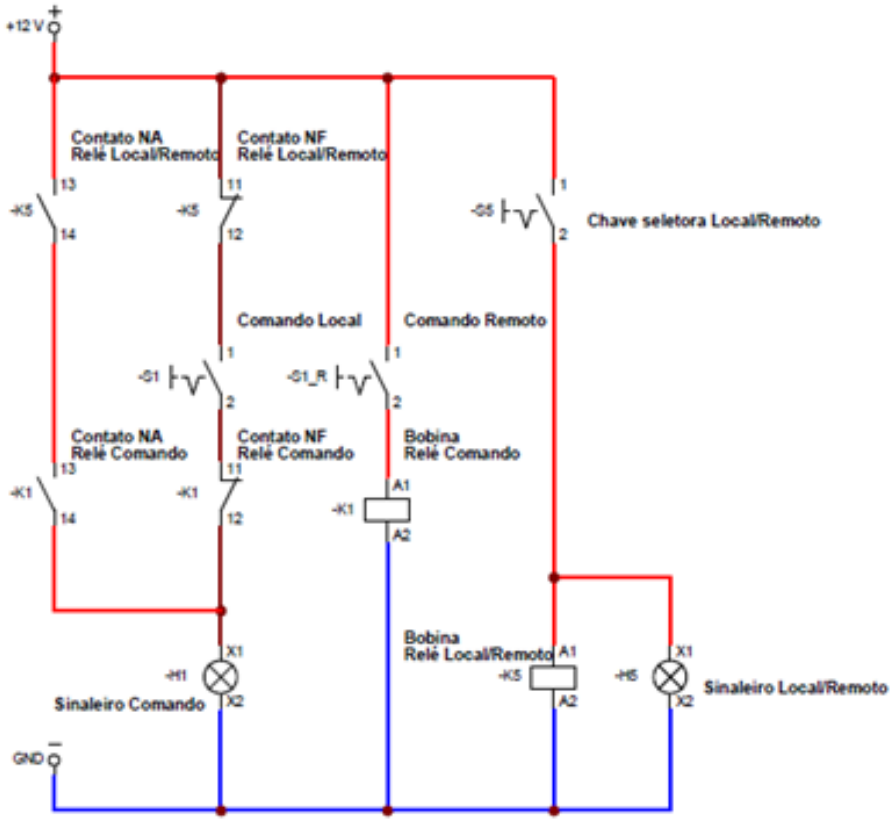


Fonte: Autoria própria

O circuito de 12 Vdc é destinado aos componentes eletromecânicos como: bobinas (K), acionamentos manual (S), acionamentos remoto (S_R) e sinaleiros (H). Estes componentes foram configurados de forma a ter dois modos de operação. O primeiro modo consiste em enviar comandos diretamente pelo PC (operação em remoto) e outra forma é utilizando as botoeiras disponíveis na frente do painel (operação em local). Para isso, foi configurada uma lógica a relé onde seleciona o seu o modo de operação (Local/Remoto) de acordo com a Figura 38, vale lembrar que essa ligação se repete aos quatro comandos disponíveis.

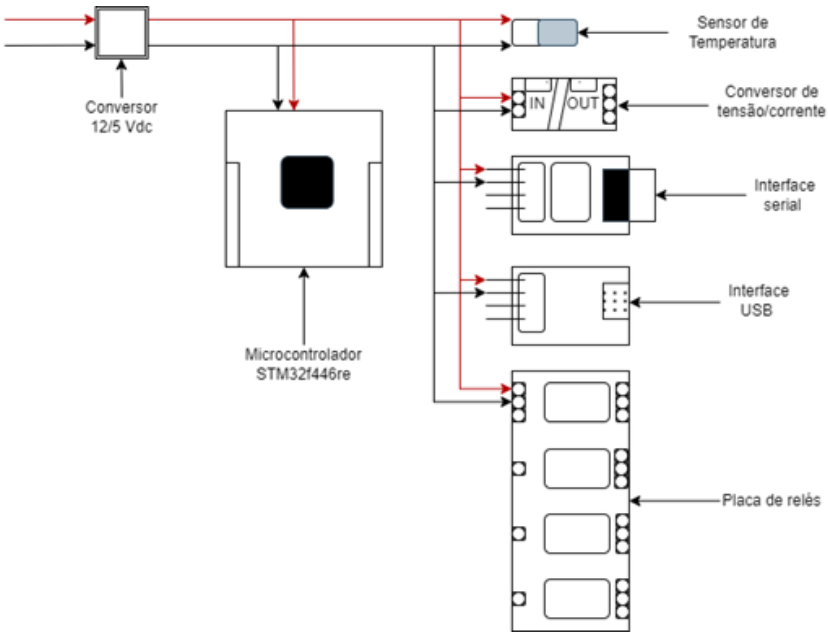
Já o circuito de 5 Vdc fornece a alimentação para as placas eletrônicas como: microcontrolador, placa de relés, cartão de memória, conversor de tensão/corrente, e as interfaces serial e USB. Descrito conforme a Figura 39.

Figura 38 – Esquema elétrico alimentação 12 Vdc



Fonte: Autoria própria

Figura 39 – Esquema elétrico alimentação 5 Vdc



Fonte: Autoria própria

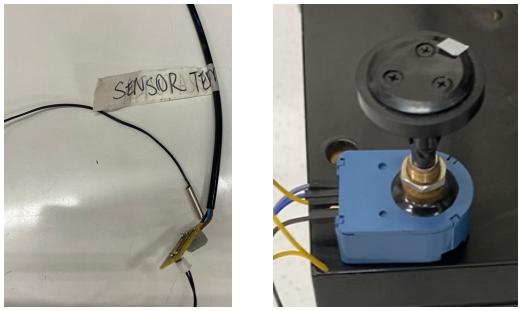
3 RESULTADOS E DISCUSSÕES

O objetivo desse projeto foi a concepção de um sistema eletrônico embarcado, que desempenhasse a automatização de um tribômetro, por meio da tecnologia *ARM* representado pelo STM32F446RE. Esse micrococntrolador, responsável pelo automatismo, foi ligado aos demais periféricos de entrada/saída através de ligações fiadas e enlaces seriais. E os resultados foram obtidos divididos em: sinais de entrada, sinais de saída, comunicação serial, comandos e aplicativo.

3.1 Sinais de entrada

Os sinais de entrada foram compostos por dois sensores responsáveis pela medição e indicação de temperatura (em °C) e rotação (em rpm), Figura 40.

Figura 40 – Sensores de temperatura e rotação



Fonte: Autoria própria

O primeiro sensor é dedicado a medição de temperatura do experimento, cujo funcionamento é baseado na resistência interna do componente *NTC*, ou seja, quando há uma variação da temperatura a sua resitência será modificada bem como a tensão sobre o sensor. Essa variação de tensão é medida pelo circuito ADC do microcontrolador que executa a mudança de unidade de engenharia (tensão para temperatura), resultando em 23.01 graus *Celsius* durante a execução do experimento, e publicada a temperatura no *display* LCD e comparado com o termômetro local do laboratório, conforme a Figura 41.

Comparando o resultado obtido com trabalhos dedicados a medição de temperatura (ZANCHIN et al., 2019) e (MARTINAZZO; ORLANDO, 2016), a solução aplicada foi composta por um termistor NTC 10kΩ, associado a um resistor fixo de 10kΩ, e ligado ao ADC do microcontrolador. Onde rendeu resultados similares ao encontrado nesse experimento, já que o circuito eletrônico usado nesse trabalho é bem similar aos estudos de medição de temperatura.

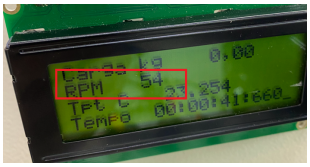
Para a medição de rotação foi utilizado um *Encoder* incremental com a resolução de 256 pulsos a cada revolução completa do eixo. Também, utilizou-se um sensor com duas linhas de pulsos (canais A e B) que são deslocadas para determinar a rotação. Cujo resultado foi de

Figura 41 – Sensor de temperatura ligado ao microcontrolador e disponibilizado no *display* LCD

Fonte: Autoria própria

54 RPM, confrontado com um tacômetro manual, de acordo com a Figura 42. Esse modo de medição foi encontrado no artigo (POP, 2022), onde a aplicação e o resultado foi similar ao apresentado.

Figura 42 – *Encoder* ligado ao microcontrolador e disponibilizado no *display* LCD

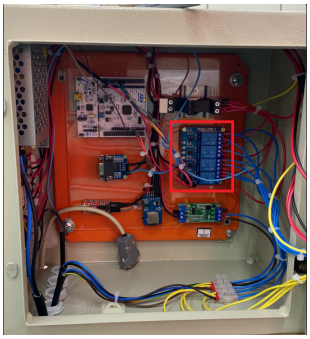


Fonte: Autoria própria

3.2 Sinais de saída

Os sinais de saída do STM32F446RE foram empregados para os quatros comandos discretos (habilita, sentido de giro, reset e parte/para) e um sinal analógico para o controle de rotação do inversor de frequência. Para os comandos discretos, os pinos do microcontrolador foram configurados como *GPIO Output* e estes ligados a placa de relés auxiliares, Figura 43.

Figura 43 – Relés auxiliares



Fonte: Autoria própria

Quando um comando é executado, o microcontrolador eleva o sinal de saída em 3,3 V acionando a bobina do relé auxiliar, assim, enviando a tensão de comando para os contatos do inversor de frequência, conforme a Figura 44.

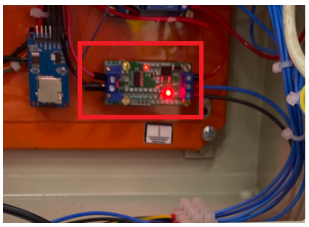
Figura 44 – Comandos discretos no inversor

Fonte: Autoria própria

Essa técnica é conhecida como *driver* de acionamento de carga, que tem uma gama de aplicações, e que foi aplicada nesse trabalho bem como nos estudos (DANECH et al., 2022) e (ZHANG; CHEN, 2018), que também utilizaram a mesma técnica de acionamento discreto obtendo o mesmo grau de funcionalidade.

Já o sinal de saída analógico é proveniente de um circuito DAC (resolução de doze *bits*) do microcontrolador que varia o seu sinal de tensão em até 3,3 V, mas para que este sinal possa ser conectado ao inversor de frequência esse foi convertido em corrente variando de 4 a 20 mA, que corresponde a faixa de rotação nominal que o motor elétrico possa operar. Para isso foi conectado a placa conversora destadaca na Figura 45.

Figura 45 – Circuito conversor tensão para corrente



Fonte: Autoria própria

Ressaltando que nesse experimento o sinal de rotação enviado para motor elétrico é fixo, portanto definido diretamente no *firmware* do microcontrolador de acordo com o apêndice A. O resultado medido no inversor de frequência foi de 990,9 rpm, Figura 46.

3.3 Comunicação serial

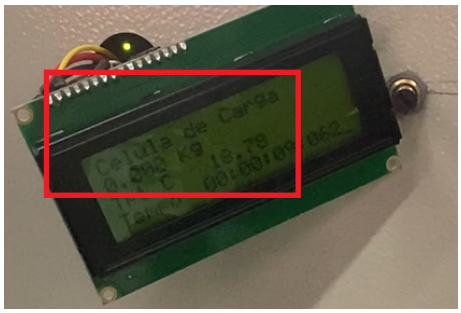
Para a troca de dados entre o microncontrolador, *display Weightech WT21-LCD* e PC, foram configurados dois canais de comunicação seriais no microcontrolador. O primeiro canal é constituído pelo enlace entre STM32F446RE e WT21-LCD, no qual o microcontrolador envia um requisição ao sistema de medição de força, e recebe um *frame* de dados contendo a força aplicada em quilogramas-força, esse valor é disponibilizado no *display* LCD do painel,

Figura 46 – Rotação medida no inversor de frequência

Fonte: Autoria própria

para simples conferência no local, mostrada pela Figura 47. O valor medido no *Weightech WT21-LCD* no momento do experimento foi de 0,00 kg, de acordo com o mostrado na Figura 48.

Figura 47 – Valor célula de carga mostrado no *display* LCD



Fonte: Autoria própria

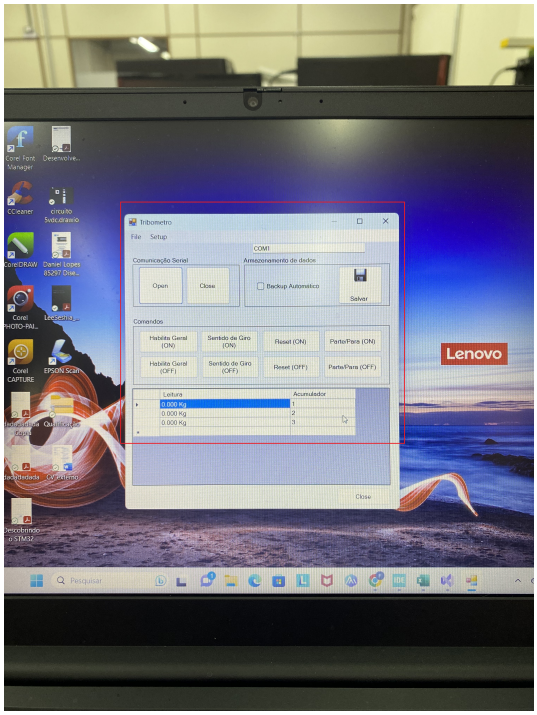
Figura 48 – Valor célula de carga mostrado no *Weightech WT21-LCD*

Fonte: Autoria própria

O segundo enlace foi estabelecido entre o microcontrolador e computador, nesse canal de comunicação as informações trocadas foram:os comandos em remoto provenientes do PC (habilita, sentido de giro, reset e parte/para) e o valor lido juntamente como o *time stamp*, Figura 49.

Para a troca de dados entre microcontrolador e periféricos (*Weightech WT21-LCD* e computador), foi observado um pequeno atraso entre o dado lido no *display* da célula de carga e o publicado no PC, este fato ocorreu devido aos tempos de atualização programados no *firmware* do microcontrolador serem diferentes entre os canais e essa manipulação de dados, que não é observada em outros trabalhos (GAO et al., 2015) e (HAN et al., 2014) dedicados a

Figura 49 – Valor célula de carga mostrado no computador



Fonte: Autoria própria

protocolos serias que monitoram outros periféricos, nesses artigos não foram manipulados os tempos de publicação nas portas seriais.

3.4 Indicações

Foram empregadas duas formas de sinalização, uma contendo sinaleiros a *LED* e outra por *display* LCD. Os sinais luminosos foram aplicados para as indicações dos comandos discretos dados ao inversor e a seleção do modo operacional, de acordo com a Figura 50. Lembrando que nesse projeto as cores empregadas não obedeceram a norma IEC 60073:2002 na sua totalidade, e esse padrão preconiza a utilização de cores dos sinaleiros conforme a sua função desempenhada em cada projeto, vide Tabela 6, confrontado com o trabalho acadêmico (PINHO; ESP8266, 2019) que faz a citação ao uso desses padrões.

Tabela 6 – Comparação de cores dos sinaleiros usadas no projeto e norma IEC 60073:2002

Cor Sinaleiro	Função Projeto	Função Norma IEC 60073:2002
Verde	Habilita Geral e modo Local/Remoto	Equipamento pronto para operar
Ambar	Sentido de Giro	Escravo para mestre
Azul	Reset	Equipamento em Operação Remota
Vermelho	Parte/Para	Equipamento em Operação

Para o *display* LCD foi usado para publicações de informações experimentais como: força de atrito, rotação, temperatura e tempo de duração do ensaio, e o resultado da imple-

Figura 50 – Sinaleiros dos comandos

Fonte: Autoria própria

mentação foi mostrado conforme a Figura 51. Esse tipo de sinalização foi implementado para demonstração da aquisição de dados provinientes do processo de forma a ser clara e objetiva assim como encontrados em artigos como (ZHANG; KANG, 2013) e (CERVO, 2023), que também fazem o uso desse dispositivo para informação ao usuário final.

Figura 51 – Informações mostradas no *display* LCD

Fonte: Autoria própria

3.5 Dados

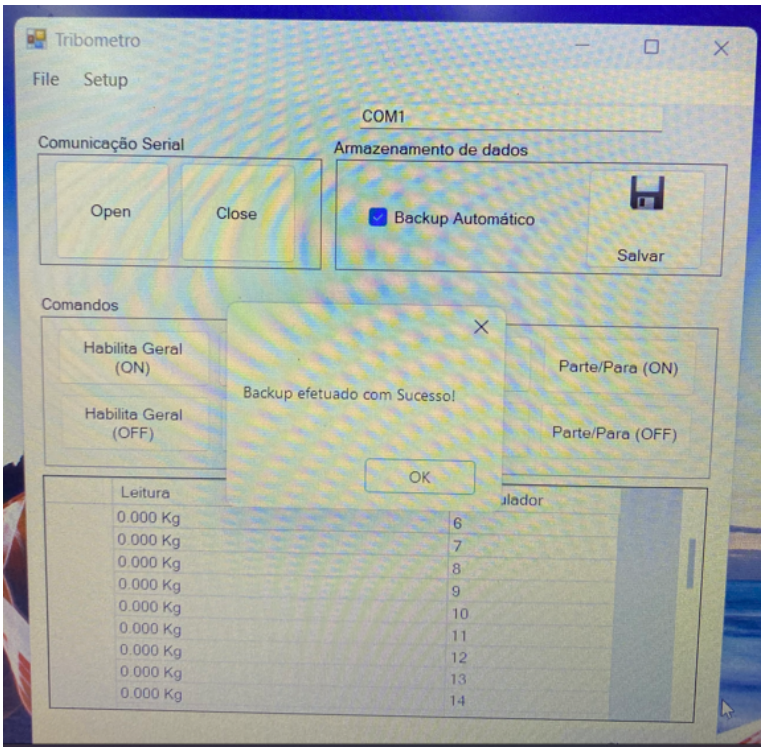
Os dados recebidos foram o valor da força aplicada e *time stamp* concatenados no microcontrolador, conforme *firmware* apresentado no apêndice A. Esses dados foram armazenados utilizando métodos distintos e separados fisicamente. O primeiro local de armazenamento foi o SD *Card*, onde os dados foram salvos por ordem cronológica de aquisição, os dados da Tabela 7 foram retirados do arquivo $log\_dados.txt$ e armazenados no momento do experimento. Onde é observado que o valor medido da força aplicada estava em 0,00 kg, isso foi configurado no *Weightech WT21-LCD* para efeito de propagação de informação no canal de comunicação.

O outro local de armazenamento para os dados de força, foi no próprio computador onde foi instalado o aplicativo Tribômetro, e dentro desse *software* existe uma rotina de salvamento automatico, conforme Figura 52.

Tabela 7 – Dados armazenados no SD *Card*

Peso (Kg)	Tempo (s)
0,00	00:00:23:093
0,00	00:00:25:816
0,00	00:00:26:593
0,00	00:00:27:414
0,00	00:00:28:234
0,00	00:00:29:054

Figura 52 – Rotina de armazenamento de dados no PC



Fonte: Autoria própria

4 CONSIDERAÇÕES FINAIS

4.1 Conclusão do projeto

O projeto primou por um sistema embarcado para automatização de um tribômetro *pin on disc*, que permitiu uma interação entre a instrumentação e eletrônica embarcada. Além disso, a implementação de algumas tecnologias e componentes que auxiliaram a integração do sistema, aumentando a sua confiabilidade e robustez para o contexto em que o tribômetro é aplicado.

O primeiro exemplo são os sensores aplicados a medição de rotação e temperatura do experimento, o sensor de temperatura utilizado (NTC 10kΩ) é bem difundido na área de termometria, tendo uma vasta área de aplicações com bons resultados. Para o sensor de rotação também foi adotada uma tecnologia estabelecida que foi um *encoder* incremental, que mede as revoluções do mecanismo e tem uma boa confiabilidade.

No caso dos comandos discretos para o inversor de frequência, foram ligadas as saída digitais do microcontrolador aos reles auxiliares, constituindo o circuito *driver* para o envio dos quatro comandos disponíveis no projeto. E o controle da rotação do tribômetro que foi realizado em malha aberta através de DAC, esse envia um sinal em tensão que foi transformado para corrente por um conversor, chegando aos bornes do inversor de frequência.

A comutação de dados entre microcontrolador e periféricos como *Display* da célula de carga e computador, foi assegurada por meio de comunicação *RS*-232. Onde o STM32F446RE centralizava as informações e distribuia conforme o seu *firmware* programado.

As indicações dos sinaleiros foram concebidas de acordo com o *status* operacional do tribômetro, bem com as indicações no *Display* LCD que recebe as informações tratadas pelo microcontrolador para informação instantânea ao usuário.

O armazenamento das informações de pesagem necessária no projeto, foi efetuada pelo STM32F446RE, que aquisitava essa variável *Display* da célula de carga através de um canal serial, na sequência o tratamento era feito e enviado para o SD *Card* via interface *SPI* e computador via *RS*-232. O protocolo *SPI* foi de grande valia, devido a melhor integração entre o *hardware* do microcontrolador e cartão de memória funcionarem em *full-duplex*, traduzindo em aumento de confiabilidade no armazenamento de *logs*. Já o envio ao PC seria uma forma de redundância no armazenamento dos valores de pesagem. Pois esta informação é essencial para o ensaio tribológico.

Por fim, o desenvolvimento desse projeto cumpre o objetivo proposto para automatização de um tribômetro *pin on disc* e concebido na forma de um protótipo, para isso foi agregada várias áreas da pós graduação de engenharia elétrica como as disciplinas de: processamento digital de sinais, anális computacional e identificação de sistemas.

4.2 Sugestões de estudos e trabalhos futuros

Tendo em vista permitir a continuidade e evolução do projeto, além do desenvolvimento de novos estudos nessa área, seguem algumas propostas de melhorias e modificações que podem ser efetuadas a partir deste trabalho:

- Adequar as cores dos sinaleiros para obedecer a norma IEC 60073:2002, deixando padronizado as indicações conforme os *status* que indicam.
- Para a melhoria das indicações no *display* LCD 20x4 uma sugestão seria uma substituição por *display* OLED 128x64 que pode ser aplicado para uma melhor definição apresentação das infromações.
- Substituição do enlace serial entre microcontrolador e PC em uma rede *ethernet*, esta rede permite a conexão de redes locais que pode ser conectados vários equipamentos como servidores, impressoras e outros computadores.
- Elaboração de um estudo relacionado a modelagem e identificação de sistemas para utilização desse projeto para simular outros processos.

Referências

ACCADROLLI, G.; VERNEY, J. de. Desenvolvimento de um dispositivo para ensaios tribológicos do tipo pino-sobre-disco. 2017. Citado na página 2.

ASTM-G99-17. **Method for Wear Testing With a Pin on Disc Apparatus, [S. l.]**. 2020. Citado 2 vezes nas páginas 2 e 3.

BOYLESTAD, R. L.; NASHELSKY, L. **Dispositivos Eletrônicos e Teoria de Circuitos,(11aed.)**. [S.l.]: Pearson, 2013. Citado na página 11.

BROWN, G. Descobrindo o microcontrolador stm32. 2012. Citado 2 vezes nas páginas 17 e 18.

CERVO, G. C. Desenvolvimento e teste de um sistema embarcado para o controle de uma guia de solda. Universidade Federal de Santa Maria, 2023. Citado na página 37.

CHAPMAN, S. J. **Fundamentos de máquinas elétricas**. [S.l.]: AMGH editora, 2013. Citado 2 vezes nas páginas 5 e 6.

DALLY, J. W.; RILEY, W. F.; MCCONNELL, K. G. Instrumentation for engineering measurements, john willey & sons. **Inc., New York, USA**, 1993. Citado na página 8.

DANECH, S. et al. Implementation of pid based automatic temperature control system using stm32. In: IEEE. **2022 6th International Conference On Computing, Communication, Control And Automation (ICCUBEA**. [S.l.], 2022. p. 1–5. Citado na página 34.

DOWSON, D. History of tribology. **(No Title)**, 1979. Citado na página 2.

FRANCHI, C. M. Inversores de frequência. **Teoria e Aplicações**, v. 2, 2013. Citado na página 7.

GAO, T. et al. Cdt communication protocol realization based on stm32. In: ATLANTIS PRESS. **2015 International Conference on Education, Management, Information and Medicine**. [S.l.], 2015. p. 100–106. Citado na página 35.

HAN, T. et al. Method and realization of central air-conditioning control strategy communication based on stm32. **Advanced Materials Research**, Trans Tech Publ, v. 1039, p. 328–333, 2014. Citado na página 35.

HOLMBERG, K.; ERDEMIR, A. Influence of tribology on global energy consumption, costs and emissions. **Friction**, Springer, v. 5, p. 263–284, 2017. Citado na página 2.

HOROWITZ, P.; HILL, W. **A arte da eletrônica: circuitos eletrônicos e microeletrônica**. [S.l.]: Porto Alegre: Bookman, 2017. Citado 2 vezes nas páginas 11 e 15.

JOST, H. P. Tribology—origin and future. **Wear**, Elsevier, v. 136, n. 1, p. 1–17, 1990. Citado na página 1.

JÚNIOR, G. C. D. N. **Máquinas elétricas: teoria e ensaios**. [S.l.]: Saraiva Educação SA, 2011. Citado na página 5.

LEE, E. A.; SESHIA, S. A. **Introduction to embedded systems: A cyber-physical systems approach**. [S.l.]: MIT press, 2016. Citado na página 2.

MARTINAZZO, C. A.; ORLANDO, T. **Comparação entre três tipos de sensores de temperatura em associação com arduíno**. [S.l.]: Perspectiva, Erechim, 2016. Citado na página 32.

PINHO, J. D. J. V.; ESP8266, C. P. U. Bacharelado em engenharia elétrica. 2019. Citado na página 36.

POP, A. A. Incremental encoder speed acquisition using an stm32 microcontroller and ni elvis. **Sensors**, MDPI, v. 22, n. 14, p. 5127, 2022. Citado 3 vezes nas páginas 13, 14 e 33.

SANTOS, Y. **PROJETO MECÂNICO DE TRIBÔMETRO DO TIPO PINO SOBRE DISCO**. Dissertação (B.S. thesis) — Universidade Federal do Rio Grande do Norte, 2021. Citado na página 2.

SOARES, L. G. **Proposta de construção de um tribômetro modelo pino sobre disco ("pin on disk")**. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2014. Citado na página 1.

ZANCHIN, G. M. et al. Mecanismo de aquisição de dados de temperatura em caldeira do tipo flamotubular com economizador. In: **8ª MOEPEX**. [S.l.: s.n.], 2019. Citado na página 32.

ZHANG, H.-f.; KANG, W. Design of the data acquisition system based on stm32. **Procedia Computer Science**, Elsevier, v. 17, p. 222–228, 2013. Citado na página 37.

ZHANG, L.; CHEN, X. Research of water quality monitoring and control system based on stm32. In: ATLANTIS PRESS. **2018 International Conference on Mechanical, Electrical, Electronic Engineering & Science (MEEES 2018)**. [S.l.], 2018. p. 26–29. Citado na página 34.

Apêndices

APÊNDICE A – Código C++ implementado no STM32F446RE compilador STMCubeIDE versão 1.12.0

```
1
2  /* Autor: Fabricio Cassanho Teodoro*/
3  /**
4  /* USER CODE END Header */
5  /* Includes ------------------------------------------------------------------*/
6  #include "main.h"
7  #include "fatfs.h"
8
9  /* Private includes ----------------------------------------------------------*/
10 /* USER CODE BEGIN Includes */
11 #include "string.h"
12 #include "stdlib.h"
13 #include "stdint.h"
14 #include "lcd_v1.1.h"
15 #include "fatfs_sd.h"
16 #include "stdio.h"
17
18 /* Private typedef -----------------------------------------------------------*/
19 /* USER CODE BEGIN PTD */
20
21 /* USER CODE END PTD */
22
23 /* Private define ------------------------------------------------------------*/
24 /* USER CODE BEGIN PD */
25 #define RtcHandle &hrtc
26 static uint16_t YearStart = 2000;
27 float valor;
28 char msg[10];
29 char message1[50], message2[50],send[100];
30 uint8_t size;
31 uint16_t mseconds;
32 char buffer_rx[1];
33 char cat[60];
34 char command[1];
35
36 /* USER CODE END PD */
37
38 /* Private macro -------------------------------------------------------------*/
39 /* USER CODE BEGIN PM */
40
41 /* USER CODE END PM */
42
43 /* Private variables ---------------------------------------------------------*/
44 RTC_HandleTypeDef hrtc;
45 SPI_HandleTypeDef hspi2;
46 TIM_HandleTypeDef htim2;
47 UART_HandleTypeDef huart4;
48 UART_HandleTypeDef huart3;
49 UART_HandleTypeDef huart6;
50
51 /* USER CODE BEGIN PV */
52
53 /* USER CODE END PV */
```

```
54
55 /* Private function prototypes -----------------------------------------------*/
56 void SystemClock_Config(void);
57 static void MX_GPIO_Init(void);
58 static void MX_RTC_Init(void);
59 static void MX_UART4_Init(void);
60 static void MX_TIM2_Init(void);
61 static void MX_SPI2_Init(void);
62 static void MX_USART3_UART_Init(void);
63 static void MX_USART6_UART_Init(void);
64 /* USER CODE BEGIN PFP */
65
66
67
68
69 /* USER CODE END PFP */
70
71 /* Private user code ---------------------------------------------------------*/
72 /* USER CODE BEGIN 0 */
73
74 /* USER CODE END 0 */
75
76 /**
77   * @brief  The application entry point.
78   * @retval int
79   */
80 int main(void)
81 {
82
83
84   /* MCU Configuration--------------------------------------------------------*/
85
86   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
87   HAL_Init();
88
89
90   /* Configure the system clock */
91   SystemClock_Config();
92
93   /* USER CODE BEGIN SysInit */
94
95   /* USER CODE END SysInit */
96
97   /* Initialize all configured peripherals */
98   MX_GPIO_Init();
99   MX_RTC_Init();
100  MX_UART4_Init();
101  MX_TIM2_Init();
102  MX_SPI2_Init();
103  MX_USART3_UART_Init();
104  MX_USART6_UART_Init();
105  MX_FATFS_Init();
106
107  HAL_TIM_Base_Start_IT(&htim2);
108
109  /* USER CODE BEGIN 2 */
110  //Inicializa o sistema
111   lcd_20x4_4bits_Init();
112   lcd_limpa_display();
113   lcd_posicao_do_cursor(0, 0);
```

```
114     lcd_escreve_string("FirmWare V3.4\r");
115     HAL_Delay(2000);
116     lcd_limpa_display();
117
118     // Inicializa RTC
119     RTC_TimeTypeDef sTime;
120     RTC_DateTypeDef sDate;
121
122     // Incializando data e hora
123     sDate.WeekDay = RTC_WEEKDAY_MONDAY;
124     sDate.Date = 22;
125     sDate.Month = 2;
126     sDate.Year = 20; // from 0-99 (1972 == 72, start year = 1900)
127     HAL_RTC_SetDate(RtcHandle, &sDate, RTC_FORMAT_BIN);
128     sTime.Hours = 00;
129     sTime.Minutes = 00;
130     HAL_RTC_SetTime(RtcHandle, &sTime, RTC_FORMAT_BIN);
131
132     // Start Wakeup timer with configuration (32768Hz / 16 = 2048) -> Counter = 2048
133     HAL_RTCEx_SetWakeUpTimer_IT(RtcHandle, 2048-1, RTC_WAKEUPCLOCK_RTCCLK_DIV16);
134   /* USER CODE END 2 */
135     /* Montagem do SDCARD-----------------------------------------------------*/
136
137       HAL_Delay (500);
138
139       // Gravacao de dados no SDCARD
140       FATFS       FatFs;                //Fatfs handle
141       FIL         fil;                  //File handle
142       FRESULT     fres;                 //Result after operations
143
144           //Mount the SD Card
145           fres = f_mount(&FatFs, "", 1);     //1=mount now
146           //Read the SD Card Total size and Free Size
147           FATFS *pfs;
148           DWORD fre_clust;
149           uint32_t totalSpace, freeSpace;
150
151           f_getfree("", &fre_clust, &pfs);
152            totalSpace = (uint32_t)((pfs->n_fatent - 2) * pfs->csize * 0.5);
153           freeSpace = (uint32_t)(fre_clust * pfs->csize * 0.5);
154   /* Infinite loop */
155   /* USER CODE BEGIN WHILE */
156   while (1)
157   {
158     /* USER CODE END WHILE */
159
160            HAL_Delay(300);
161            HAL_UART_Receive(&huart4, (uint8_t*)buffer_rx, 1, 100);
162            HAL_Delay(300);
163
164            // Inicia a primeira linha do STATUS
165           lcd_posicao_do_cursor(0, 0);
166           lcd_escreve_string("Status\r");
167
168            //Comando de Habilita o inversor
169            if(buffer_rx[0] == 'a')
170            {
171            lcd_posicao_do_cursor(0, 0);
172            lcd_escreve_string("Status = Habilitado\r");
173            HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
```

```
174             HAL_Delay(100);
175             HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
176             HAL_Delay(100);
177             }
178           //Comando de sentido de Giro
179           if(buffer_rx[0] == 'b')
180           {
181           lcd_posicao_do_cursor(0, 0);
182           lcd_escreve_string("Status = Reverse \r");
183           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_SET);
184           HAL_Delay(100);
185           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_5, GPIO_PIN_RESET);
186           HAL_Delay(100);
187           }
188           //Comando de reset
189           if(buffer_rx[0] == 'c')
190           {
191           lcd_posicao_do_cursor(0, 0);
192           lcd_escreve_string("Status = Reset \r");
193           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_SET);
194           HAL_Delay(100);
195           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_6, GPIO_PIN_RESET);
196           HAL_Delay(100);
197           }
198           //Comando de Parte/Para
199           if(buffer_rx[0] == 'd')
200           {
201           lcd_posicao_do_cursor(0, 0);
202           lcd_escreve_string("Status = Parte/Para \r");
203           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_SET);
204           HAL_Delay(100);
205           HAL_GPIO_WritePin(GPIOA, GPIO_PIN_7, GPIO_PIN_RESET);
206           HAL_Delay(100);
207           }
208           //Geracao de Data e Hora
209            HAL_RTC_GetTime(RtcHandle, &sTime, RTC_FORMAT_BIN);
210            HAL_RTC_GetDate(RtcHandle, &sDate, RTC_FORMAT_BIN);
211            mseconds = (sTime.SubSeconds * 1000) / (sTime.SecondFraction + 1);
212         //Configuracao do tempo
213            size = sprintf(message2, "%2.2u:%2.2u:%2.2u:%3.3u\r", sTime.Hours, sTime.Minutes,
214                   sTime.Seconds, mseconds);
215
216            HAL_Delay(10);
217            lcd_posicao_do_cursor(1, 0);
218            lcd_escreve_string("RPM\r");
219            HAL_Delay(10);
220            lcd_posicao_do_cursor(2, 0);
221            lcd_escreve_string("Temperatura C\r");
222            lcd_posicao_do_cursor(3,0);
223            lcd_escreve_string("Tempo\r");
224            lcd_posicao_do_cursor(3,7);
225            lcd_escreve_string(message2);
226            HAL_Delay(1000);
227
228            size = sprintf(cat, message2);
229            size = sprintf(cat, "teste \n");
230            HAL_Delay(50);
231
232            //Criacao e abertura do arquivo
233            fres = f_open(&fil, "log_dados_1.txt", FA_WRITE |FA_READ | FA_OPEN_APPEND);
```

```
234             HAL_Delay(100);
235             fres = f_lseek(&fil, fil.fptr);
236             fres = f_puts(cat, &fil);
237             fres = f_close(&fil);
238       /* USER CODE BEGIN 3 */
239     }
240     /* USER CODE END 3 */
241   }
242
243   /**
244     * @brief System Clock Configuration
245     * @retval None
246     */
247   void SystemClock_Config(void)
248   {
249     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
250     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
251
252     /** Configure the main internal regulator output voltage
253     */
254     __HAL_RCC_PWR_CLK_ENABLE();
255     __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE3);
256
257     /** Initializes the RCC Oscillators according to the specified parameters
258     * in the RCC_OscInitTypeDef structure.
259     */
260     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI|RCC_OSCILLATORTYPE_LSI;
261     RCC_OscInitStruct.LSEState = RCC_LSE_OFF;
262     RCC_OscInitStruct.HSIState = RCC_HSI_ON;
263     RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
264     RCC_OscInitStruct.LSIState = RCC_LSI_ON;
265     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
266     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
267     RCC_OscInitStruct.PLL.PLLM = 8;
268     RCC_OscInitStruct.PLL.PLLN = 50;
269     RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
270     RCC_OscInitStruct.PLL.PLLQ = 2;
271     RCC_OscInitStruct.PLL.PLLR = 2;
272     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
273     {
274       Error_Handler();
275     }
276
277     /** Initializes the CPU, AHB and APB buses clocks
278     */
279     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
280                                   |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
281     RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
282     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
283     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
284     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
285
286     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_1) != HAL_OK)
287     {
288       Error_Handler();
289     }
290   }
291
292   /**
293     * @brief RTC Initialization Function
```

```
294   * @param None
295   * @retval None
296   */
297 static void MX_RTC_Init(void)
298 {
299
300   /* USER CODE BEGIN RTC_Init 0 */
301
302   /* USER CODE END RTC_Init 0 */
303
304   RTC_TimeTypeDef sTime = {0};
305   RTC_DateTypeDef sDate = {0};
306   RTC_AlarmTypeDef sAlarm = {0};
307
308   /* USER CODE BEGIN RTC_Init 1 */
309
310   /* USER CODE END RTC_Init 1 */
311
312   /** Initialize RTC Only
313   */
314   hrtc.Instance = RTC;
315   hrtc.Init.HourFormat = RTC_HOURFORMAT_24;
316   hrtc.Init.AsynchPrediv = 127;
317   hrtc.Init.SynchPrediv = 255;
318   hrtc.Init.OutPut = RTC_OUTPUT_DISABLE;
319   hrtc.Init.OutPutPolarity = RTC_OUTPUT_POLARITY_HIGH;
320   hrtc.Init.OutPutType = RTC_OUTPUT_TYPE_OPENDRAIN;
321   if (HAL_RTC_Init(&hrtc) != HAL_OK)
322   {
323     Error_Handler();
324   }
325
326   /* USER CODE BEGIN Check_RTC_BKUP */
327
328   /* USER CODE END Check_RTC_BKUP */
329
330   /** Initialize RTC and set the Time and Date
331   */
332   sTime.Hours = 0x0;
333   sTime.Minutes = 0x0;
334   sTime.Seconds = 0x0;
335   sTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
336   sTime.StoreOperation = RTC_STOREOPERATION_RESET;
337   if (HAL_RTC_SetTime(&hrtc, &sTime, RTC_FORMAT_BCD) != HAL_OK)
338   {
339     Error_Handler();
340   }
341   sDate.WeekDay = RTC_WEEKDAY_MONDAY;
342   sDate.Month = RTC_MONTH_JANUARY;
343   sDate.Date = 0x1;
344   sDate.Year = 0x0;
345
346   if (HAL_RTC_SetDate(&hrtc, &sDate, RTC_FORMAT_BCD) != HAL_OK)
347   {
348     Error_Handler();
349   }
350
351   /** Enable the Alarm A
352   */
353   sAlarm.AlarmTime.Hours = 0x0;
```

```
354   sAlarm.AlarmTime.Minutes = 0x0;
355   sAlarm.AlarmTime.Seconds = 0x0;
356   sAlarm.AlarmTime.SubSeconds = 0x0;
357   sAlarm.AlarmTime.DayLightSaving = RTC_DAYLIGHTSAVING_NONE;
358   sAlarm.AlarmTime.StoreOperation = RTC_STOREOPERATION_RESET;
359   sAlarm.AlarmMask = RTC_ALARMMASK_NONE;
360   sAlarm.AlarmSubSecondMask = RTC_ALARMSUBSECONDMASK_ALL;
361   sAlarm.AlarmDateWeekDaySel = RTC_ALARMDATEWEEKDAYSEL_DATE;
362   sAlarm.AlarmDateWeekDay = 0x1;
363   sAlarm.Alarm = RTC_ALARM_A;
364   if (HAL_RTC_SetAlarm(&hrtc, &sAlarm, RTC_FORMAT_BCD) != HAL_OK)
365   {
366     Error_Handler();
367   }
368   /* USER CODE BEGIN RTC_Init 2 */
369
370   /* USER CODE END RTC_Init 2 */
371
372 }
373
374 /**
375   * @brief SPI2 Initialization Function
376   * @param None
377   * @retval None
378   */
379 static void MX_SPI2_Init(void)
380 {
381
382   /* USER CODE BEGIN SPI2_Init 0 */
383
384   /* USER CODE END SPI2_Init 0 */
385
386   /* USER CODE BEGIN SPI2_Init 1 */
387
388   /* USER CODE END SPI2_Init 1 */
389   /* SPI2 parameter configuration*/
390   hspi2.Instance = SPI2;
391   hspi2.Init.Mode = SPI_MODE_MASTER;
392   hspi2.Init.Direction = SPI_DIRECTION_2LINES;
393   hspi2.Init.DataSize = SPI_DATASIZE_8BIT;
394   hspi2.Init.CLKPolarity = SPI_POLARITY_LOW;
395   hspi2.Init.CLKPhase = SPI_PHASE_1EDGE;
396   hspi2.Init.NSS = SPI_NSS_SOFT;
397   hspi2.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
398   hspi2.Init.FirstBit = SPI_FIRSTBIT_MSB;
399   hspi2.Init.TIMode = SPI_TIMODE_DISABLE;
400   hspi2.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
401   hspi2.Init.CRCPolynomial = 10;
402   if (HAL_SPI_Init(&hspi2) != HAL_OK)
403   {
404     Error_Handler();
405   }
406   /* USER CODE BEGIN SPI2_Init 2 */
407
408   /* USER CODE END SPI2_Init 2 */
409
410 }
411
412 /**
413   * @brief TIM2 Initialization Function
```

```
414   * @param None
415   * @retval None
416   */
417 static void MX_TIM2_Init(void)
418 {
419
420   /* USER CODE BEGIN TIM2_Init 0 */
421
422   /* USER CODE END TIM2_Init 0 */
423
424   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
425   TIM_MasterConfigTypeDef sMasterConfig = {0};
426
427   /* USER CODE BEGIN TIM2_Init 1 */
428
429   /* USER CODE END TIM2_Init 1 */
430   htim2.Instance = TIM2;
431   htim2.Init.Prescaler = 50-1;
432   htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
433   htim2.Init.Period = 65535;
434   htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
435   htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
436   if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
437   {
438     Error_Handler();
439   }
440   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
441   if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
442   {
443     Error_Handler();
444   }
445   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
446   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
447   if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
448   {
449     Error_Handler();
450   }
451   /* USER CODE BEGIN TIM2_Init 2 */
452
453   /* USER CODE END TIM2_Init 2 */
454
455 }
456
457 /**
458   * @brief UART4 Initialization Function
459   * @param None
460   * @retval None
461   */
462 static void MX_UART4_Init(void)
463 {
464
465   /* USER CODE BEGIN UART4_Init 0 */
466
467   /* USER CODE END UART4_Init 0 */
468
469   /* USER CODE BEGIN UART4_Init 1 */
470
471   /* USER CODE END UART4_Init 1 */
472   huart4.Instance = UART4;
473   huart4.Init.BaudRate = 9600;
```

```
474   huart4.Init.WordLength = UART_WORDLENGTH_8B;
475   huart4.Init.StopBits = UART_STOPBITS_1;
476   huart4.Init.Parity = UART_PARITY_NONE;
477   huart4.Init.Mode = UART_MODE_TX_RX;
478   huart4.Init.HwFlowCtl = UART_HWCONTROL_NONE;
479   huart4.Init.OverSampling = UART_OVERSAMPLING_16;
480   if (HAL_UART_Init(&huart4) != HAL_OK)
481   {
482     Error_Handler();
483   }
484   /* USER CODE BEGIN UART4_Init 2 */
485
486   /* USER CODE END UART4_Init 2 */
487
488 }
489
490 /**
491   * @brief USART3 Initialization Function
492   * @param None
493   * @retval None
494   */
495 static void MX_USART3_UART_Init(void)
496 {
497
498   /* USER CODE BEGIN USART3_Init 0 */
499
500   /* USER CODE END USART3_Init 0 */
501
502   /* USER CODE BEGIN USART3_Init 1 */
503
504   /* USER CODE END USART3_Init 1 */
505   huart3.Instance = USART3;
506   huart3.Init.BaudRate = 115200;
507   huart3.Init.WordLength = UART_WORDLENGTH_8B;
508   huart3.Init.StopBits = UART_STOPBITS_1;
509   huart3.Init.Parity = UART_PARITY_NONE;
510   huart3.Init.Mode = UART_MODE_TX_RX;
511   huart3.Init.HwFlowCtl = UART_HWCONTROL_NONE;
512   huart3.Init.OverSampling = UART_OVERSAMPLING_16;
513   if (HAL_UART_Init(&huart3) != HAL_OK)
514   {
515     Error_Handler();
516   }
517   /* USER CODE BEGIN USART3_Init 2 */
518
519   /* USER CODE END USART3_Init 2 */
520
521 }
522
523 /**
524   * @brief USART6 Initialization Function
525   * @param None
526   * @retval None
527   */
528 static void MX_USART6_UART_Init(void)
529 {
530
531   /* USER CODE BEGIN USART6_Init 0 */
532
533   /* USER CODE END USART6_Init 0 */
```

```
534
535   /* USER CODE BEGIN USART6_Init 1 */
536
537   /* USER CODE END USART6_Init 1 */
538   huart6.Instance = USART6;
539   huart6.Init.BaudRate = 115200;
540   huart6.Init.WordLength = UART_WORDLENGTH_8B;
541   huart6.Init.StopBits = UART_STOPBITS_1;
542   huart6.Init.Parity = UART_PARITY_NONE;
543   huart6.Init.Mode = UART_MODE_TX_RX;
544   huart6.Init.HwFlowCtl = UART_HWCONTROL_NONE;
545   huart6.Init.OverSampling = UART_OVERSAMPLING_16;
546   if (HAL_UART_Init(&huart6) != HAL_OK)
547   {
548     Error_Handler();
549   }
550   /* USER CODE BEGIN USART6_Init 2 */
551
552   /* USER CODE END USART6_Init 2 */
553
554 }
555
556 /**
557   * @brief GPIO Initialization Function
558   * @param None
559   * @retval None
560   */
561 static void MX_GPIO_Init(void)
562 {
563   GPIO_InitTypeDef GPIO_InitStruct = {0};
564 /* USER CODE BEGIN MX_GPIO_Init_1 */
565 /* USER CODE END MX_GPIO_Init_1 */
566
567   /* GPIO Ports Clock Enable */
568   __HAL_RCC_GPIOC_CLK_ENABLE();
569   __HAL_RCC_GPIOA_CLK_ENABLE();
570   __HAL_RCC_GPIOB_CLK_ENABLE();
571
572   /*Configure GPIO pin Output Level */
573   HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7
574                           |LCD_RS_Pin|GPIO_PIN_9|LCD_D7_Pin, GPIO_PIN_RESET);
575
576   /*Configure GPIO pin Output Level */
577   HAL_GPIO_WritePin(GPIOB, GPIO_PIN_0|LCD_EN_Pin|LCD_D6_Pin|LCD_D4_Pin
578                           |LCD_D5_Pin, GPIO_PIN_RESET);
579
580   /*Configure GPIO pins : PA4 PA5 PA6 PA7
581                            LCD_RS_Pin PA9 LCD_D7_Pin */
582   GPIO_InitStruct.Pin = GPIO_PIN_4|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_7
583                           |LCD_RS_Pin|GPIO_PIN_9|LCD_D7_Pin;
584   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
585   GPIO_InitStruct.Pull = GPIO_NOPULL;
586   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
587   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
588
589   /*Configure GPIO pins : PB0 LCD_EN_Pin LCD_D6_Pin LCD_D4_Pin
590                            LCD_D5_Pin */
591   GPIO_InitStruct.Pin = GPIO_PIN_0|LCD_EN_Pin|LCD_D6_Pin|LCD_D4_Pin
592                           |LCD_D5_Pin;
593   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
```

```
594     GPIO_InitStruct.Pull = GPIO_NOPULL;
595     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
596     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
597
598   /* USER CODE BEGIN MX_GPIO_Init_2 */
599   /* USER CODE END MX_GPIO_Init_2 */
600   }
601
602   /* USER CODE BEGIN 4 */
603
604   /* USER CODE END 4 */
605
606   /**
607     * @brief  This function is executed in case of error occurrence.
608     * @retval None
609     */
610   void Error_Handler(void)
611   {
612     /* USER CODE BEGIN Error_Handler_Debug */
613     /* User can add his own implementation to report the HAL error return state */
614     __disable_irq();
615     while (1)
616     {
617     }
618     /* USER CODE END Error_Handler_Debug */
619   }
620
621   #ifdef  USE_FULL_ASSERT
622   /**
623     * @brief  Reports the name of the source file and the source line number
624     *         where the assert_param error has occurred.
625     * @param  file: pointer to the source file name
626     * @param  line: assert_param error line source number
627     * @retval None
628     */
629   void assert_failed(uint8_t *file, uint32_t line)
630   {
631     /* USER CODE BEGIN 6 */
632     /* User can add his own implementation to report the file name and line number,
633        ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
634     /* USER CODE END 6 */
635   }
636   #endif /* USE_FULL_ASSERT */
```

APÊNDICE B – Código C em MS Visual Studio 2022 - IHM

```
1
2  using System;
3  using Excel = Microsoft.Office.Interop.Excel;
4  using System.Collections.Generic;
5  using System.ComponentModel;
6  using System.Data;
7  using System.Drawing;
8  using System.IO.Ports;
9  using System.Linq;
10 using System.Text;
11 using System.Threading.Tasks;
12 using System.Windows.Forms;
13 using static System.Windows.Forms.LinkLabel;
14
15 namespace Tribometro
16 {
17     public partial class Form1 : Form
18          {
19         // Declaracao de variaveis globais
20         int cont = 0;
21         string[] ports = SerialPort.GetPortNames();
22         string comm;
23
24
25         public Form1()
26         {
27
28             InitializeComponent();
29
30         }
31
32         private void button1_Click(object sender, EventArgs e)
33         {
34             Close();
35             serialPort1.Close();
36         }
37
38         private void Form1_Load(object sender, EventArgs e)
39         {
40             // Condicoes iniciais
41             // Pre - set baud raute
42             serialPort1.BaudRate = 9600;
43             bpsToolStripMenuItem.Checked = true;
44             //Portas seriais existente no PC
45             comm = ports[0].ToString();
46             textBox1.Text = comm;
47         }
48
49         private void Btn_Liga_Click(object sender, EventArgs e)
50         {
51             // Comando de liga inversor
52             serialPort1.Write("a");
53
54         }
55
```

```
56          private void Btn_Desliga_Click(object sender, EventArgs e)
57          {
58              // Comando de desliga inversor
59              serialPort1.Write("b");
60          }
61
62          private void Btn_LocRem_Click(object sender, EventArgs e)
63          {
64              serialPort1.Write("d");
65          }
66
67          private void button2_Click(object sender, EventArgs e)
68          {
69              serialPort1.Write("c");
70          }
71
72          private void fecharToolStripMenuItem_Click(object sender, EventArgs e)
73          {
74              Close();
75              serialPort1.Close();
76          }
77
78
79
80          private void button3_Click(object sender, EventArgs e)
81          {
82              // Ativa temporaizador
83              timer1.Enabled = true;
84              timer2.Enabled = true;
85
86              // Tratamento de excecao
87              if (comm == serialPort1.PortName)
88              {
89                  // Abre porta Serial
90                  serialPort1.Open();
91                  // Habilita comando
92                  Btn_Liga.Enabled = true;
93                  Btn_Desliga.Enabled = true;
94                  Btn_LocRem.Enabled = true;
95                  Btn_Reset.Enabled = true;
96                  btn_Salvar.Enabled = true;
97                  checkBox1.Enabled = true;
98              }
99              else
100             {
101                 MessageBox.Show("Selecionar a Porta Correta");
102             }
103
104
105         }
106
107
108         private void button4_Click(object sender, EventArgs e)
109         {
110             serialPort1.Close();
111             // Desabilita comando
112             Btn_Liga.Enabled = false;
113             Btn_Desliga.Enabled = false;
114             Btn_LocRem.Enabled = false;
115             Btn_Reset.Enabled = false;
```

```
116             checkBox1.Enabled = false;
117             btn_Salvar.Enabled = false;
118         }
119
120
121         private void timer1_Tick(object sender, EventArgs e)
122
123         {
124             // Declaracao de variaveis locais
125             string tempVal;
126
127             if (serialPort1.IsOpen)
128             {
129                 tempVal = serialPort1.ReadExisting();
130                 if (tempVal != "")
131                 {
132
133                     cont++;
134                     Grid.Rows.Add(tempVal, cont);
135
136                 }
137
138             }
139
140         }
141
142         private void button5_Click(object sender, EventArgs e)
143         {
144             SaveFileDialog salvar = new SaveFileDialog(); // novo
145
146
147             // Salvando o arquivo
148
149             Excel.Application App; // Aplicacao Excel
150             Excel.Workbook WorkBook; // Pasta
151             Excel.Worksheet WorkSheet; // Planilha
152             object misValue = System.Reflection.Missing.Value;
153
154             App = new Excel.Application();
155             WorkBook = App.Workbooks.Add(misValue);
156             WorkSheet = (Excel.Worksheet)WorkBook.Worksheets.get_Item(1);
157             int i = 0;
158             int j = 0;
159
160             // passa as celulas do DataGridView para a Pasta do Excel
161             for (i = 0; i <= Grid.RowCount - 1; i++)
162             {
163                 for (j = 0; j <= Grid.ColumnCount - 1; j++)
164                 {
165                     DataGridViewCell cell = Grid[j, i];
166                     WorkSheet.Cells[i + 1, j + 1] = cell.Value;
167                 }
168             }
169
170             // define algumas propriedades da caixa salvar
171             salvar.Title = "Exportar para Excel";
172             salvar.Filter = "Arquivo do Excel *.xls | *.xls";
173             // mostra o arquivo
174             salvar.ShowDialog();
175
```

```
176             // salva o arquivo
177             WorkBook.SaveAs(salvar.FileName, Excel.XlFileFormat.xlWorkbookNormal, misValue,
178             misValue, misValue, misValue,
179
180             Excel.XlSaveAsAccessMode.xlExclusive, misValue, misValue, misValue,
181             misValue, misValue);
182             WorkBook.Close(true, misValue, misValue);
183             App.Quit(); // encerra o excel
184
185             MessageBox.Show("Exportado com sucesso!");
186
187         }
188
189         private void cOM1ToolStripMenuItem_Click(object sender, EventArgs e)
190         {
191             // Porta serial COM1 selecionada
192             serialPort1.PortName = "COM1";
193             Btn_Open.Enabled = true;
194             Btn_Close.Enabled = true;
195
196         }
197
198         private void cOM2ToolStripMenuItem_Click(object sender, EventArgs e)
199         {
200             // Porta serial COM2 selecionada
201             serialPort1.PortName = "COM2";
202             Btn_Open.Enabled = true;
203             Btn_Close.Enabled = true;
204         }
205
206         private void cOM3ToolStripMenuItem_Click(object sender, EventArgs e)
207         {
208             // Porta serial COM3 selecionada
209             serialPort1.PortName = "COM3";
210             Btn_Open.Enabled = true;
211             Btn_Close.Enabled = true;
212         }
213
214         private void cOM4ToolStripMenuItem_Click(object sender, EventArgs e)
215         {
216             // Porta serial COM4 selecionada
217             serialPort1.PortName = "COM4";
218             Btn_Open.Enabled = true;
219             Btn_Close.Enabled = true;
220
221         }
222
223         private void cOM5ToolStripMenuItem_Click(object sender, EventArgs e)
224         {
225             // Porta serial COM5 selecionada
226             serialPort1.PortName = "COM5";
227             Btn_Open.Enabled = true;
228             Btn_Close.Enabled = true;
229
230
231         }
232
233         private void bpsToolStripMenuItem_Click(object sender, EventArgs e)
234         {
235             // Taxa baud rate 9600bps
```

```
236             serialPort1.BaudRate = 9600;
237
238         }
239
240         private void bpsToolStripMenuItem1_Click(object sender, EventArgs e)
241         {
242             // Taxa baud rate 4800bps
243             serialPort1.BaudRate = 4800;
244
245         }
246
247         private void bpsToolStripMenuItem2_Click(object sender, EventArgs e)
248         {
249             // Taxa baud rate 2400bps
250             serialPort1.BaudRate = 2400;
251         }
252
253         private void bpsToolStripMenuItem3_Click(object sender, EventArgs e)
254         {
255             // Taxa baud rate 1200bps
256             serialPort1.BaudRate = 1200;
257         }
258
259         private void button5_Click_1(object sender, EventArgs e)
260         {
261             {
262
263                 SaveFileDialog salvar = new SaveFileDialog(); // novo
264
265
266                 // Salvando o arquivo
267
268                 Excel.Application App; // Aplicacao Excel
269                 Excel.Workbook WorkBook; // Pasta
270                 Excel.Worksheet WorkSheet; // Planilha
271                 object misValue = System.Reflection.Missing.Value;
272
273                 App = new Excel.Application();
274                 WorkBook = App.Workbooks.Add(misValue);
275                 WorkSheet = (Excel.Worksheet)WorkBook.Worksheets.get_Item(1);
276                 int i = 0;
277                 int j = 0;
278
279                 // passa as celulas do DataGridView para a Pasta do Excel
280                 for (i = 0; i <= Grid.RowCount - 1; i++)
281                 {
282                     for (j = 0; j <= Grid.ColumnCount - 1; j++)
283                     {
284                         DataGridViewCell cell = Grid[j, i];
285                         WorkSheet.Cells[i + 1, j + 1] = cell.Value;
286                     }
287                 }
288
289                 // define algumas propriedades da caixa salvar
290                 salvar.Title = "Exportar para Excel";
291                 salvar.Filter = "Arquivo do Excel *.xls | *.xls";
292                 salvar.FileName = "Log_" + DateTime.Now.ToString("ddMMyyyy_HHmmss");
293                 salvar.ShowDialog(); // mostra
294
295                 // salva o arquivo
```

```
296                 WorkBook.SaveAs(salvar.FileName, Excel.XlFileFormat.xlWorkbookNormal,
297                 misValue, misValue, misValue, misValue,
298
299                 Excel.XlSaveAsAccessMode.xlExclusive, misValue, misValue, misValue,
300                 misValue, misValue);
301                 WorkBook.Close(true, misValue, misValue);
302                 App.Quit(); // encerra o excel
303
304                 MessageBox.Show("Exportado com sucesso!");
305
306             }
307         }
308
309         private void timer2_Tick(object sender, EventArgs e)
310         {
311
312             if (checkBox1.Checked == true)
313             {
314                 SaveFileDialog salvar = new SaveFileDialog(); // novo
315
316
317             // Salvando o arquivo
318
319             Excel.Application App; // Aplicacao Excel
320             Excel.Workbook WorkBook; // Pasta
321             Excel.Worksheet WorkSheet; // Planilha
322             object misValue = System.Reflection.Missing.Value;
323
324             App = new Excel.Application();
325             WorkBook = App.Workbooks.Add(misValue);
326             WorkSheet = (Excel.Worksheet)WorkBook.Worksheets.get_Item(1);
327             int i = 0;
328             int j = 0;
329
330             // passa as celulas do DataGridView para a Pasta do Excel
331             for (i = 0; i <= Grid.RowCount - 1; i++)
332             {
333                 for (j = 0; j <= Grid.ColumnCount - 1; j++)
334                 {
335                     DataGridViewCell cell = Grid[j, i];
336                     WorkSheet.Cells[i + 1, j + 1] = cell.Value;
337                 }
338             }
339
340                 // define algumas propriedades da caixa salvar
341                 salvar.Filter = "Arquivo do Excel *.xls | *.xls";
342                 salvar.FileName = "Log_" + DateTime.Now.ToString("ddMMyyyy_HHmmss");
343                 // salva o arquivo
344                 WorkBook.SaveAs  = "Log_" + DateTime.Now.ToString("ddMMyyyy_HHmmss");
345
346                 WorkBook.SaveAs(salvar.FileName,
347                 Excel.XlFileFormat.xlWorkbookNormal, misValue,
348                 misValue, misValue, misValue,
349
350
351                 Excel.XlSaveAsAccessMode.xlExclusive, misValue, misValue,
352                 misValue, misValue, misValue);
353                 WorkBook.Save();
354
355                // WorkBook.Close(true, misValue, misValue);
```

```
356                     App.Quit(); // encerra o excel
357
358                     MessageBox.Show("Backup efetuado com Sucesso!");
359                 }
360                 else
361                     MessageBox.Show("Operacao cancelada!");
362             }
363         }
364 }
```