UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO DE JOINVILLE
CURSO DE ENGENHARIA MECATRÔNICA

GUILHERME TURATTO

DESENVOLVIMENTO DE UMA BANCADA EXPERIMENTAL PARA ESTUDOS DE EMULSÕES ÁGUA-ÓLEO

Joinville
2022

GUILHERME TURATTO

DESENVOLVIMENTO DE UMA BANCADA EXPERIMENTAL PARA ESTUDOS DE EMULSÕES ÁGUA-ÓLEO

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica no curso de Engenharia Mecatrônica, da Universidade Federal de Santa Catarina, Centro Tecnológico de Joinville.

Orientador: Dr. Anderson Wedderhoff Spengler

Joinville
2022

RESUMO

Escoamentos multifásicos entre água e óleo são encontrados em diversos setores da indústria. Nas indústrias petroquímicas, esses escoamentos ocorrem em condições adversas de operação, como alta pressão, vibrações e ambiente corrosivo. Identificar e prever os fenômenos envolvidos nesse tipo escoamento é crucial para o controle, segurança e eficiência dos processos produtivos. Técnicas convencionais de medição monofásica implicam em custo e espaço adicionais, que são fatores proibitivos em grande parte das aplicações. Nesse caso, são utilizados instrumentos de medição multifásica, possibilitando a medição de diferentes variáveis de cada uma das fases simultaneamente. Este trabalho visa desenvolver um ambiente de testes que emule as condições de operação de tais equipamentos de medição nas indústrias petroquímicas, proporcionando a criação de emulsões e diferentes padrões de escoamento entre água e óleo crude além da aquisição de dados das principais variáveis envolvidas nesse processo, como vazão, pressão e temperatura. Para isso, serão levantados os requisitos do sistema, servindo de base para selecionar os equipamentos necessários. O sistema mecânico, hardware e software serão desenvolvidos, realizando a integração entre os equipamentos e possibilitando o controle e aquisição de dados dos experimentos em um computador de controle através do software LabVIEW.

Palavras-chave: Instrumentação Multifásica. Aquisição de Dados. Protocolos de Comunicação. LabVIEW.

ABSTRACT

Multiphase flows between water and oil are found in many sectors of industry. In petrochemical industries, these flows occur under adverse operating conditions, such as high pressure, vibrations and corrosive environment. Identifying and predicting the phenomena involved in this type of flow is crucial for the control, safety and efficiency of production processes. Conventional single-phase measurement techniques imply additional cost and space, which are prohibitive factors in most applications. In this case, multiphase measuring instruments are used, allowing the measurement of different variables of each phase simultaneously. This work aims to develop a test environment that emulates the operating conditions of such measuring equipment in petrochemical industries, providing the creation of emulsions and different flow patterns between water and crude oil in addition to data acquisition from the main variables involved in this process, such as flow rate, pressure and temperature. For this, the system requirements will be surveyed, serving as a basis for selecting the necessary equipment. The mechanical system, hardware and software will be developed, performing the integration between the equipment and enabling the control and data acquisition of the experiments in a control computer through LabVIEW software.

Keywords: Multiphase Instrumentation. Data Acquisition. Communication Protocols. LabVIEW.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

W/O	Emulsão de água em óleo
O/W	Emulsão de óleo em água
Bb	Padrão de escoamento com bolhas dispersas
SG	Padrão de escoamento pistonado
ST	Padrão de escoamento estratificado
AN	Padrão de escoamento anular
DC	Padrão de escoamento dual e contínuo
QUM	Quantity Under Measurement
S	Sensor
A	Amplifier
SCF	Signal Conditioning Filter
DSO	Digital Storage Oscilloscope
ATR	Analog Tape Recorder
AF	Anti-Aliasing Filter
ADC	Analog-to-Digital Converter
DAC	Digital-to-Analog Converter
SNR	Signal-to-Noise-Ratio
LSB	Least Significant Bit
MSB	Most Significant Bit
RTD	Resistance Temperature Detector
OSI	Open System Interconnection
PDU	Protocol Data Unit
ADU	Application Data Unit

RTU	Remote Terminal Unit
BIPM	Bureau International des Poids et Mesures
ASCII	American Standard Code for Information Interchange
CRC	Cyclic Redundancy Check
UART	Universal Asynchronous Receiver-Transmitter
bps	Bits per second
IC	Integrated Circuit
I2C	Inter-Integrated Circuit
SDA	Serial data line
SCL	Serial clock line
MCU	Microcontroller Unit
RAM	Random Access Memory
SRAM	Static Random Access Memory
EEPROM	Electrically-Erasable Programmable Read-Only Memory
LCD	Liquid Crystal Display
CMOS	Complementary Metal-Oxide-Semiconductor
NMOS	Negatively Doped Metal Oxide Semiconductor
BJT	Bipolar Junction Transistor
ACK	Acknowledge
NACK	Not Acknowledge
SR	Start Repeated
RFxx	Requisito Funcional
RNFxx	Requisito Não-Funcional
ARM	Advanced RISC Machine
TTL	Transistor-Transistor Level
USB	Universal Serial Bus

AWG	American Wire Gauge
ISO	International Organization for Standardization
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
T2F	Thermal Fluid Flow Group

LISTA DE SÍMBOLOS

ε_n	Erro absoluto da enésima medição	
y_n	Valor da enésima medição	
y_0	Valor real do mensurando	
A_n	Exatidão percentual da enésima medição	
P_n	Precisão percentual da enésima medição	
f_p	Função de densidade de probabilidade	
σ	Desvio padrão	
μ	Média	
$s(y)$	Desvio padrão amostral	
$\overline{y}$	Média amostral	
$s(\overline{y})$	Desvio padrão da média amostral	
N	Número de amostras	
δx	Incremento infinitesimal no mensurando	
Δx^R	Resolução do sistema de medição	
x^s	Sinal analógico	
x^s_{min}	Nível mínimo do sinal analógico	
x^s_{max}	Nível máximo do sinal analógico	
ΔX^{LSB}	Incremento mínimo no sinal digital	
$Nbits$	Número de bits utilizados no processo de quantização	
K	Sensibilidade do sistema de medição	
Δx^L	Limiar do sistema de medição	
$\vec{v}$	Velocidade	$[\mathrm{m/s}]$
l	Comprimento	$[\mathrm{m}]$

ΔU	Diferença de potencial	[V]
$\vec{B}$	Densidade de fluxo magnético	[$\mathrm{Wb/m^2}$]
ΔU_{EE}	Diferença de potencial entre eletrodos	[V]
$\overline{v}$	Velocidade média	[$\mathrm{m/s}$]
D	Diâmetro	[m]
Q_v	Vazão volumétrica	[$\mathrm{m^3/s}$]
Q_m	Vazão mássica	[$\mathrm{kg/s}$]
ρ	Densidade	[$\mathrm{kg/m^3}$]
ρ_r	Densidade relativa	
$\vec{f_c}$	Força Coriolis	[N]
$\vec{v_r}$	Velocidade relativa	[$\mathrm{m/s}$]
$\vec{\omega}$	Velocidade angular	[$\mathrm{rad/s}$]
m	Massa	[kg]
ΔP	Diferença de pressão	[kPa]
δc	Deflexão	[m]
R	Raio	[m]
φ	Espessura	[m]
Υ	Módulo de Young	
ν	Coeficiente de Poisson	
R	Resistência elétrica	[Ω]
V_{DD}	Tensão de alimentação	[V]
V_H	Faixa de tensão do nível lógico alto	[V]
V_L	Faixa de tensão do nível lógico baixo	[V]
R_{13bits}	Referência de frequência em 13 bits	
R_{Hz}	Referência de frequência em Hertz	
$Q^{Nom.}_{m\ Coriolis}$	Vazão nominal de operação do medidor de vazão Coriolis	[$\mathrm{kg/s}$]

$i_{Coriolis}$	Leitura em corrente do medidor de vazão Coriolis	[A]
$Q_{m\ Coriolis}$	Leitura convertida do medidor de vazão coriolis	[kg/s]
$i_{Eletromag}$	Leitura em corrente do medidor de vazão eletromagnético	[A]
$Q^{agua}_{m\ Eletromag}$	Leitura convertida do medidor de vazão eletromagnético para escoamentos de água	[kg/s]
$Q^{oleo}_{m\ Eletromag}$	Leitura convertida do medidor de vazão eletromagnético para escoamentos de óleo	[kg/s]
P_{PX409}	Leitura convertida dos transdutores de pressão PX409	[kPa]
i_{PX409}	Leitura em corrente dos transdutores de pressão PX409	[A]
S_A	Leituras em corrente antes da correção do Bias	[A]
S^*_A	Leituras em corrente após a correção do Bias	[A]

SUMÁRIO

1 INTRODUÇÃO

O escoamento simultâneo de dois ou mais fluidos imiscíveis na mesma tubulação é conhecido como escoamento multifásico e está presente em diversos processos industriais, sendo encontrados em indústrias farmacêuticas, alimentícias e petroquímicas (FAGHRI; ZHANG, 2020). Nessa última, são encontrados escoamentos do tipo água-óleo-gás em condições adversas de operação, como alta pressão, vibrações e ambiente corrosivo. A capacidade de medir e prever o comportamento dos fenômenos envolvidos nesse tipo escoamento é crucial para o controle, segurança e eficiência dos processos produtivos (BRENNEN, 2005).

Devido à natureza do processo de extração de hidrocarbonetos, como petróleo e gás natural, são gerados padrões complexos de escoamentos multifásicos nas linhas de produção. A caracterização desses fenômenos requer a medição de um grande número de variáveis, como vazão, fração e temperatura de cada uma das fases. Nas plataformas de extração, a separação de fases é comumente utilizada, aplicando métodos tradicionais de medição monofásica, apesar de exigir equipamentos e infraestrutura adicionais (FALCONE et al., 2009). O custo e espaço necessários para implementação da separação de fases são fatores proibitivos em determinadas aplicações, sendo necessária a utilização de instrumentos de medição multifásica (BAKER, 2016).

De acordo com Falcone et al. (2009), as vantagens obtidas ao utilizar instrumentos de medição multifásica na indústria de óleo e gás são observadas em diversos setores. No layout das instalações de produção, reduzindo os equipamentos necessários para a produção e minimizando o espaço utilizado pelas plataformas de extração. Nos testes de poços, evitando o uso de separadores de fase, os quais são onerosos por requererem manutenção regular e despenderem muito tempo até atingirem regime permanente. No monitoramento da produção e gerenciamento das reservas, fornecendo informações contínuas e em tempo real aos operadores. Na alocação de produção, identificando rapidamente mudanças na produtividade e na composição dos fluidos. E nos custos, reduzindo a infraestrutura necessária e otimizando decisões administrativas.

Tendo em vista a importância dessa classe de instrumentos de medição e os benefícios operacionais, econômicos e ambientais de sua utilização, se faz necessário o estudo de novas técnicas e abordagens para ampliar a faixa de operação, aumentar a precisão e exatidão além de reduzir o tamanho e custo de tais equipamentos. Para isso, se faz necessário um sistema que emule as condições de operação em escala laboratorial, possibilitando diferentes configurações e testes reprodutíveis para

validação das tecnologias desenvolvidas (BAKER, 2016).

Com objetivo de proporcionar esse ambiente de testes e contribuir com os avanços no estudo da instrumentação e controle desses fenômenos, este trabalho visa o desenvolvimento de uma bancada experimental que permita o controle das variáveis envolvidas na formação dos escoamentos multifásicos e emulsões entre água e óleo. Com a bancada em funcionamento será possível gerar diferentes padrões de escoamento e níveis de emulsão, regulando a vazão, pressão e turbulência do escoamento.

Para gerar uma vazão controlada em cada uma das fases, a bancada contará com duas motobombas acionadas através de inversores de frequência. Sensores de vazão, pressão e temperatura serão disponibilizados em cada um dos ramais antes da emulsão. Para armazenamento dos fluidos três tanques serão utilizados, um para água, um para óleo e outro para decantação.

Após a união dos fluxos de água e óleo, o escoamento passará por uma válvula de controle eletropneumática, alterando a emulsão e o padrão de escoamento de acordo com seu nível de abertura. Uma seção de testes, construída com tubulação de acrílico, será disponibilizada após a válvula. Por fim, todo o sistema será integrado a um computador de controle, fornecendo ao pesquisador uma interface unificada para utilização da bancada. Todo o trabalho será realizado com apoio do Laboratório de pesquisa Thermal Fluid Flow Group (T2F), do Centro Tecnológico de Joinville, da Universidade Federal de Santa Catarina (UFSC), campus Joinville.

1.1 OBJETIVOS

Para avançar os estudos na área de instrumentação multifásica, propõe-se neste trabalho os seguintes objetivos.

1.1.1 Objetivo geral

Proporcionar um ambiente de testes que emule condições de operação de equipamentos de instrumentação multifásica nos processos encontrados em indústrias petroquímicas.

1.1.2 Objetivos específicos

- Definir os requisitos e características de operação necessárias;
- Construir os sistemas mecânico e eletrônico que compõem a bancada;
- Desenvolver os softwares para controle e aquisição de dados.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 ESCOAMENTOS MULTIFÁSICOS

O fluido resultante da extração de poços de petróleo raramente é aquele de interesse em sua forma pura. Normalmente, a extração resulta em um padrão complexo de escoamento multifásico. Em um caso ideal, o escoamento deve ser composto unicamente pelos hidrocarbonetos de óleo crude de gás natural. Contudo, em diversos sistemas de extração, água e fases sólidas como areia, hidratos e asfaltenos também estão presentes (FALCONE et al., 2009).

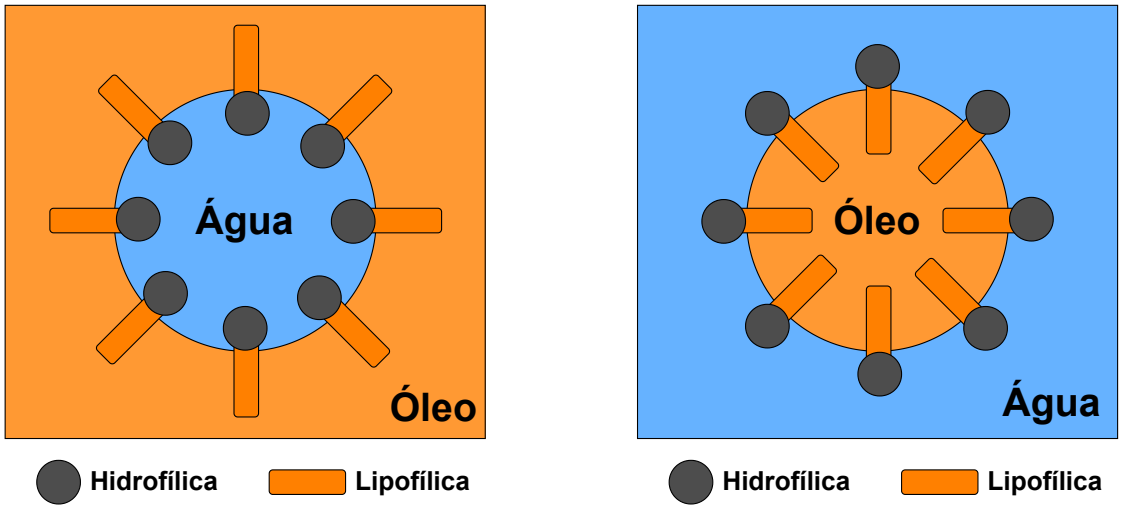
O escopo deste trabalho se limita à abordagem de escoamentos multifásicos compostos por água e óleo. Tais escoamentos são parte fundamental na modelagem do processo produtivo e podem ocorrer em forma de emulsões de água em óleo (W/O) ou óleo em água (O/W). O escoamento resultante das emulsões tem sua viscosidade e densidade elevadas em relação às características originais do óleo crude, tornando-o mais mais pesado e dificultoso de ser transportado pelas linhas de produção (FINGAS, 2014).

2.1.1 Emulsões água-óleo

De acordo com Wong, Lim e Dol (2015), o estudo e caracterização das emulsões é crucial para compreender a natureza dos escoamentos multifásicos entre óleo e água. Apesar de ambas as fases estarem inicialmente separadas, a ação da turbulência e agitação geradas pelas bombas, válvulas e tubulações da linha de produção causam a emulsificação. O escoamento emulsificado pode ser caracterizado como uma união de dois líquidos imiscíveis, onde uma das fases está dispersa em formas de glóbulos na fase contínua.

Segundo Khan et al. (2011), a presença natural de agentes emulsificantes na composição do óleo crude favorece a emulsificação. Tais compostos, como asfaltenos e resinas, formam um filme envolto nos glóbulos da fase dispersa, impedindo sua coalescência. Os asfaltenos são os principais estabilizantes das emulsões, enquanto as resinas agem como solventes, diluindo os asfaltenos (FINGAS, 2014). Normalmente os componentes emulsificantes possuem em sua estrutura molecular uma componente hidrofílica, que tende a ser atraída pela água, e outra lipofílica, atraída pelo óleo. A Figura 1a apresenta a ação dos agentes emulsificantes em uma emulsão W/O, enquanto a Figura 1b representa emulsões O/W.

Figura 1 – Ação dos componentes emulsificantes.



(a) Gotícula de água dispersa em óleo.

(b) Gotícula de óleo dispersa em água.

Fonte: adaptado de Khan et al. (2011).

As gotículas da fase dispersa são injetadas na fase contínua por meio da turbulência e podem ser estabilizadas temporariamente pela viscosidade dos fluidos. A estabilização de longo prazo ocorre com a migração dos asfaltenos solúveis para a interface entre água e óleo nas gotículas, dada pela difusão. Na interface, as moléculas de asfaltenos se agregam, formando um filme viscoelástico e estável. Tais emulsões afetam a qualidade do óleo crude, diminuem a vida útil dos equipamentos devido à corrosão, e dificultam o emprego de equipamentos de medição multifásica, motivando o estudo de novas técnicas de medição (FINGAS, 2014).

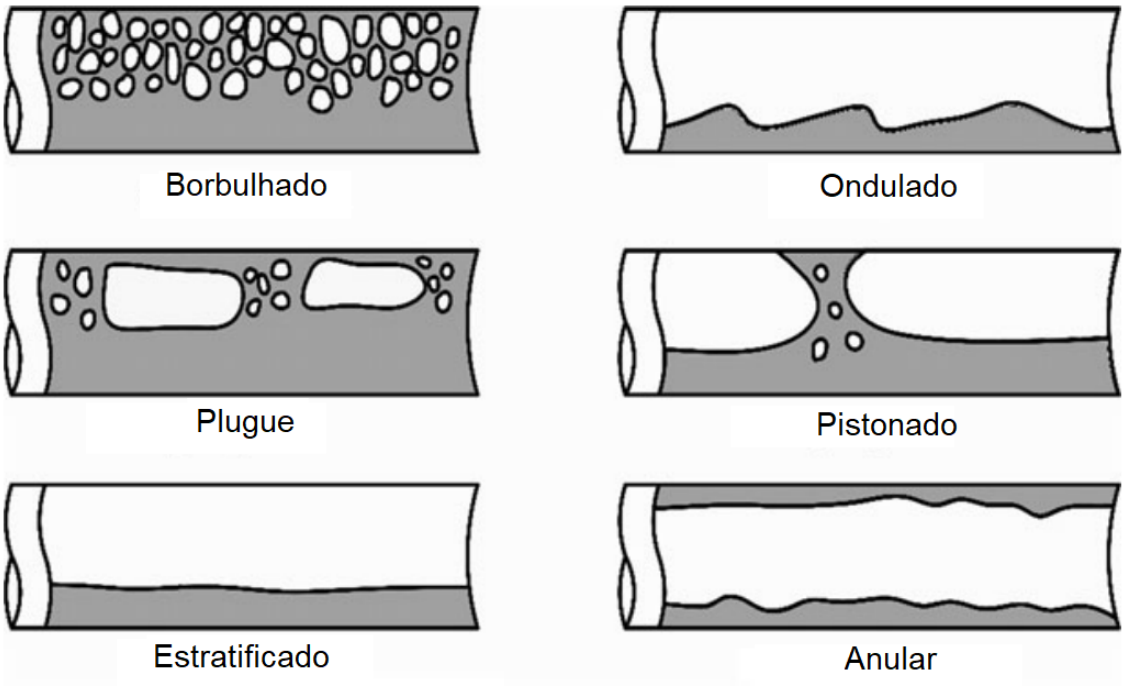
2.1.2 Padrões de escoamentos

O formato e comportamento da interface entre as fases de uma mistura multifásica é o que determina o padrão ou regime de escoamento. De acordo com Falcone et al. (2009), os principais fatores que determinam o padrão de escoamento são:

- Propriedades das fases, como densidade e composição;
- Fração e velocidade de cada fase;
- Temperatura e pressão de operação;
- Diâmetro, formato, inclinação e rugosidade da tubulação;
- Presença de válvulas, juntas ou curvas;

Escoamentos bifásicos em dutos horizontais apresentam padrões assimétricos em relação ao eixo horizontal, pois a gravidade faz com que a fase de maior densidade ocupe a região inferior do duto. Nesse tipo de escoamento são encontrados seis principais padrões de escoamento: bolhas dispersas, ondulado, plugue, pistonado, estratificado e anular (Figura 2).

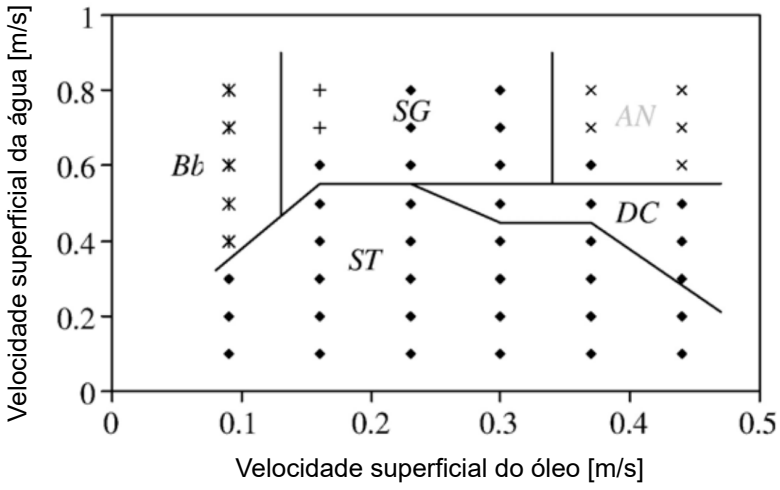
Figura 2 – Padrões de escoamentos bifásicos em dutos horizontais.



Fonte: adaptado de Faghri e Zhang (2020).

A classificação dos regimes de escoamento pode ocorrer experimentalmente, através da observação de escoamentos multifásicos controlados. Mapeamentos em duas dimensões podem ser gerados através dos resultados obtidos, representando os padrões observados em função da variação dos parâmetros (WAHAIBI et al., 2007).

Figura 3 – Mapeamento dos padrões de escoamento.



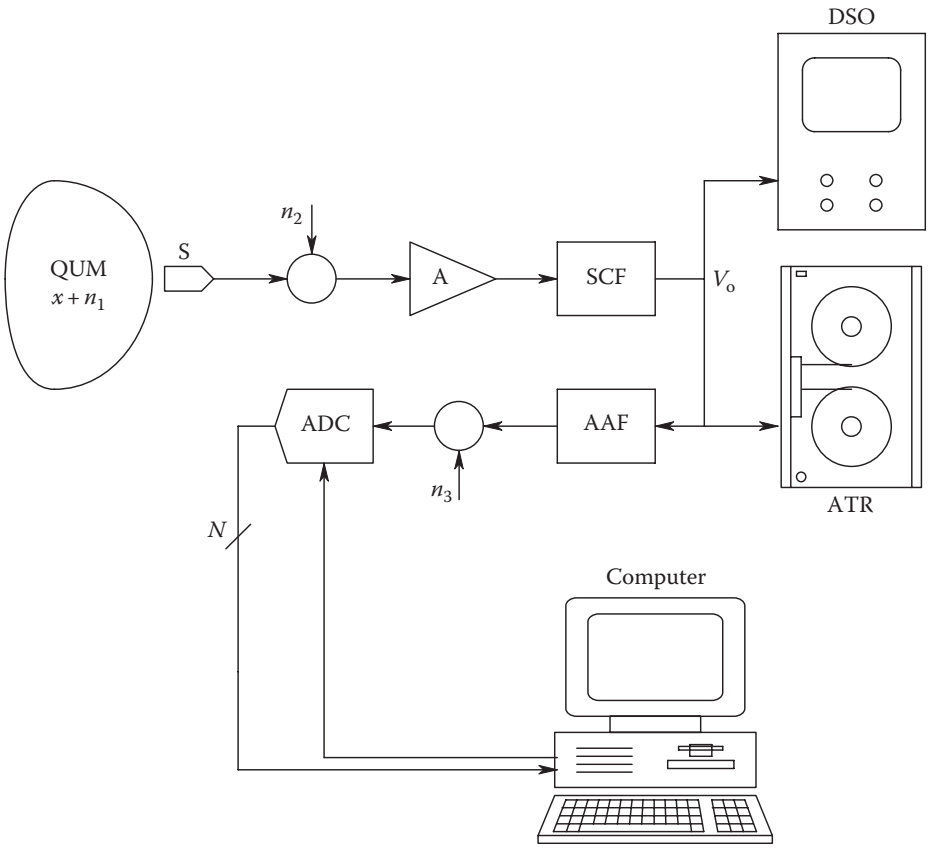
Fonte: adaptado de Wahaibi et al. (2007).

Comumente, os parâmetros controlados são a fração e velocidade das fases, suficientes para alterar o padrão de escoamento. Na Figura 3, é apresentado o mapeamento obtido por Wahaibi et al. (2007) com um escoamento bifásico entre água e óleo. Onde Bb representa bolhas dispersas, SG pistonado, ST estratificado ou ondulado, AN anular e DC dual e contínuo.

2.2 INSTRUMENTAÇÃO

Sistemas de medição são tipicamente utilizados para mensurar grandezas físicas e elétricas, como temperatura, velocidade, pressão, corrente, tensão, capacitância, entre outros (NORTHROP, 2014). Geralmente, em âmbito industrial, são utilizados em conjunto com sistemas de controle ou monitoramento de processos. Segundo Aguirre (2013), o funcionamento dos sistemas de instrumentação pode ser descrito em termos de seus subsistemas e respectivas funções. A Figura 4 apresenta uma visão geral desses sistemas.

Figura 4 – Visão geral de um sistema de medição. QUM = x = Quantity Under Measurement. S = Sensor. A = Amplifier. SCF = Signal Conditioning Filter (analógico). DSO = Digital Storage Oscilloscope. ATR = Analog Tape Recorder. AF = Anti-Aliasing Filter (analog). ADC = Analog-to-Digital Converter. n1 = Ruído que acompanha o QUM. n2 = Ruído inerente da eletrônica. n3 = Ruído de quantização.



Fonte: Northrop (2014).

Conforme Figura 4, a grandeza a ser medida, ou QUM, é convertida pelo sensor ou transdutor (S) para uma forma de energia conveniente, como tensão ou corrente. O sinal é então amplificado e filtrado por um subsistema de condicionamento (SCF), a fim de ajustar a impedância de saída e melhorar o SNR (Signal-to-Noise Ratio). Após condicionado, o sinal (aqui assumido ser um sinal de tensão V_o) pode ser exibido em displays ou lido e armazenado por osciloscópios (DSO e ATR). Comumente,

o sinal analógico V_o é enviado a um filtro passa-baixa anti-aliasing (AAF), que previne a ocorrência de aliasing durante a amostragem e conversão para um sinal digital pelo ADC (NORTHROP, 2014).

A saída do ADC é um sinal digital, discretizado no tempo e na amplitude. O intervalo entre cada amostra é definido pela frequência de amostragem e a resolução é limitada pelo número de bits utilizados no processo de quantização do sinal analógico. Por fim, o sinal digital pode ser utilizado por um computador ou microcontrolador, onde as leituras são tratadas e utilizadas para fins de controle e aquisição de dados (PELGROM, 2021).

Segundo Northrop (2014), existem três principais fontes de ruídos no típico sistema de medição apresentado na Figura 4. A primeira delas (n_1) se refere ao ruído inerente à natureza das medições, também chamado de medições espúrias. São fenômenos atrelados àquele que se deseja medir e que causam uma variação na medição que não corresponde a uma alteração na variável de interesse (AGUIRRE, 2013). Em sequência, a fonte de ruído n_2 representa os ruídos inseridos no sistema de condicionamento do sinal, provenientes de interferências e induções eletromagnéticas. Por fim, em n_3 é inserido o ruído causado pelo processo de conversão do sinal analógico para digital, definido como ruído de quantização (PELGROM, 2021).

2.2.1 Qualidade de um sistema de medição

A qualidade das medições pode ser quantificada em termos de precisão e exatidão, que são definidas através de conceitos estatísticos e estão relacionadas a erros aleatórios e sistemáticos, respectivamente. A resolução, sensibilidade e limiar dos instrumentos de medição utilizados também são características fundamentais para avaliar o desempenho de um sistema de aquisição de dados (AGUIRRE, 2013).

2.2.1.1 Precisão e Exatidão

Como resultado de uma medição tem-se um provável valor para o mensurando (QUM) associado a uma incerteza, inerente ao processo de medição apresentado acima. Para analisar as medições de modo probabilístico é adequado empregar o conceito de função de densidade de probabilidade $f_p(y)$, a qual define a probabilidade de uma observação y estar no intervalo $[a\ b]$:

$$P(a \leq y \leq b) = \int_a^b f_p(y) d_y \tag{1}$$

Considerando que a densidade de probabilidade seja dada pela função gaussiana apresentada pela Equação 2, $f_p(y)$ será caracterizada pela média μ e desvio padrão σ:

$$f_p(y) = \frac{1}{\sigma\sqrt{2\pi}}\exp\left(-\frac{1}{2\sigma^2}(y-\mu)^2\right) \tag{2}$$

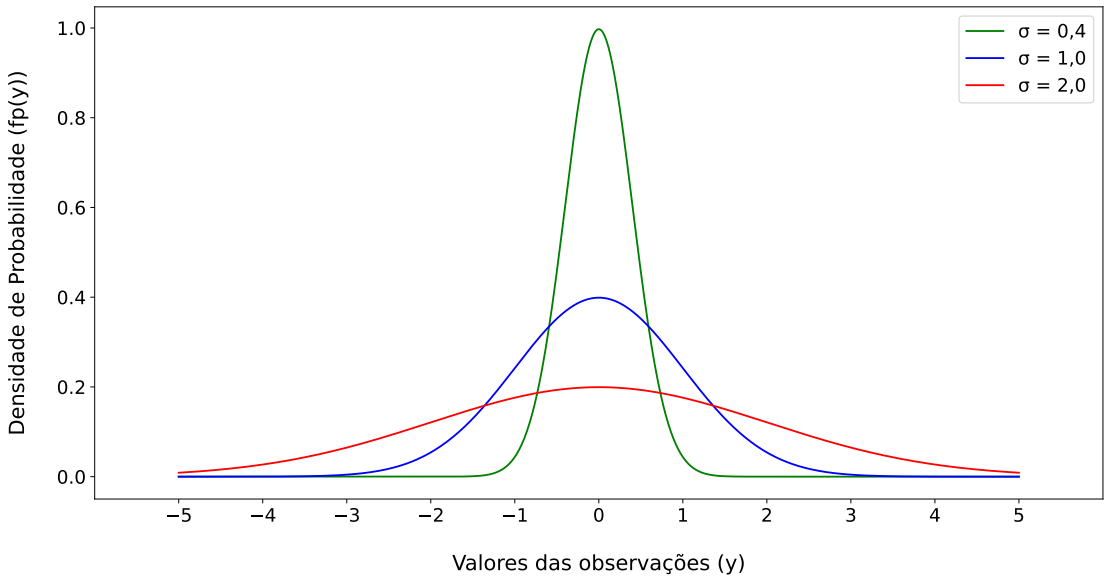
Sendo que μ e σ podem ser aproximados pela média $\overline{y}$ e desvio padrão $s(y)$ amostrais de um conjunto de observações $y = \{y_1, y_2, ...y_N\}$ contendo N amostras:

$$\mu \approx \overline{y} = \frac{1}{N}\sum_{n=1}^{N} y_n \tag{3}$$

$$\sigma \approx s(y) = \sqrt{\frac{1}{N-1}\sum_{n=1}^{N}(y_n - \overline{y})^2} \tag{4}$$

A Figura 5 apresenta o comportamento da função gaussiana para diferentes valores de desvio padrão. Nota-se que a média indica o valor no eixo das abscissas onde a função estará centralizada, e o desvio padrão está relacionado com a dispersão das amostras.

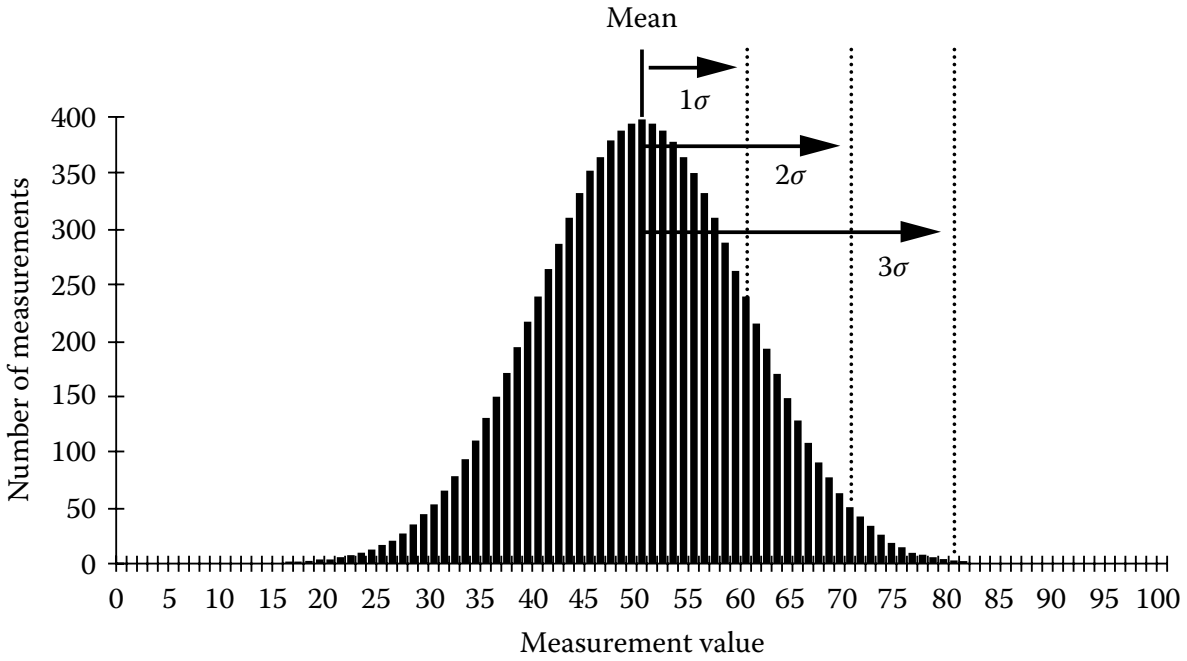
Figura 5 – Função gaussiana com $\mu = 0$ e $\sigma = 0{,}4$, $\sigma = 1{,}0$ e $\sigma = 2{,}0$.



Fonte: autor (2022).

De acordo com Aguirre (2013), a função de densidade de probabilidade dos instrumentos de medição geralmente não é conhecida. Contudo, um histograma gerado a partir dos valores de N medições se aproxima das características de f_p conforme $N \to \infty$. Segundo Webster e Eren (2018), tal histograma teria a forma da Figura 6, onde 68% das medições estariam do intervalo de $\mu \pm 1\sigma$; 95% em $\mu \pm 2\sigma$; e 99,7% estariam em $\mu \pm 3\sigma$.

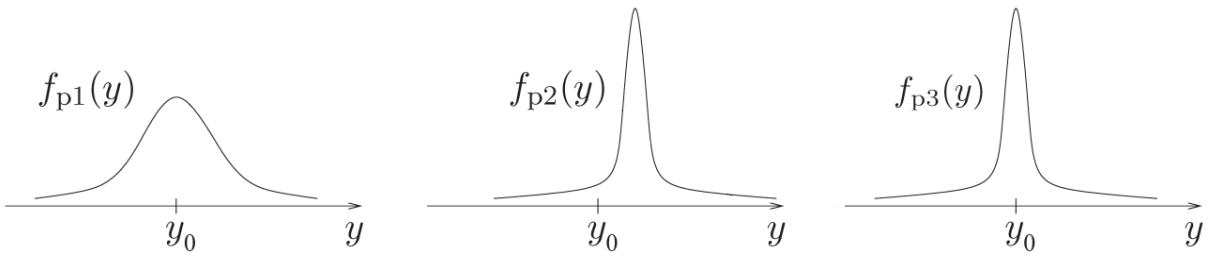
Figura 6 – Histograma gerado a partir de um conjunto de amostras.



Fonte: Webster e Eren (2018).

Supondo que, ao medir simultaneamente a mesma grandeza física com valor real y_0, a f_p de 3 sensores fosse conhecida e apresentada pela Figura 7.

Figura 7 – Comparando a densidade de probabilidade de diferentes sensores.



Fonte: adaptado de Aguirre (2013).

A densidade de probabilidade do primeiro sensor $f_{p1}(y)$ é centrada em $\mu = y_0$, ou seja, a leitura mais provável é o próprio valor real da variável medida. Contudo, sua dispersão é grande, indicando que, em um conjunto de amostras, os valores observados estarão dispersos em torno do valor real. Sendo um sensor exato, porém impreciso. A $f_{p2}(y)$ indica que o segundo sensor possui uma precisão maior que o anterior, pois sua dispersão é menor. Por outro lado, é inexato ou tendencioso, tendo sua f_p centrada em um valor diferente do real. O sensor 3, com f_{p3} pouco dispersa e centrada no valor de referência, demonstra ser mais exato e preciso, características desejadas em um instrumento de medição (AGUIRRE, 2013).

Os erros em um sistema de medição podem ser classificados em duas componentes, uma aleatória e outra sistemática (Figura 7). A componente aleatória

está atrelada à precisão do sensor e aos efeitos imprevisíveis das entradas espúrias e demais ruídos. Essa componente afeta o desvio padrão amostral e implica em variações nas medições sequenciais do mensurando. De acordo com Aguirre (2013), ao dispor de um conjunto de N medições, pode-se utilizar a média amostral $\overline{y}$ como estimativa para o valor real do mensurando em lugar de utilizar apenas uma das N amostras. Sendo assim, a estimativa do desvio padrão σ que antes seria $s(y)$ agora passa a ser $\sigma \approx s(\overline{y})$, onde:

$$s(\overline{y}) = \frac{s(y)}{\sqrt{N}} \tag{5}$$

Portando, tomando a média de N amostras como resultado do processo de medição melhora-se a qualidade das medidas em termos de erros aleatórios, pois $s(\overline{y}) < s(y) \quad \forall \quad y = \{y_1, y_2, ..., y_N\}$.

Já a componente sistemática do erro tem origem em fatores que alteram a relação entre a entrada e saída do instrumento de medição, assim como na correlação previsível entre as entradas espúrias e o mensurando. Erros sistemáticos causam o deslocamento da função de densidade de probabilidade, gerando um offset entre as medidas e o valor real. Pode-se amenizar esta parcela do erro calibrando o instrumento de medição ou subtraindo o offset observado na média das leituras, caso o valor real do mensurando seja conhecido.

O erro da enésima medição pode ser definido como:

$$\varepsilon_n = y_n - y_0 \tag{6}$$

Onde y_n é o valor medido e y_0 o valor real — dado analiticamente ou por um instrumento calibrado e certificado de acordo com um padrão reconhecido, como o BIPM (Bureau International des Poids et Mesures), por exemplo.

Northrop (2014) define a exatidão A_n (Accuracy) e precisão P_n percentuais da enésima amostra como sendo:

$$A_n = \left(1 - \left|\frac{y_n - y_0}{y_0}\right|\right) 100 \tag{7}$$

$$P_n = \left(1 - \left|\frac{y_n - \overline{y}}{\overline{y}}\right|\right) 100 \tag{8}$$

Por fim, o resultado de um processo de medição é dado pela estimativa do valor real do mensurando associado a uma faixa de incerteza. Para a estimativa, pode-se

utilizar a média de N medições, reduzindo o desvio padrão por um fator de $1/\sqrt{N}$, como visto na Equação 5. Segundo Aguirre (2013), a incerteza padrão do Tipo A, é dada pelo desvio padrão da média $s(\overline{y})$. Sendo assim, a estimativa do mensurando é dada, com 95% de confiabilidade (ver Figura 6), na forma:

$$y_0 \approx \frac{1}{N}\sum_{n=1}^{N} y_n \pm 2\frac{\sqrt{\frac{1}{N-1}\sum_{n=1}^{N}(y_n - \overline{y})^2}}{\sqrt{N}}$$

Ou:

$$y_0 \approx \overline{y} \pm 2s(\overline{y}) \tag{9}$$

2.2.1.2 Resolução

De acordo com Northrop (2014), a resolução de um sistema de medição pode ser descrita como o menor incremento do mensurando que pode ser detectado com fidelidade. Aguirre (2013) reforça que este conceito deve ser aplicado somente quando o valor medido for diferente de 0, descartando a possibilidade de a medida estar abaixo do limiar do sensor.

Iniciando com uma medição não nula $y_1 \approx x_1 \neq 0$, ao adicionar incrementos infinitesimais δx no mensurando, o sistema de medição não apresentará alteração no valor medido até que o acréscimo acumulado seja maior que um determinado valor Δx^R. Ao ultrapassar esse valor crítico, a saída será atualizada para $y_2 \approx x_1 + \Delta x^R$. Esse mínimo incremento detectável Δx^R é a resolução do sistema de medição.

A limitação da resolução ocorre em 3 principais pontos: no próprio instrumento de medição, no processo de conversão do sinal analógico para digital e no menor incremento possível na escala de exibição. A limitação no instrumento é devido aspectos construtivos, princípio físico utilizado pelo sensor e qualidade dos componentes utilizados. Na escala de exibição, a resolução é limitada pela metade do menor dígito exibido (AGUIRRE, 2013).

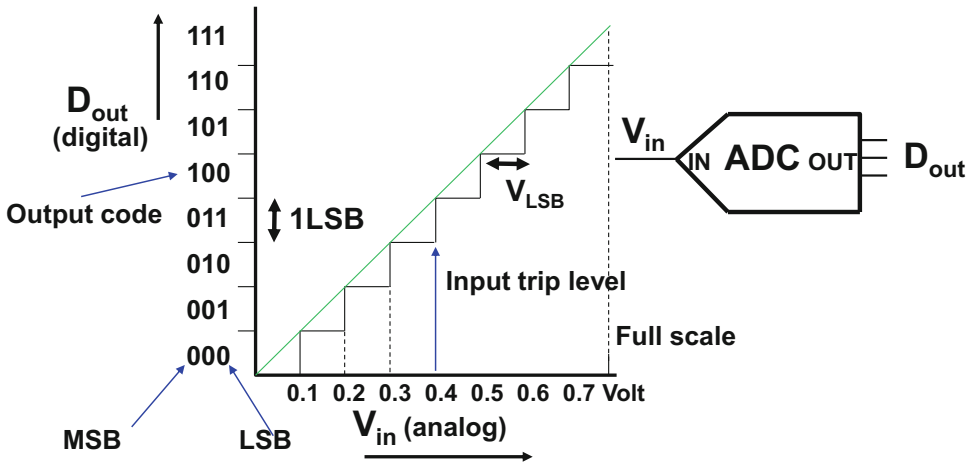
No processo de conversão, a limitação se dá pelos erros de quantização, de acordo com a faixa de medição e o número de bits utilizados para representar digitalmente o sinal analógico x^s. O ADC possui apenas 2^{Nbits} valores para representar toda a faixa de medição, sendo $Nbits$ o número de bits. Para isso, a excursão do sinal analógico é quantificada em 2^{Nbits} patamares. Dessa forma, a resolução pode ser expressa em termos do LSB (Least Significant Bit), ou seja, a variação mínima do sinal analógico que gera uma mudança no bit menos significativo do sinal digital (PELGROM, 2021).

Portanto, o incremento mínimo Δx^{LSB} do sinal digital apresentado na saída do ADC corresponde à diferença entre dois níveis de quantização consecutivos. Podendo ser escrito na forma:

$$\Delta x^{LSB} = \frac{x^s_{max} - x^s_{min}}{2^{Nbits}} \tag{10}$$

Onde $x^s_{max} - x^s_{min}$ representa a faixa de excursão do sinal analógico, que é proporcional ao mensurando.

Figura 8 – Parâmetros da conversão de um sinal analógico para digital.



Fonte: Pelgrom (2021).

A Figura 8 exemplifica a quantização de um sinal analógico em uma faixa de $0,0\mathrm{V}$ a $0,8\mathrm{V}$ utilizando 3 bits. O sinal analógico é, portanto, representado em 8 níveis de quantização, sendo $\Delta x^{LSB} = (0,8 - 0,0)/2^3 = 0,1\mathrm{V}$.

2.2.1.3 Sensibilidade

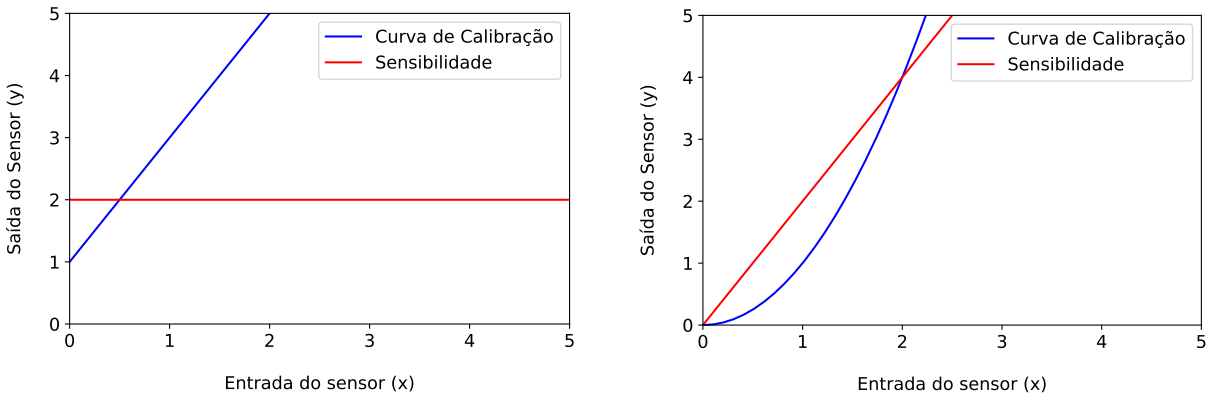
A sensibilidade de um instrumento de medição corresponde à taxa de variação da saída em relação à variação do mensurando, expressa em unidade da variável física de saída por unidade da variável de entrada (AGUIRRE, 2013). A sensibilidade em um medidor de temperatura que tem como saída um sinal analógico de corrente, por exemplo, pode ser descrita como $\mathrm{mA}/\mathrm{^\circ C}$.

Experimentalmente, pode-se obter a sensibilidade K pela divisão entre a variação da saída Δy pela variação entrada Δx após a estabilização das leituras, ou seja:

$$K = \frac{\Delta y}{\Delta x} \tag{11}$$

A sensibilidade pode variar de acordo com o ponto de operação, a depender da curva de resposta ou curva de calibração do instrumento. Analiticamente, a sensibilidade é caracterizada pela derivada da curva de calibração, sendo igual para todos os pontos de operação apenas se a relação entre entrada e saída for linear (AGUIRRE, 2013).

Figura 9 – Sensibilidade de acordo com a curva de calibração.



(a) $y_a = 2x + 1$ (b) $y_b = x^2$

Fonte: autor (2022).

Como apresentado na Figura 9a, a sensibilidade se apresenta constante em toda a faixa de operação, pois $K = dy_a(x)/dx = 2$. Contudo, no caso onde a curva de calibração é não-linear (Figura 9b), a sensibilidade é $K = dy_b(x)/dx = 2x$, variando de acordo com o valor da medição.

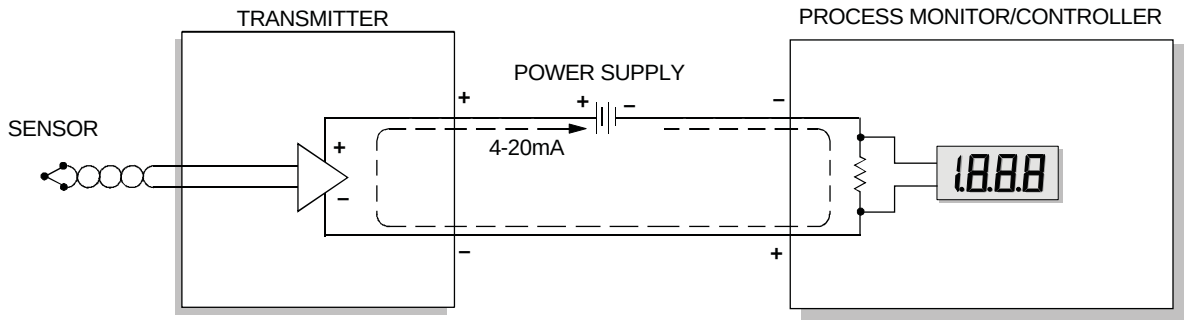
2.2.1.4 Limiar

O limiar de um sistema de medição indica o nível mínimo do mensurando, partindo de um valor nulo, que pode ser detectado (WEBSTER; EREN, 2018). Ou seja, a leitura permanece com valor nulo enquanto o mensurando não atingir o limiar Δx^L. Sendo assim, $y = 0 \ \forall \ x < \Delta x^L$ (AGUIRRE, 2013).

2.2.2 Loops de corrente

Loops de corrente são amplamente utilizados para transmissão de sinais analógicos em sistemas de monitoramento e controle de processos industriais (MURATA, 2015). Com um baixo custo de implementação, imunidade a ruídos e capacidade de transmissão a longas distâncias, os loops de corrente são adequados para aplicações em ambientes industriais e estão disponíveis em uma série de instrumentos de medição, tais como medidores de pressão, vazão e temperatura (NATIONAL INSTRUMENTS, 2022).

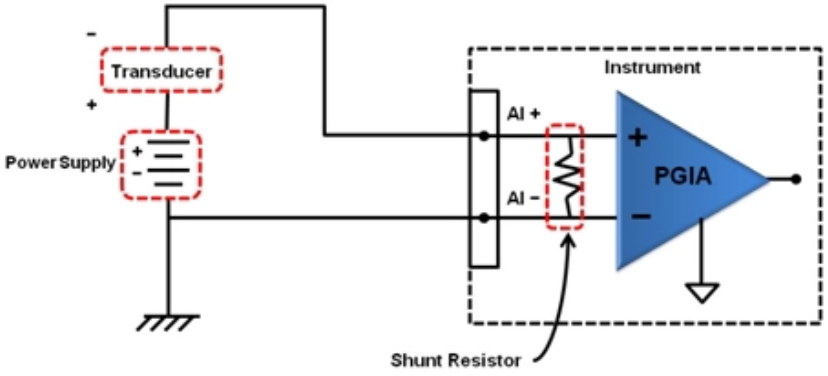
Figura 10 – Componentes de um loop de corrente.



Fonte: Murata (2015).

Seguindo a topologia da Figura 10, sensores utilizados em aplicações industriais se beneficiam da utilização de loops de corrente por não enfrentarem a atenuação do sinal devido à queda de tensão ao longo cabos pela resistência dos fios (NATIONAL INSTRUMENTS, 2022). De acordo com a Lei das Correntes de Kirchhoff, a soma das correntes que chegam a um nó do circuito deve ser igual a soma das correntes que partem daquele mesmo nó (NILSSON; RIEDEL, 2016). Portanto, em um loop de corrente, todas as correntes presentes no início da malha devem atingir o ponto final.

Figura 11 – Topologia padrão de um loop de corrente.

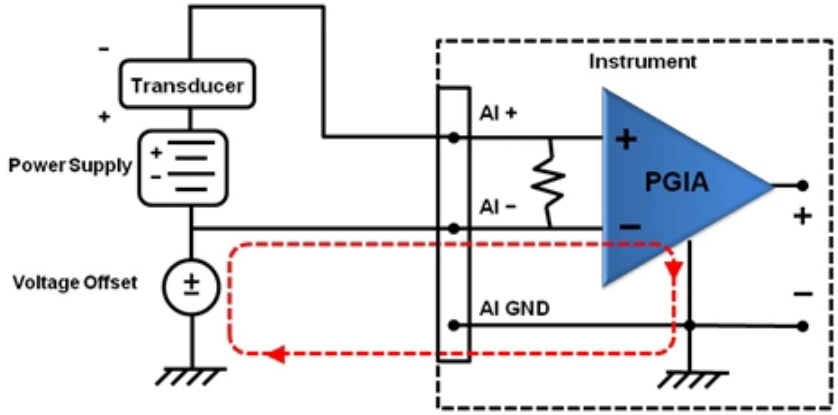


Fonte: National Instruments (2022).

Os principais componentes de uma malha de corrente incluem o transdutor, um sistema de aquisição de dados, uma fonte de alimentação em corrente contínua e o cabeamento ligando-os em série, conforme Figura 11. A fonte de alimentação fornece a potência para o sistema. O transdutor regula a corrente que circula pela malha, onde $4\,\mathrm{mA}$ representam o valor 0 do mensurando e $20\,\mathrm{mA}$ indicam o fundo de escala da medição. Assim, é possível a detecção de problemas no cabeamento caso a leitura de corrente seja nula. Por fim, o sistema de aquisição de dados realiza a leitura da medição, que pode ser efetuada através da conversão do sinal de corrente para um sinal de tensão utilizando um resistor de precisão entre os terminais do amplificador de entrada do sistema (NATIONAL INSTRUMENTS, 2022).

O projeto de um sistema de aquisição de dados utilizando malhas de corrente deve considerar pontos importantes para a integração dos componentes. A escolha do transdutor deve atender as especificações de leitura desejadas, tais como grandeza física a ser medida, faixa de operação, precisão e tensão de alimentação. Em seguida, a fonte deve atender a tensão de alimentação mínima e potência exigidas pelo transdutor, assim como não ultrapassar a tensão máxima permitida, sob risco de danificar o instrumento de medição. O nível de tensão mínimo deve ser atingido mesmo com as quedas de tensão ocasionadas pela resistência dos cabos da malha e pelo resistor de precisão.

Figura 12 – Correntes inseridas por loop de terra.



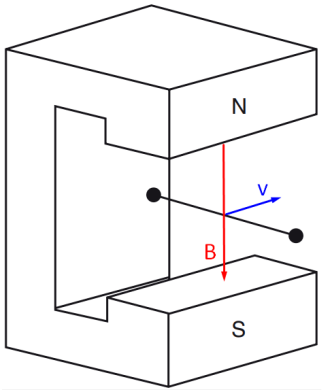
Fonte: National Instruments (2022).

O sistema de aquisição de dados desempenha um papel fundamental, afetando diretamente a qualidade e confiabilidade das medições. Segundo National Instruments (2022), é importante evitar loops de terra, os quais geram erros de offset nas leituras devido à inserção de corrente na malha, causada por possíveis diferenças de potencial entre os pontos de referência do transdutor e do sistema de aquisição (Figura 12). Este problema é contornado através da equipotencialização do sistema de aterramento ou por isolação, separando eletricamente as referências do transdutor e do amplificador do sistema de medição, podendo ser realizada através optoacopladores ou isolação galvânica (NATIONAL INSTRUMENTS, 2022).

2.2.3 Medidor de vazão eletromagnético

Essa classe de instrumentos de medição baseia-se na indução eletromagnética definida pela Lei de Faraday. Ao mover com velocidade constante $\vec{v}$ um condutor de comprimento l através de um campo com densidade de fluxo magnético $\vec{B}$, uma diferença de potencial $\Delta U = (\vec{v} \times \vec{B})l$ é induzida entre as extremidades do condutor, onde $\times$ indica o produto vetorial. Podendo ser expressa como $\Delta U = vBl$ caso a velocidade seja perpendicular ao campo (Figura 13).

Figura 13 – Um condutor em movimento através de um campo magnético.



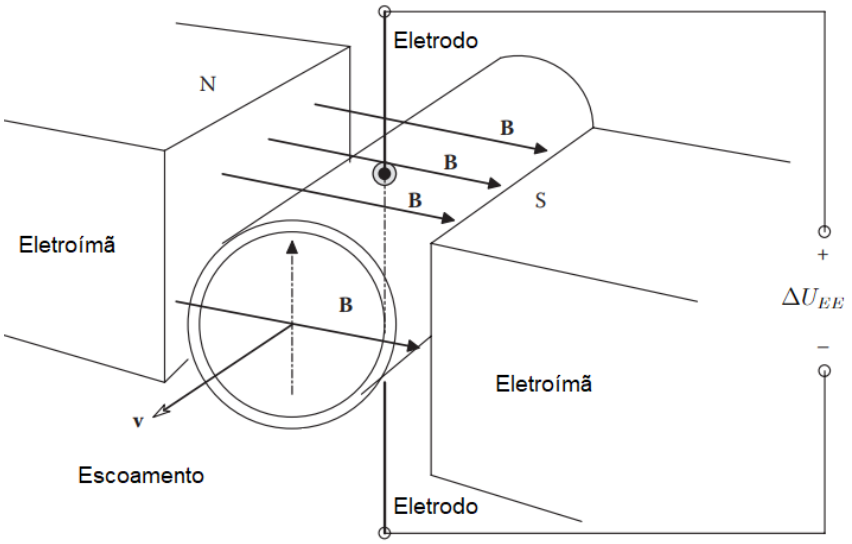
Fonte: adaptado de Baker (2016).

Em um medidor de vazão eletromagnético, o campo magnético é gerado através de eletroímãs e a indução de tensão ocorre nos elementos de um fluido condutor, que escoa em um tubo circular com isolação elétrica na parte interna. A detecção da tensão induzida é realizada por eletrodos, posicionados nas extremidades do duto. Segundo Baker (2016), os filamentos de fluido, onde ocorre a indução, abrangem a seção transversal entre um eletrodo e outro. Sendo assim, podem mover-se com velocidades e posições diferentes, causando variações na tensão induzida total. Com isso, a equação de operação, que determina a tensão entre os eletrodos ΔU_{EE}, é dada por:

$$\Delta U_{EE} = \overline{v} B D \tag{12}$$

Onde $\overline{v}$ é a velocidade média do fluido, em metros por segundo, B é a densidade de fluxo magnético, em Tesla, e D é o diâmetro do tubo, em metros.

Figura 14 – Diagrama dos componentes de um medidor de vazão eletromagnético.



Fonte: adaptado de Northrop (2014).

A Figura 14 representa a visão geral de um medidor de vazão eletromagnético, onde os eletrodos são posicionados nas extremidades do duto, perpendicularmente ao campo magnético e ao escoamento. Como a densidade de fluxo e área interna da tubulação ($\pi D^2/4$) são parâmetros conhecidos do instrumento, a tensão obtida pode ser relacionada com a velocidade e, portanto, com a vazão volumétrica do escoamento (em metros cúbicos por segundo):

$$Q_v = D\pi\frac{\Delta U_{EE}}{4B} \tag{13}$$

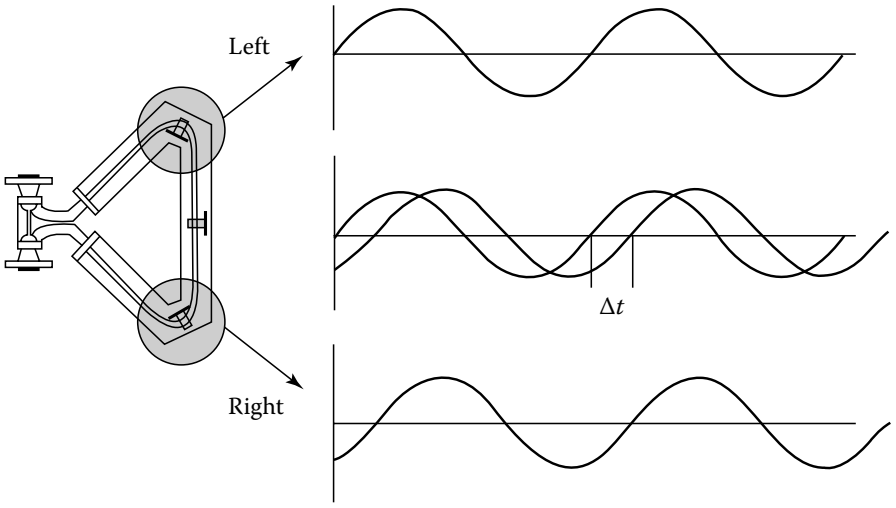
2.2.4 Medidor de vazão Coriolis

Segundo Aguirre (2013), será exercida uma força de Coriolis $\vec{fc}$ a um objeto com massa m que desloca-se com velocidade $\vec{v_r}$ em relação à um referencial de velocidade $\vec{\omega}$:

$$\vec{fc} = 2m(\vec{\omega} \times \vec{v_r}) \tag{14}$$

Fazendo com que o escoamento passe por um tubo em formato de U, os medidores por efeito Coriolis geram vibrações na tubulação através de atuadores eletromagnéticos. O perfil das vibrações é conhecido quando não há escoamento. A força de Coriolis, agindo sobre os elementos de um fluido em movimento, gera torque e, consequentemente, uma torção no tubo. Afetando a amplitude e frequência das vibrações.

Figura 15 – Detecção da frequência e defasagem entre as vibrações em um medidor de vazão por efeito Coriolis.



Fonte: Webster e Eren (2018).

Sensores posicionados em ambos os lados do tubo em U detectam a frequência das vibrações e defasagem entre os formatos de onda obtidos (Figura 15). Com isso, é possível determinar a vazão mássica e densidade do fluido.

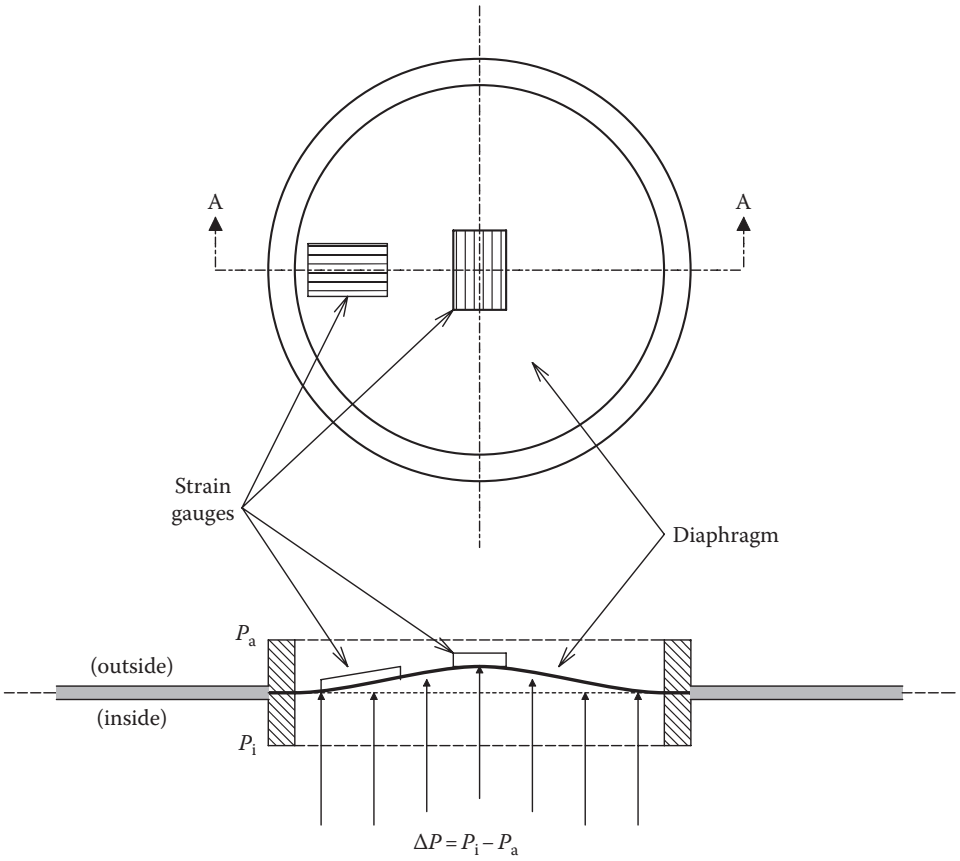
2.2.5 Transdutores de pressão

Transdutores de pressão normalmente utilizam a deflexão de um diafragma, causada pela diferença de pressão, para alterar as propriedades elétricas de um material e correlacionar o mensurando com um sinal elétrico. Segundo Northrop (2014), quando submetido a uma diferença de pressão, o diafragma exerce uma força de tensionamento em sua região central e compressão nas laterais. A deflexão da região central do diafragma pode ser aproximada por:

$$\delta c \approx \frac{3\Delta P R^4 (1-\nu^2)}{16 \Upsilon \varphi^3} \tag{15}$$

Onde ΔP é a diferença de pressão entre as superfícies do diafragma, R é o raio do diafragma, φ é a espessura, Υ o módulo de Young e ν é o coeficiente de Poisson do material.

Figura 16 – Diagrama de um transdutor de pressão baseado em deflexão de diafragma em conjunto com extensômetros.



Fonte: Northrop (2014).

Diversos métodos e materiais podem ser aplicados para obter um sinal elétrico a partir de uma deflexão, tais como extensômetros, sensores piezo-resistivos e materiais piezoelétricos (BAKER, 2016). A Figura 16 exemplifica o uso de extensômetros para tal medição.

2.2.6 Termorresistências

A variação da resistência elétrica de certos materiais de acordo com a temperatura é utilizada como princípio de medição das termorresistências (AGUIRRE, 2013). Uma classe desses dispositivos são os RTDs (Resistance Temperature Detector), que são construídos a partir de um filamento metálico envolto em um núcleo de cerâmica ou vidro, protegidos por uma sonda.

De acordo com Northrop (2014), a resistência dos metais normalmente aumenta com a temperatura, apresentando uma relaçã não-linear que pode ser aproximada pelas equações de Callendar-Van Dusen, apresentadas nas Equações 16 e 17, que são utilizadas para temperaturas negativas e positivas, respectivamente.

$$R(T) = R_0 \left[1 + A \times T + B \times T^2 + C \times T^3 \times (T - 100\ ℃)\right] \tag{16}$$

$$R(T) = R_0 \left[1 + A \times T + B \times T^2\right] \tag{17}$$

Onde T é a temperatura em graus Celsius, $R(T)$ é a resistência elétrica do material em tal temperatura, R_0 corresponde a resistência do material na temperatura de 0 ℃ e A, B e C, são coeficientes que dependem do material utilizado. Diferentes materiais e respectivas temperaturas de referência são encontrados em sensores comerciais, um exemplo é o PT-100, que apresenta resistência de $100\,\Omega$ a uma temperatura de 0 ℃.

2.3 PROTOCOLOS DE COMUNICAÇÃO

Protocolos de comunicação são amplamente utilizados em qualquer sistema que exija troca de informação entre componentes de hardware ou software. Segundo Kurose e Ross (2013), protocolos de comunicação podem ser aplicados em diferentes níveis da informação, controlando desde o fluxo de bits até a taxa com que os pacotes de dados são transmitidos entre a origem e o destino.

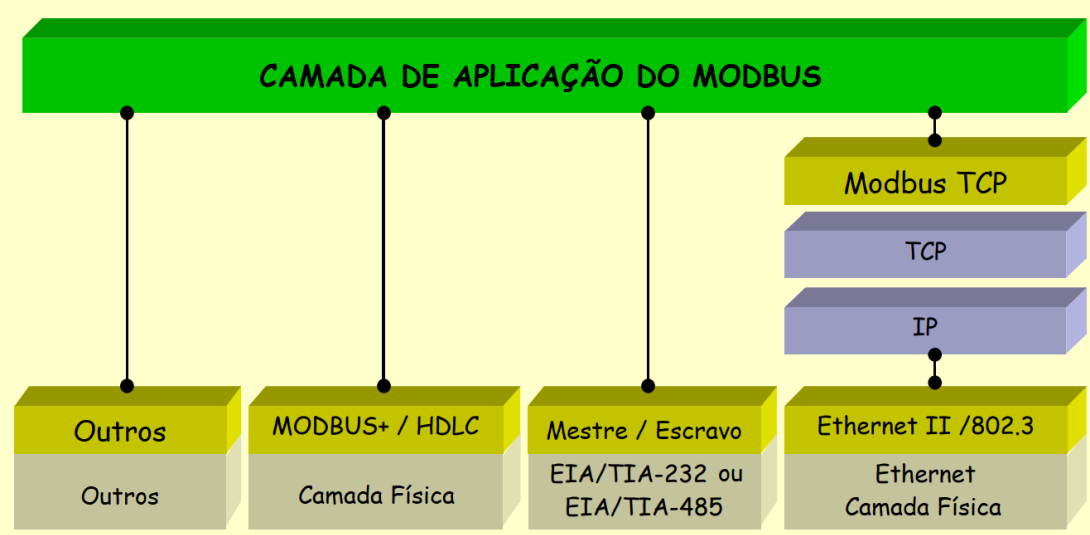
Um protocolo tem a função de definir o formato e a ordem das mensagens trocadas entre duas entidades, assim como as ações e comportamentos esperados na transmissão e no recebimento das mensagens (KUROSE; ROSS, 2013). Atualmente, inúmeros protocolos são disponibilizados em um mesmo dispositivo, a fim de garantir

a troca de informação e interoperabilidade entre os sistemas. Nesta seção serão apresentados os principais protocolos de comunicação utilizados no desenvolvimento deste trabalho.

2.3.1 Modbus

O Modbus é um protocolo de mensagens da camada de aplicação, posicionada na sétima camada do modelo OSI (Open System Interconnection), o qual define uma estrutura de cliente e servidor entre dispositivos. A conexão entre eles pode ocorrer através de diferentes redes ou barramentos da camada física, como por exemplo o RS-485, RS-232 e Ethernet 802.3, conforme ilustrado pela Figura 17 (MODBUS, 2012).

Figura 17 – Pilha de comunicação Modbus.



Fonte: adaptado de (MODBUS, 2012).

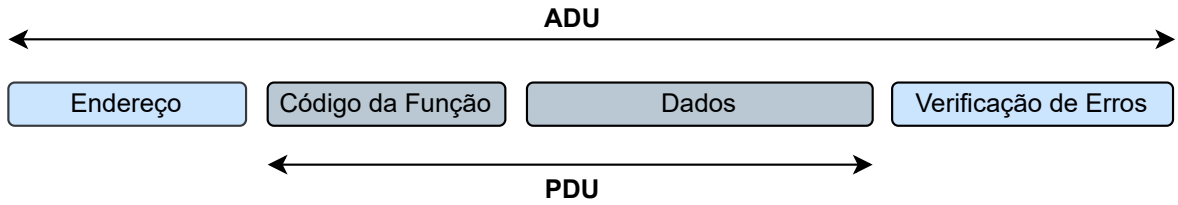
Operando no modelo requisição e resposta, esse protocolo especifica *códigos de funções*, que são enviados juntamente com as mensagens e definem a operação desejada (leitura, escrita, tipo de dado a ser manipulado, entre outros). Em redes industriais, esse protocolo é comumente utilizado com a estrutura mestre-escravo, onde existe apenas um cliente fazendo as requisições e vários servidores disponibilizando serviços.

Segundo Modbus (2012), a estrutura de mensagens definida pelo protocolo Modbus é subdividida em 4 partes: endereço, código da função, dados e verificação de erros (Figura 18). O código da função e os dados podem ser categorizados como PDU (Protocol Data Unit), sendo a parte da mensagem que é independente das camadas de comunicação subjacentes. Já a mensagem como um todo, contendo as 4 partes, é chamada de ADU (Application Data Unit), e pode sofrer alterações nos campos de endereçamento e verificação de erros de acordo com a camada física utilizada.

Existem dois modos de transmissão das mensagens em uma rede Modbus, o ASCII (American Standard Code for Information Interchange) e o RTU (Remote Terminal

Unit). Operando no modo ASCII, cada byte de 8 bits na mensagem é codificado e enviado como sendo dois caracteres ASCII, tendo a vantagem de possibilitar intervalos de até um segundo entre os caracteres sem causar erros de transmissão. Já no modo RTU, cada byte é representado como dois caracteres hexadecimais de 4 bits cada um, tendo uma taxa de transferência de dados maior que o modo ASCII para a mesma taxa de transmissão (MODBUS, 2006).

Figura 18 – Estrutura geral das mensagens Modbus.



Fonte: adaptado de (MODBUS, 2012).

Neste trabalho será utilizado o modo de transmissão RTU e o padrão RS485 na camada física. Portanto, a especificação das mensagens feita a seguir será de acordo com esse modo de transmissão, podendo divergir da estrutura utilizada no modo ASCII.

De acordo com Modbus (2006), a transmissão de cada byte da mensagem no modo RTU ocorre como apresentado na Figura 19. Inicialmente é transmitido um *start bit*, indicando o inicio da mensagem, seguido pelo bit menos significativo daquele byte, ou LSB. Os demais bits vêm na sequência até chegar no oitavo e último bit, sendo este o mais significativo, ou MSB (Most Significant Bit). Por fim, são enviados dois bits ao fim da mensagem, podendo ser dois bits de parada (*stop bits*), ou um bit de parada e outro de paridade (parity bit), a depender da configuração realizada nos dispositivos da rede.

Figura 19 – Ordem dos bits no modo de transmissão RTU.

Com Verificação de Paridade

Start	1	2	3	4	5	6	7	8	Par	Stop

Sem Verificação de Paridade

Start	1	2	3	4	5	6	7	8	Stop	Stop

Fonte: adaptado de (MODBUS, 2006).

A paridade é uma técnica de verificação de erros que garante a integridade de cada byte transmitido. Se os dispositivos forem configurados para utilizá-la, um bit de paridade é inserido em cada byte das mensagens. Podendo ser configurada como

paridade par ou ímpar, o bit adicionado será ajustado para que o número de bits em nível lógico alto esteja de acordo com a paridade selecionada. No recebimento das mensagens, a cada byte é realizada a contagem do número de bits em nível lógico alto, caso a paridade seja diferente daquela configurada um erro de transmissão é detectado. Contudo, esse método não determina quais bits foram modificados, o byte normalmente é descartado e uma retransmissão é necessária (FOROUZAN, 2009).

O primeiro campo do ADU tem tamanho de 1 byte e indica o endereço do dispositivo a ser enviado a mensagem na rede Modbus. Caso o valor desse campo seja um endereço válido de 1 a 247, a comunicação será em modo unicast e a mensagem será recebida apenas pelo dispositivo que contém aquele endereço. Neste caso é esperado uma mensagem de resposta, onde o dispositivo insere seu próprio endereço neste campo. Porém, se o campo de endereço conter o valor 0 na mensagem de requisição, a transmissão será em modo broadcast e todos os dispositivos da rede irão recebê-la, sendo este modo reservado para operações de escrita e não havendo mensagem de resposta.

O segundo campo contém o código da função e também tem tamanho de 1 byte. Quando a mensagem é enviada do dispositivo cliente ao dispositivo servidor, o valor desse campo indica o tipo de ação a ser executada, como por exemplo, a leitura do diagnóstico do dispositivo, leitura do estado de um grupo de variáveis, escrita em registradores, entre outros. Na mensagem de resposta, o dispositivo insere o mesmo código da ação performada para indicar sucesso, ou o código da função alterado, com o bit mais significativo em nível lógico alto, para indicar erro. Como mostra a Tabela 1.

Tabela 1 – Código de Função nas Mensagens de Requisição e Resposta.

Tipo da Mensagem	Binário	Hexadecimal
Requisição	0000 0100	0x04
Resposta - Sucesso	0000 0100	0x04
Resposta - Erro	1000 0100	0x84

Fonte: autor (2022).

O terceiro campo do ADU, contendo os dados da mensagem, tem a finalidade definida de acordo com a função especificada, fornecendo informações adicionais para que os dispositivos performem a ação desejada. Em uma requisição de leitura de um grupo de registradores, por exemplo, o campo de dados pode indicar o endereço inicial e o número de registradores a serem lidos. Assim como em uma operação de escrita, onde o campo de dados indica o endereço do registrador e o valor a ser gravado.

Algumas funções, como a leitura do diagnóstico ou dados do fabricante, não requerem dados adicionais. Nesse caso, o campo de dados é vazio. Caso a operação ocorra com sucesso, os dispositivos inserem os dados requisitados no campo de dados da mensagem de resposta, se a operação for de leitura, ou replicam os dados adicionais enviados pelo cliente, nas operações de escrita. O protocolo Modbus não define um

número de bytes específico para o campo de dados, sendo esse tamanho definido de acordo com cada função e dispositivo.

O quarto e último campo do ADU tem a finalidade de garantir a integridade das mensagens transmitidas pela rede Modbus. Tendo em vista que suas aplicações normalmente se encontram em ambientes críticos, com altos níveis de ruídos e interferências, como no caso de ambientes fabris, a confiabilidade na rede de comunicação utilizada entre os dispositivos é essencial para a segurança e correto funcionamento dos sistemas.

Contendo dois bytes, o campo de verificação de erros do protocolo Modbus utiliza a técnica CRC16 (Cyclic Redundancy Check), com 16 bits de tamanho. Além de contar com a verificação de erros pela paridade dos bits, que pode ser opcionalmente utilizada, garantindo a integridade de cada byte, o CRC16 é calculado e aplicado a nível de toda a mensagem. Utilizando os demais campos do ADU e um algorítmo baseado em divisão polinomial, o CRC16 é calculado e adicionado ao fim da mensagem, garantindo que erros causados por mudanças acidentais no nível lógico dos bits da mensagem possam ser detectados.

No caso de mensagens de requisição, o dispositivo mestre realiza o cálculo do CRC16, e o adiciona ao fim da mensagem. O dispositivo ao qual a mensagem for entregue, após receber todos os bytes, realiza novamente o cálculo do CRC16 e o compara com o valor recebido no quarto campo do ADU. Caso ocorra alguma divergência entre os valores, a mensagem é descartada. Portanto, nenhuma ação é realizada e não é enviada uma mensagem de resposta. O dispositivo cliente, que iniciou a comunicação, fica responsável por aguardar um período determinado pela resposta do servidor e implementar a tratativa do erro em caso de timeout.

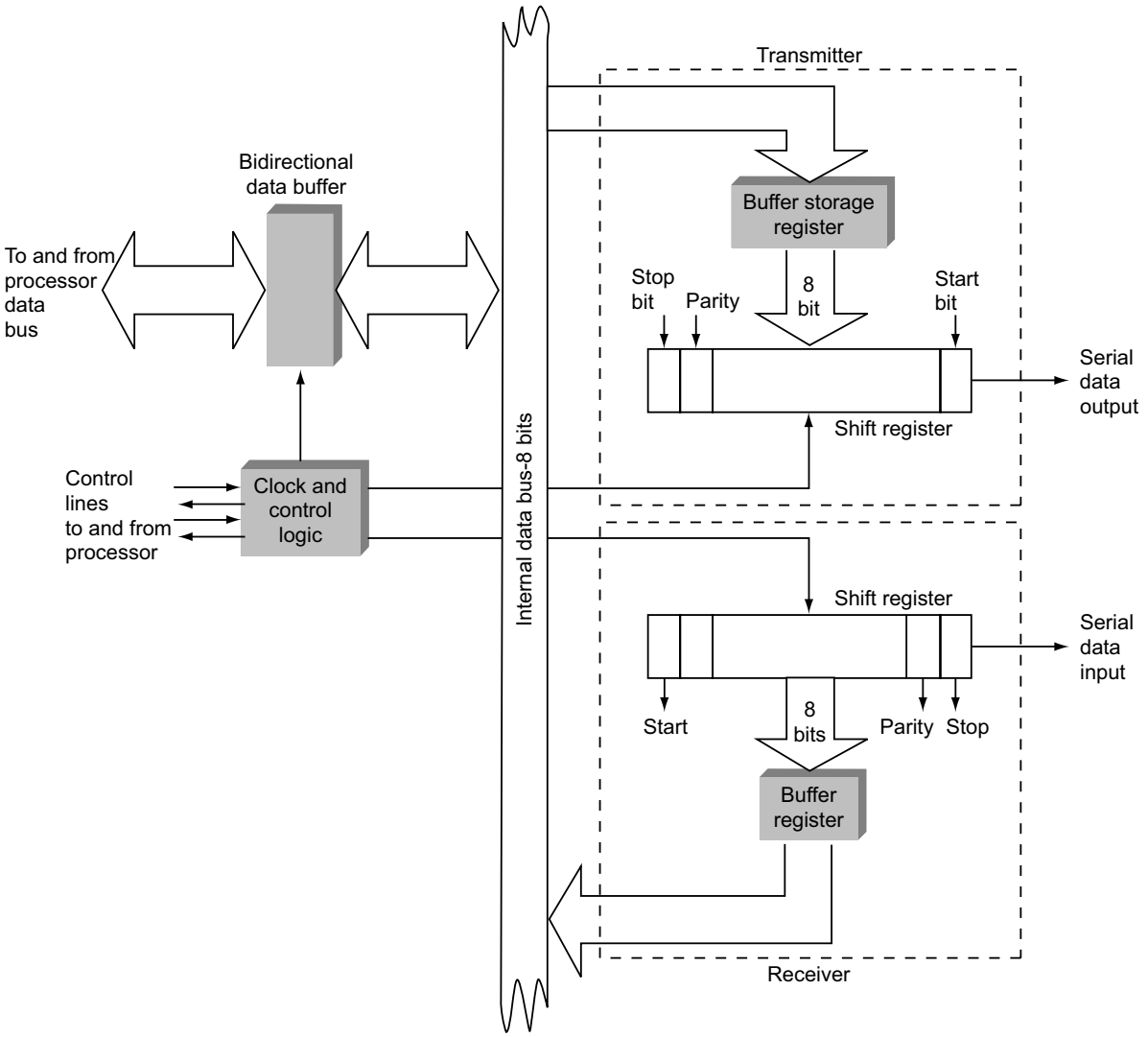
2.3.2 UART

UART (Universal Asynchronous Receiver-Transmitter) é um protocolo de comunicação serial assíncrona que utiliza apenas duas linhas de comunicação (RX e TX) e possibilita transferências de dados em modo half-duplex ou full-duplex entre dois dispositivos (ANALOG DEVICES, 2020). Adicionando bits para indicar o inicio e o fim de cada byte e operando com uma taxa de transmissão pré-configurada nos dispositivos, este padrão elimina a necessidade de uma linha dedicada para transmissão de clock e sincronicidade da comunicação (FRENZEL, 2016).

De acordo com Frenzel (2016), diversas interfaces de comunicação serial e protocolos fundamentam-se neste padrão, entre eles o RS-232 e RS-485. Por ser amplamente difundida em sistemas embarcados e microcontroladores, a UART normalmente é incorporada em outros circuitos integrados e possui um modelo padrão (Figura 20), contendo um submódulo para envio (*Transmitter*) e outro para recebimento (*Receiver*). Os dados a serem transmitidos normalmente têm origem em

um processador ou controlador embarcado. Armazenados em um buffer bidirecional, os bytes são enviados para a UART através de um barramento paralelo (*Internal data bus*).

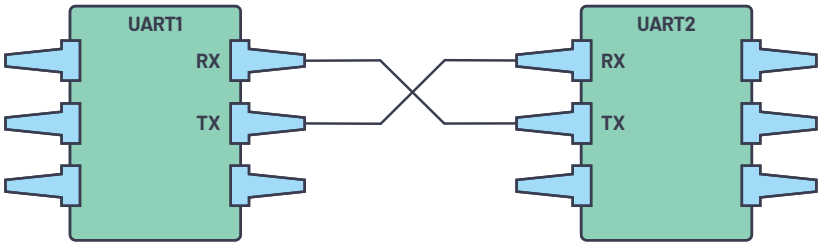
Figura 20 – Topologia padrão UART.



Fonte: Frenzel (2016).

Ao receber os dados do barramento paralelo, a UART os armazena em um registrador interno no módulo transmissor. O byte a ser enviado é então carregado em um registrador de deslocamento, onde são adicionados os bits de início, fim e, opcionalmente, o bit de paridade, podendo ser paridade par ou ímpar. Os bits são enviados serialmente ao meio físico de transmissão através do sinal de clock, configurado de acordo com a taxa de transmissão selecionada (FRENZEL, 2016). A taxa de transmissão é expressa em bps (bits per second) e comumente utilizada nos valores de 9600bps e 115200bps.

Figura 21 – Conexão física entre dois dispositivos UART.



Fonte: Analog Devices (2020).

No dispositivo destinatário, os bits chegam serialmente ao registrador de deslocamento do módulo receptor, onde são removidos os bits de start, stop e paridade. Passando por um buffer, os bytes são enviados ao barramento paralelo e armazenados no buffer bidirecional, sendo disponibilizados ao IC (Integrated Circuit) final onde serão utilizados. Para que haja comunicação bidirecional e full-duplex, é necessário que o módulo transmissor de cada dispositivo esteja conectado ao módulo receptor do outro, conforme Figura 21.

Figura 22 – Formato das mensagens na comunicação UART.

Fonte: Analog Devices (2020).

A UART admite a transmissão de dados contendo de 5 a 9 bits, sendo mais comum a configuração de 8 bits. Em todos os casos, a ordem de transmissão é a mesma, iniciando com o start bit, seguido pelos bits do byte a ser enviado (LSB primeiro) e finalizando com os bits de paridade e stop (Figura 22). As linhas de comunicação permanecem em nível lógico alto enquanto não estão transmitindo dados. O bit de inicio consiste em uma mudança no nível lógico de alto para baixo com duração de 1 ciclo de clock. Já o bit de término é indicado pelo nível lógico alto durante um ciclo de clock, caso a comunicação for sem paridade, ou dois ciclos, caso a paridade esteja configurada (ANALOG DEVICES, 2020).

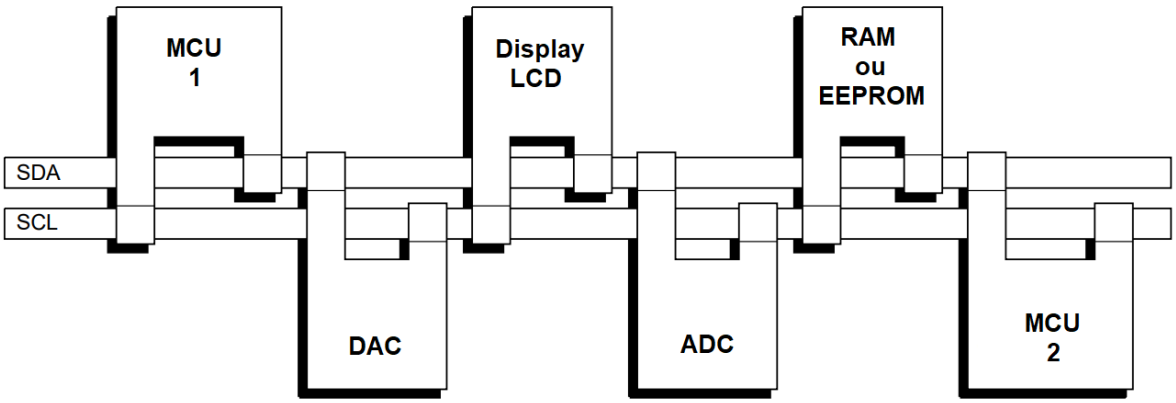
2.3.3 I2C

O I2C (Inter-Integrated Circuit) é um barramento de comunicação serial bidirecional desenvolvido pela Phillips Semiconductors (hoje NXP Semiconductors) com a finalidade de proporcionar a compatibilidade de comunicação entre circuitos integrados de diferentes fabricantes. Constituído por duas linhas de comunicação, SDA (serial data line) e SCL (serial clock line), o padrão I2C especifica a forma de endereçamento dos dispositivos conectados ao barramento, condições para detecção de início e fim da comunicação e formato dos bytes na transmissão das mensagens

(NXP, 2021).

Em um barramento I2C, dispositivos de diferentes tipos, como microcontroladores ou MCU (Microcontroller Unit), displays, memórias, conversores ADC e DAC, podem trocar informações (Figura 23). Cada um deles é identificado por um endereço único no barramento e pode atuar como controlador ou alvo da comunicação, podendo em cada um dos casos receber e enviar dados. O dispositivo controlador é responsável por gerar o clock na linha SCL, sincronizando o envio e recebimento dos bits, e também por iniciar a comunicação, endereçando o dispositivo alvo (NXP, 2021).

Figura 23 – Exemplo de barramento I2C.

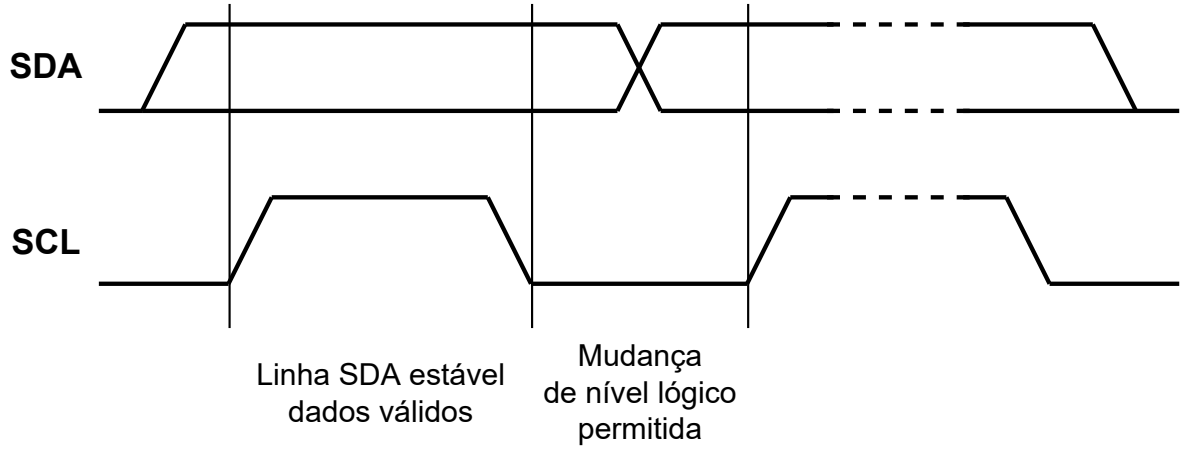


Fonte: adaptado de NXP (2021).

Tanto a linha SDA quanto a SCL são conectadas à uma fonte de tensão positiva através de resistores pull-up. Quando não há comunicação, ambas possuem nível lógico alto, indicando que o barramento está livre. Por possibilitar a conexão entre ICs fabricados com diferentes tecnologias, tais como CMOS (Complementary Metal-Oxide-Semiconductor), NMOS (Negatively Doped Metal Oxide Semiconductor) e BJT (Bipolar Junction Transistor) por exemplo, as faixas de tensão correspondentes aos níveis lógicos alto e baixo não são definidos em valores absolutos, mas sim como uma porcentagem da tensão de alimentação (VDD) dos dispositivos. Sendo $V_H \geq 0,7 \times VDD$ e $V_L \leq 0,3 \times VDD$, onde V_H e V_L são as faixas de tensão que definem os níveis lógicos alto e baixo, respectivamente (NXP, 2021).

O protocolo I2C define também taxas de transmissão padrão de 100 kbit/s, 400 kbit/s, 1 Mbit/s e 3,4 Mbit/s, sendo as duas primeiras mais comumente utilizadas. Cada bit na linha SDA é lido no período em que o clock da linha SCL está em nível lógico alto. Portanto, durante a transmissão dos bytes, a mudança no nível lógico da linha SDA deve ocorrer apenas durante o período de nível lógico baixo do clock em SCL, conforme exemplificado na Figura 24.

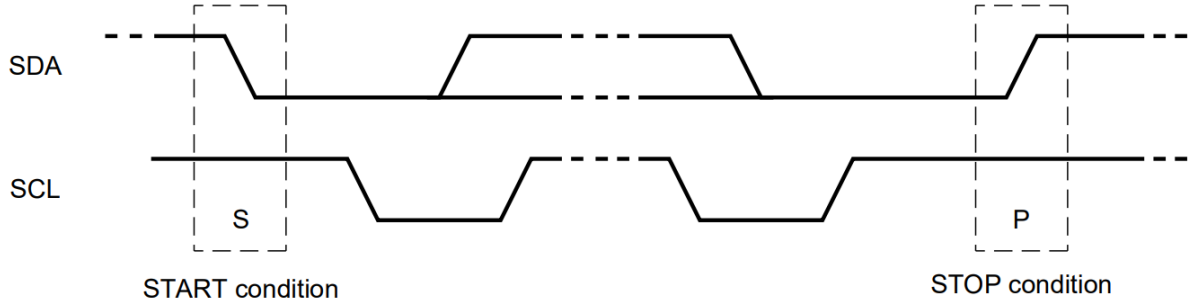
Figura 24 – Mudança do nível lógico na linha SDA.



Fonte: adaptado de NXP (2021).

Contudo, existem duas condições que permitem a transição de nível lógico em SDA enquanto SCL estiver alto. Essas condições indicam o início (START condition) e o término (STOP condition) de cada transmissão. De acordo com a Figura 25, uma transição de descida em SDA, enquanto SCL estiver alto, sinaliza o início de uma transmissão. Já o término da transmissão é indicado por uma transição de subida em SDA na mesma circunstância. O barramento é considerado ocupado após uma condição de início de transmissão, voltando a estar livre somente após a condição de término. Assim como o sinal de clock, ambas as transições são geradas pelo dispositivo controlador.

Figura 25 – Mudança do nível lógico na linha SDA.

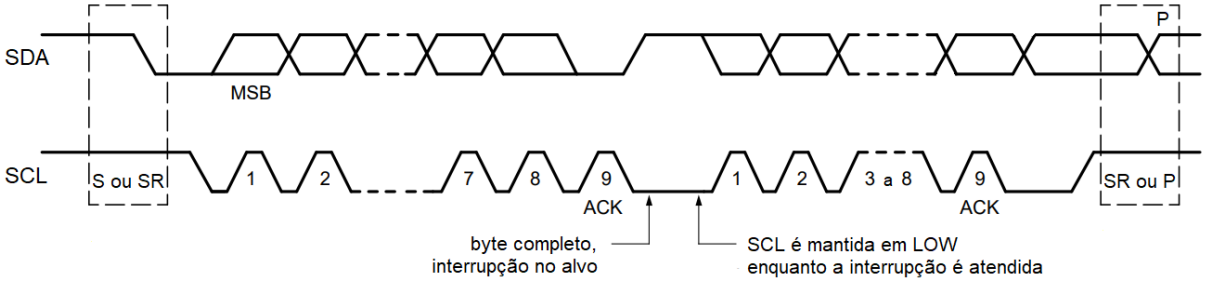


Fonte: NXP (2021).

O protocolo não define um número máximo de bytes por menssagem, sendo que a transmissão de cada um deles é iniciada pelo MSB. Após cada byte, ou seja, oito ciclos de clock, é esperado um bit de verificação de erros, denominado ACK (Acknowledge) em caso de sucesso na transmissão do byte, ou NACK (Not Acknowledge) caso contrário (Figura 26). Para indicar sucesso (ACK), o dispositivo transmissor libera a linha SDA durante o nono ciclo de clock, possibilitando que o receptor force o nível lógico baixo em SDA e o mantenha durante o período alto

em SCL. Caso o nível de SDA permaneça alto durante o nono ciclo de clock, um NACK é detectado e o dispositivo controlador pode abortar a transmissão ou reiniciá-la, enviando novamente uma condição de início, representada na Figura 26 por SR (START Repeated).

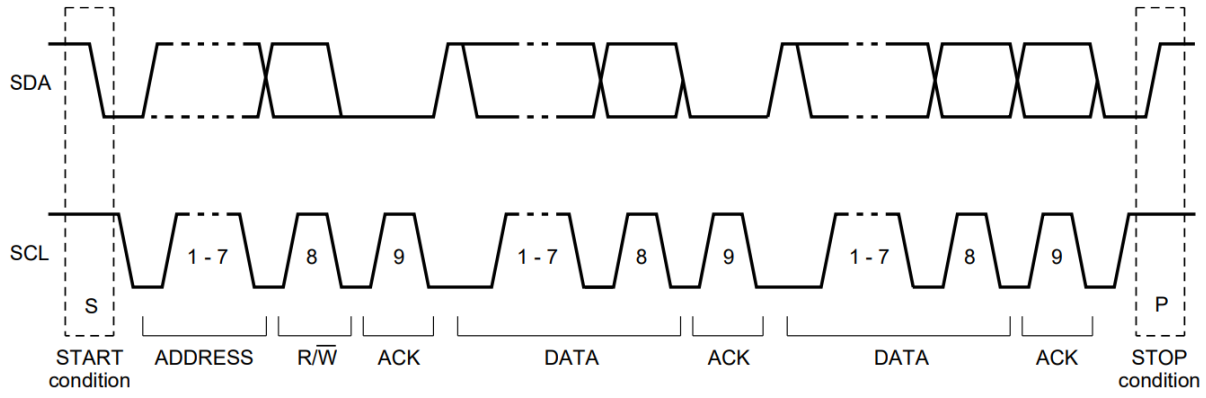
Figura 26 – Bit para verificação de erros.



Fonte: adaptado de NXP (2021).

Em alguns casos, o dispositivo alvo pode sofrer interrupções de mais alta prioridade durante a comunicação I2C, como é apresentado na Figura 26. Para não gerar falha na transmissão, o dispositivo pode forçar o nível lógico baixo na linha SCL, indicando ao controlador um estado de espera. A transferência dos dados continua assim que a linha SCL é liberada.

Figura 27 – Bit de escrita ou leitura.



Fonte: NXP (2021).

Após a condição de início, o dispositivo controlador envia um endereço de 7 bits na linha SDA, identificando o alvo da comunicação. O oitavo bit (R/W), enviado após o endereço, indica a direção da transmissão. Caso possua nível lógico baixo, indica uma operação de escrita e os bits são enviados do controlador ao alvo. Se possuir nível lógico alto, a operação é de leitura e indica que o dispositivo alvo é responsável pela inserção dos bits na linha SDA (Figura 27). Todavia, em ambos os casos a transmissão é encerrada pelo dispositivo controlador. Caso uma nova comunicação seja desejada, o controlador pode gerar novamente a condição de início, endereçando o mesmo ou um novo dispositivo alvo. Nesse caso, a condição de parada não é necessária.

2.4 MODELAGEM DE SISTEMAS

O processo de modelagem e desenvolvimento de sistemas consiste na aplicação de um conjunto de ferramentas e métodos, com a finalidade de compreender e atender as necessidades e expectativas dos usuários (VAZQUEZ; SIMÕES, 2016). Nesse contexto, a engenharia de requisitos, baseada na necessidade de um sistema em atingir determinados objetivos, estabelece as metodologias para definição, documentação e manutenção dos requisitos que serão utilizados durante as fases de projeto, desenvolvimento e testes do sistema (CROWDER; HOFF, 2022).

A engenharia de requisitos é amplamente aplicada ao desenvolvimento de software, tendo normas e recomendações específicas para esta área, como as definidas pela IEEE (Institute of Electrical and Electronics Engineers) nas recomendações da norma IEEE830 (IEEE, 1998). Os mesmos conceitos também podem ser aplicados em um âmbito geral do desenvolvimento de sistemas, assim como demonstram as padronizações da ISO (International Organization for Standardization) na ISO/IEC/IEEE29148 e da IEEE, na norma IEEE1233 (ISO, 2018) (IEEE, 1996). Segundo Crowder e Hoff (2022), a especificação de requisitos é um esforço multidisciplinar, devido à ampla natureza dos requisitos necessários para descrever e delimitar todas as funcionalidades exigidas de um sistema que envolva diversas áreas do conhecimento.

De acordo com a IEEE (1998), uma especificação de requisitos é considerada adequada se produzir um conjunto inequívoco e completo de funcionalidades e obrigações a serem implementadas e entregues pelo sistema. É recomendado que o levantamento de tais regras e funcionalidades seja realizado em conjunto com os clientes e usuários finais do produto, garantindo o alinhamento entre a necessidade dos usuários e o que será entregue ao fim do projeto.

O processo de criação de uma especificação de requisitos de software é definido pela norma IEEE830, que descreve e elenca os principais benefícios obtidos através de uma especificação adequada (IEEE, 1998). Considerando o contexto multidisciplinar deste projeto, os requisitos gerais serão abordados a nível de todo o sistema e aprofundados por cada subárea (mecânica, hardware e software). Os benefícios para este caso são descritos pela norma ISO (2018):

- *Estabelecimento de uma base de acordo entre clientes e fornecedores sobre o que o sistema deverá fazer.* A descrição completa das funcionalidades possibilita aos usuários determinarem se o sistema atende suas necessidades ou como ele deve ser modificado para que atenda;
- *Redução do esforço de desenvolvimento.* Os requisitos obrigam o projetista a considerar rigorosamente as especificações de cada funcionalidade ao desenvolver a solução, reduzindo o tempo gasto com reprojetos e retestes;

- *Fornece uma base para estimar os custos e cronogramas.* A definição do produto a ser desenvolvido é uma base realista para estimar os custos com materiais e o tempo necessário para implementar cada funcionalidade;
- *Fornece métricas para verificação e validação.* Os requisitos específicos fornecem uma linha de base para verificar a conformidade do produto final;
- *Facilita a interoperabilidade, modularidade e transferência.* Por serem definições completas e inequívocas, o mesmo resultado poderá ser obtido por diferentes desenvolvedores, facilitando a subdivisão de tarefas, transferência ou continuação do projeto;
- *Possibilita melhorias.* Por ser uma definição do que deve ser feito e não de como cada funcionalidade deve ser implementada, a especificação de requisitos possibilita futuras mudanças nas soluções inicialmente propostas.

Para que uma especificação de requisitos seja adequada, o processo descrito pela norma IEEE 830 define que ela deve ser: correta, inequívoca, completa, consistente, classificada por importância, verificável, modificável e rastreável. A estrutura da especificação de requisitos da bancada experimental de emulsões de água e óleo será baseada nos critérios e procedimentos definidos pelas normas supracitadas.

3 ESPECIFICAÇÃO DE REQUISITOS

Este trabalho se insere no contexto das pesquisas realizadas pelo laboratório T2F no estudo de fenômenos de transporte encontrados nas plataformas de extração de petróleo e gás natural. Mais especificamente, este projeto faz parte de uma linha de pesquisa voltada à instrumentação e caracterização de escoamentos multifásicos, tendo seu escopo focado no desenvolvimento de uma bancada de testes que possibilite o avanço das pesquisas nessa área.

A especificação de software e hardware do sistema foi definida com base nas funcionalidades exigidas da bancada de emulsão, as quais foram obtidas através do levantamento de requisitos junto aos pesquisadores (professores, mestrandos e doutorandos) do laboratório T2F. Restrições em relação ao tamanho, custo e disponibilidade de equipamentos também foram levadas em consideração. Os requisitos apresentados nesta seção serão classificados em funcionais *[RFxx]* e não funcionais *[RNFxx]*, distribuídos entre requisitos gerais do sistema, requisitos mecânicos e requisitos de hardware e software.

3.1 REQUISITOS DO SISTEMA

De uma forma geral, a bancada de testes deve emular os ecoamentos multifásicos entre água e óleo crude encontrados em uma plataforma de extração de petróleo, possibilitando a geração de diferentes padrões de escoamento e níveis de emulsão. Uma interface para o controle e aquisição de dados dos experimentos deverá ser disponibilizada ao usuário, abstraindo as questões operacionais do sistema.

Os diferentes padrões de escoamento serão formados de acordo com a fração e velocidade de cada fase, as quais serão determinadas pela vazão nos ramais de água e de óleo. A emulsificação, definida pelas características dos fluidos, como densidade, viscosidade e tensão superficial da interface, além do tamanho e volume das gotículas da fase dispersa, será influenciada principalmente pela velocidade e turbulência do escoamento multifásico, que também deverá ser regulada.

A reprodutibilidade deve ser assegurada, garantindo que o mesmo padrão de escoamento e nível de emulsão possam ser gerados em diferentes testes, possibilitando a comparação entre os resultados obtidos através das técnicas de medição multifásica a serem estudadas. Uma seção de testes translúcida deve ser disponibilizada após a emulsão, permitindo a instalação dos dispositivos a serem testados e a inspeção visual do escoamento. Variáveis como pressão, vazão e temperatura deverão ser mensuradas e armazenadas, de forma que seja possível acessá-las posteriormente para análise e correlação dos dados.

Portanto, os requisitos funcionais do sistema são:

- **[RF01]** Gerar vazões nos ramais de alimentação de água e óleo chegando ao menos em $0{,}5\,\mathrm{kg/s}$;
- **[RF02]** Medir a vazão mássica de cada um dos ramais antes da emulsão em uma faixa de 0 a $1\,\mathrm{kg/s}$ com precisão mínima de $\pm 2\,\%$;
- **[RF03]** Medir a pressão absoluta de cada um dos ramais antes da emulsão em uma faixa de 0 a $200\,\mathrm{kPa}$ com precisão mínima de $\pm 5\,\%$;
- **[RF04]** Medir a temperatura dos fluidos em cada um dos ramais antes da emulsão em uma faixa de 0 a $100\,^{\circ}\mathrm{C}$ com precisão mínima de $\pm 2\,\%$;
- **[RF05]** Disponibilizar uma válvula, após a união dos fluxos de água e óleo, que possibilite regular remotamente as turbulências no escoamento;
- **[RF06]** Disponibilizar uma interface para o usuário controlar todos os atuadores e realizar a aquisição dos dados através de um computador;
- **[RF07]** Armazenar todas as leituras, valores convertidos e parâmetros dos atuadores para acessar após os testes;
- **[RNF01]** Disponibilizar, após a emulsão, uma seção de testes com $500\,\mathrm{mm}$ de comprimento feita em tubulação de acrílico com diâmetro de $50\,\mathrm{mm}$.

Os requisitos gerais do sistema serão complementados pelos requisitos de cada subárea do projeto (mecânica, hardware e software). Requisitos mecânicos irão abranger aspectos construtivos das estruturas e tubulações utilizadas. Os sistemas elétrico e eletrônico serão especificados pelos requisitos de hardware. Por fim, os requisitos de software definirão o comportamento esperado do sistema e sua interação com o usuário.

3.1.1 Requisitos mecânicos

A estrutura da bancada experimental terá a função de suportar a instalação dos equipamentos e componentes de hardware necessários para o cumprimento dos requisitos gerais do sistema. A estabilidade, robustez e segurança da estrutura serão cruciais para o correto funcionamento dos testes. Limitações em relação à mobilidade da bancada, espaço ocupado e disponibilidade de materiais também deverão ser atendidas.

Os fluidos utilizados (água e óleo crude) devem ser devidamente armazenados, garantindo a vedação. Deve haver disponibilidade de um volume suficientemente grande de ambos os líquidos para manter a bancada em funcionamento por um tempo mínimo de estabilização dos escoamentos e realização dos testes. O escoamento emulsificado também deve ser armazenado, possibilitando a separação das fases por decantação.

As tubulações utilizadas devem ser resistentes à corrosão causada pela reação entre o material e os fluidos e, no ramal de óleo, deve suportar a passagem de soluções

ácidas para fins de limpeza. Sendo assim, os requisitos mecânicos a serem cumpridos são definidos como:

- **[RF08]** Disponibilizar armazenamento mínimo de $150\,\mathrm{L}$ de cada fluido (água e óleo crude);
- **[RF09]** Disponibilizar armazenamento mínimo de $300\,\mathrm{L}$ da emulsão água-óleo (para decantação);
- **[RNF02]** Utilizar tubulação de PVC com $25{,}4\,\mathrm{mm}$ de diâmetro no ramal de água;
- **[RNF03]** Utilizar tubulação de aço inox com $25{,}4\,\mathrm{mm}$ de diâmetro nos ramais que contém óleo;

3.1.2 Requisitos de hardware

O objetivo principal do hardware é dar suporte às funcionalidades que serão implementadas via software, fornecendo as medições requeridas e formas de atuar nas principais variáveis do processo. O hardware também deve garantir a correta conexão entre os dispositivos do sistema, possibilitando a transferência de potência entre as fontes e os equipamentos, assim como a transmissão de dados e comunicação.

Especificar os atuadores que serão utilizados será o ponto inicial para a modelagem do hardware, tendo em vista que o restante dos componentes deverão se adequar aos requisitos de operação desses equipamentos. De acordo com o requisito funcional RF07, o controle dos atuadores deve ser realizado através de uma interface gráfica, a qual será executada em um computador de controle. Sendo assim, os atuadores, responsáveis por gerar as vazões de água, óleo e regular a turbulência na emulsão, devem ser capazes de receber comandos enviados pelo computador. O hardware deverá fornecer os barramentos de alimentação, comunicação e conversões necessárias para esta funcionalidade.

Com isso, foram definidos os seguintes requisitos de hardware:

- **[RF10]** Utilizar um computador com sistema operacional Windows, versão 7 ou superior, possuindo ao menos 4GB de RAM para realizar a interface com o usuário;
- **[RNF04]** Realizar as conversões necessárias para que todas as comunicações entre os equipamentos da bancada e o computador sejam realizadas através das portas USB;

3.1.3 Requisitos de software

A principal função do software é implementar as funcionalidades requeridas e abstrair as nuances operacionais do sistema, como leitura dos sinais de medição, comunicação entre dispositivos e controle dos atuadores. O software deve fornecer ao usuário uma interface gráfica que simplifique a utilização da bancada e disponibilize

as funções necessárias para geração, controle e aquisição de dados de diferentes padrões de escoamento.

Cálculos para conversão das leituras dos sinais de medição e controle dos atuadores também deverão ser realizados. Os sinais de medição, valores convertidos e valores de referência utilizados para comandar os atuadores, devem ser armazenados em um arquivo de texto. O usuário deve poder escolher o nome e diretório do arquivo a ser gerado.

Os valores convertidos deverão ser apresentados graficamente ao usuário. Os gráficos devem ser subdivididos em abas, de acordo com o tipo de medição (vazão, pressão ou temperatura). Os eixos dos gráficos devem estar fixados de acordo com a faixa de medição, deixando explicito ao usuário a unidade de medida das leituras.

O estado de funcionamento dos atuadores deve ser exibido através de indicadores visuais, sendo o estado ligado sinalizado por um ícone aceso e o estado desligado por um ícone apagado na interface gráfica. Por fim, a interface deve fornecer botões para iniciar e parar o experimento, armazenando os dados e desligando os atuadores.

Requisitos funcionais do software:

- **[RF11]** Realizar os cálculos de conversão dos sinais de medição para unidades de engenharia;
- **[RF12]** Realizar a conversão dos comandos do usuário para valores de referência dos atuadores;
- **[RF14]** Exibir graficamente os valores de medição em unidades de engenharia, separados por abas de acordo com o tipo de medição (vazão, pressão ou temperatura);
- **[RF13]** Armazenar os dados de leitura e de comandos em um arquivo de texto;
- **[RNF05]** Disponibilizar campo para que o usuário defina o nome e diretório do arquivo de texto para armazenamento dos dados;
- **[RNF06]** Indicar o estado dos atuadores através de ícones visuais na tela principal da interface gráfica (aceso indicando ligado, e apagado indicando desligado ou inativo);
- **[RNF07]** Disponibilizar um botão na tela principal da interface para finalizar o experimento, desligar os atuadores e salvar os dados.

4 METODOLOGIA

Considerando o conjunto de requisitos descritos acima, neste capítulo será apresentada a síntese da solução e os métodos utilizados para a elaboração dos sistemas mecânico, de hardware e software da bancada experimental de emulsões água-óleo.

Inicialmente, serão apresentados os atuadores e instrumentos de medição utilizados, seguidos pelo sistema mecânico, composto pelas tubulações, tanques de armazenamento e estruturas de fixação. Com isso, os equipamentos auxiliares e hardware necessários para o envio de comandos, comunicação e aquisição de dados podem ser definidos. Por fim, o firmware e software serão modelados de acordo com as funcionalidades exigidas e condições demandadas pelo hardware.

4.1 ATUADORES

De acordo com os requisitos *[RF01]*, *[RF05]* e *[RF06]*, será necessário gerar vazões nos ramais de alimentação e regular a turbulência do escoamento, possibilitando controlar remotamente os atuadores através de um computador. Para isso, os materiais e métodos utilizados serão descritos a seguir.

4.1.1 Motobombas

A formação dos escoamentos bifásicos e emulsões entre água e óleo crude irá ocorrer por meio da união de fluxos monofásicos de água e óleo. Para gerar vazões de até $0{,}5\,\mathrm{kg/s}$ nos ramais de entrada, conforme requisito *[RF01]*, serão utilizadas duas motobombas modelo *Schneider* BC-92S, apresentada na Figura 28.

Figura 28 – Motobomba *Schneider* BC-92S.

Fonte: Franklin Electric (2022).

De acordo com o fabricante, tais motobombas são formadas por um motor de indução trifásico $380\,\mathrm{V}$, de 2 polos e $60\,\mathrm{Hz}$, e um rotor de bronze com selo mecânico Viton. Com uma potência de $1{,}1\,\mathrm{kW}$ ($1{,}5\,cv$), elas fornecem no pior caso uma vazão de $6{,}1\,\mathrm{m^3/h}$ à uma pressão de $24\,M.C.A$ (Metros de Coluna d'Água), que corresponde à aproximadamente $235\,\mathrm{kPa}$ (ver Figura 29).

Figura 29 – Características hidráulicas das motobombas BC-92S.

Modelo	Potência (cv)	Características Hidráulicas																						
		Altura Manométrica Total (m.c.a.)																						
		2	4	6	8	10	12	14	16	18	20	22	24	26	28	30	32	34	36	38	40	42	44	46
		Vazão em m³/h válida para sucção de 0 m.c.a.																						
BC-92 S/T 1B	1	*	*	*	19,7	17,8	15,7	13,4	10,9	7,9														
	1/5	*	*	*	*	21,9	20,2	18,4	16,4	14,3	12	9,3	6,1											
	2 (127mm)	*	*	*	*	*	*	22	20,4	18,6	16,8	14,7	12,5	10	7,1									
	2 (137mm)	*	*	*	*	*	*	*	*	*	*	*	*	14,8	12,7	10,4	7,8	4,7						
	3 (143mm)	*	*	*	*	*	*	*	*	23,2	21,8	20,4	18,9	17,2	15,5	13,6	11,6	9,2	6,6					
	3 (155mm)	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	*	13,7	11,9	9,9	7,6	5	

Fonte: adaptado de Franklin Electric (2022).

De acordo com o requisito *[RF03]*, a pressão em ambos os ramais deve ser lida em uma faixa de 0 a $200\,\mathrm{kPa}$. Portanto, o sistema deverá operar com pressões sempre abaixo de $200\,\mathrm{kPa}$. Ao operar na pressão máxima permitida, a vazão atingirá pelo menos $6{,}1\,\mathrm{m^3/h}$, pois $200\,\mathrm{kPa} < 235\,\mathrm{kPa}$. Sendo assim, a vazão mássica que pode ser obtida no ramal de água é, analiticamente:

$$\begin{aligned} Q_m &= \rho * Q_v \\ &= 997\,\mathrm{kg/m^3} * 6{,}1\,\mathrm{m^3/h} \\ &= 6.081{,}7\,\mathrm{kg/h} \\ &= 1{,}69\,\mathrm{kg/s} \end{aligned} \tag{18}$$

Onde Q_v é a vazão volumétrica e ρ é a densidade da água a $25\,^{\circ}\mathrm{C}$. Com isso, esse modelo de motobomba atende o requisito de gerar vazões de até $0{,}5\,\mathrm{kg/s}$ no ramal de água.

Os dados disponibilizados pelo fabricante não correlacionam as vazões com a densidade e viscosidade do fluido. Portanto, devido a diferença de tais características entre os dados do fabricante, obtidos com água, e o óleo crude, os dados referentes à vazão mássica atingida no ramal de óleo serão obtidos experimentalmente.

4.1.2 Inversores de frequência

Para o acionamento das motobombas serão utilizados dois inversores de frequência CFW300, fabricados pela WEG (Figura 30). Esses inversores fornecem

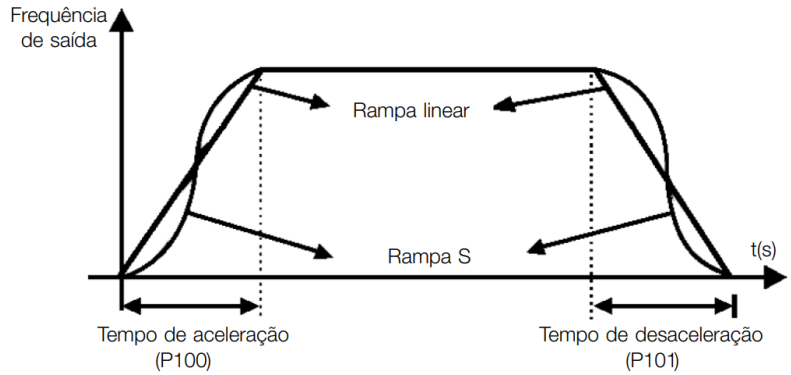
alimentação trifásica de $380\,\mathrm{V}$ com corrente nominal de até $15{,}2\,\mathrm{A}$. Disponibilizam diversos parâmetros para controle de motores de indução, como rampa linear ou rampa S para aceleração e desaceleração, modos de controle V/f e vetorial, além de um módulo de expansão para comunicação serial, que utiliza o padrão RS485 e protocolo Modbus.

Figura 30 – Inversor de frequência CFW300.

Fonte: WEG (2021).

A fim de suavizar os choques mecânicos, rampas em formato S foram utilizadas neste trabalho para realizar o acionamento e desligamento das motobombas, com duração de $5\,\mathrm{s}$ e $10\,\mathrm{s}$, respectivamente. Conforme a Figura 31, a configuração das rampas é feita através dos parâmetros $P100$ e $P101$ e a seleção da rampa linear ou em S é dada por $P104$.

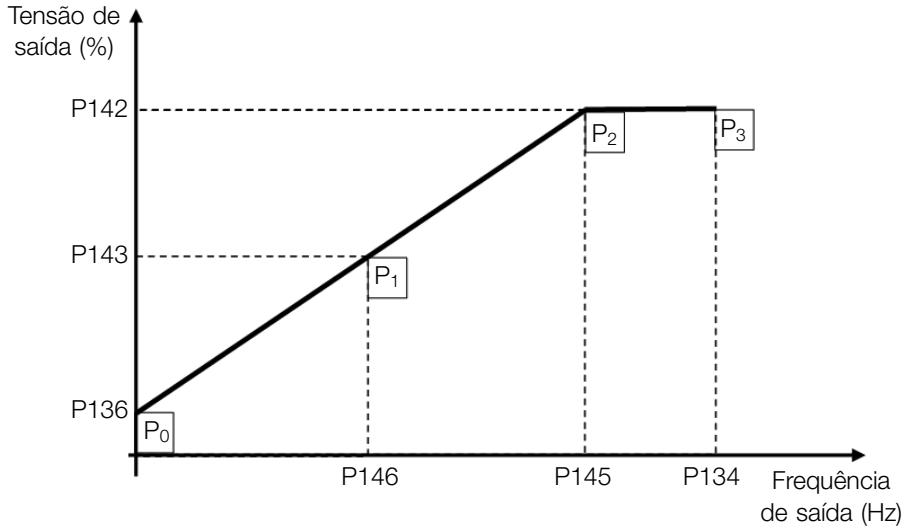
Figura 31 – Rampas de aceleração e desaceleração.



Fonte: adaptado de WEG (2021).

Para o controle da velocidade foi configurado o modo V/f, tipicamente utilizado em motores de indução trifásicos. Nesse modo, o inversor atua como uma fonte de tensão, ajustando a frequência para evitar a saturação do fluxo magnético no motor. A relação entre tensão e frequência é dada por um coeficiente linear, de acordo com a curva V/f, que pode ser ajustada através dos parâmetros da Figura 32 para atender as características da aplicação.

Figura 32 – Parâmetros da curva V/f.



Fonte: WEG (2021).

O ponto P_0 indica a amplitude da tensão aplicada em $0\,\mathrm{Hz}$, P_2 define a amplitude e frequência nominais e a região de enfraquecimento do campo é determinada entre P_2 e P_3. O ajuste da curva para uma relação não-linear entre torque e frequência pode ser realizado no ponto P_1.

Os inversores CFW300 possuem uma curva V/f pré-ajustada para motores de indução trifásicos de $60\,\mathrm{Hz}$. Essa configuração pode ser carregada através do parâmetro $P204 = 5$, e define $P_0 = [0\,\mathrm{Hz}, 5\,\%]$, $P_1 = [30\,\mathrm{Hz}, 50\,\%]$ e $P_2 = [60\,\mathrm{Hz}, 100\,\%]$ e $P_3 = [66\,\mathrm{Hz}, 100\,\%]$. O último ponto foi ajustado para $P_3 = [60\,\mathrm{Hz}, 100\,\%]$, evitando que o motor opere acima das condições nominais. A tensão correspondente à amplitude de $100\,\%$ é dada por $P296$ e foi ajustada para $380\,\mathrm{V}$.

A velocidade do motor e, consequentemente, a vazão entregue pela motobomba, são definidas pela referência de frequência, que será ajustada através da interface de comunicação serial via Modbus. Internamente a CPU do inversor utiliza variáveis de 16 bits para tratamento das referências de frequência. Contudo, para as referências digitais, dadas pelas interfaces de comunicação, uma escala de 13 bits é utilizada (WEG, 2021). Sendo assim, o valor de 8192 (2^{13}) equivale à frequência nominal do motor, configurada no parâmetro $P403$. Com isso, para definir a frequência de referência (R_{Hz}) será necessário ajustar o valor para a escala de 13 bits, convertendo a faixa de 0 a $60\,\mathrm{Hz}$ para 0 a 8192:

$$R_{13bits} = \frac{R_{Hz} \times 8192}{60} \tag{19}$$

Para configurar os inversores com as funcionalidades apresentadas acima, os parâmetros foram configurados de acordo com a Tabela 2.

Tabela 2 – Parâmetros para configuração dos inversores de frequência.

Parâmetro	Valor	Unidade	Função
P204	5	cfg	Carrega padrão de fábrica
P100	5	s	Tempo de aceleração
P101	10	s	Tempo de desaceleração
P104	1	cfg	Utilizar rampa S
P105	0	cfg	Utilizar a 1ª rampa (P100 e P101)
P121	0,0 a 400,0	Hz	Referência de velocidade LOC pela IHM
P134	60	Hz	Define P3 na curva V/f
P136	5	%	Define P0 na curva V/f
P142	100	%	Define P2 e P3 na curva V/f
P143	50	%	Define P1 na curva V/f
P145	60	Hz	Define P2 na curva V/f
P146	30	Hz	Define P1 na curva V/f
P202	0	cfg	Modo de controle V/f
P220	0/1	cfg	Referencia de velocidade LOC/REM
P221	0	cfg	Referência LOC via IHM
P222	9	cfg	Referência REM via comunicação serial
P223	0	cfg	Sentido de giro LOC Horário
P224	0	cfg	Gira/Para LOC via IHM
P225	0	cfg	JOG LOC Inativo
P226	0	cfg	Sentido de giro REM Horário
P227	2	cfg	Gira/Para REM via comunicação serial
P228	0	cfg	JOG REM Inativo
P229	0	cfg	Modo de parada por rampa
P296	4	V	Tensão nominal da Rede de alimentação 380V
P308	1 ou 2	cfg	Endereço serial
P310	4	cfg	Taxa da comunicação serial 76800 bits/s
P311	1	cfg	8 bits, paridade par, 1 stop bit
P312	2	cfg	Protocolo serial Modbus RTU
P313	0	cfg	Ação para erro de comunicação inativa
P314	0	cfg	Watchdog serial inativo
P403	60	Hz	Frequência nominal do motor

Fonte: autor (2022).

Os inversores serão alimentados com $380\,\mathrm{V}$, posicionados em um painel que conta com um disjuntor trifásico na entrada, protegendo-os contra curtos e sobrecarga.

4.1.3 Válvula Eletropneumática

Para atender o requisito *[RF05]* uma válvula modelo ARI-STEVI Pro 470 ANSI em conjunto com um posicionador eletropneumático NT-1000L será utilizada logo após a união dos escoamentos de água e óleo. A válvula é acoplada a um atuador pneumático com ação reversível por molas, que deve ser alimentado com ar comprimido a uma pressão de até $6\,\mathrm{bar}$. Ambos são apresentados na Figura 33.

Figura 33 – Detalhes do atuador pneumático e válvula.

(a) Vista detalhada do atuador pneumático. (b) Vista detalhada da válvula.

Fonte: adaptado de Bermo (2019).

O posicionador NT-1000L atua linearmente, controlando o fluxo de ar no atuador pneumático e limitando a ação das molas de retorno, possibilitando o controle do nível de abertura da válvula. Sua interface para comunicação é através de uma malha de corrente de 0 a $20\,\mathrm{mA}$, correspondendo a 0 e $100\,\%$ de abertura, respectivamente.

Figura 34 – Conjunto completo da válvula eletropneumática.

Fonte: JY Válvulas (2020).

A Figura 34 mostra o conjunto contendo a válvula, atuador pneumático e posicionador.

4.2 INSTRUMENTOS DE MEDIÇÃO

A medição das principais variáveis do processo será necessária para controlar e caracterizar os experimentos. Segundo *[RF02]*, *[RF03]* e *[RF04]* será necessário

medir a vazão mássica, pressão absoluta e temperatura nos ramais de água e óleo antes da união dos escoamentos. Os equipamentos utilizados para tais funções serão apresentados a seguir.

4.2.1 Medição de vazão

Devido a diferença das propriedades dos fluidos e necessidade de limpeza e manutenção dos equipamentos, a medição das vazões será realizada de maneira distinta nos ramais de água e óleo.

4.2.1.1 Ramal de água

No ramal de água será utilizado um medidor de vazão Coriolis modelo CMF200M em conjunto com um transmissor 2700, ambos da linha Micro Motion, fabricados pela Emerson Electric (Figura 35). Segundo Emerson (2021), esse medidor opera com vazão nominal de $47\,900\,\mathrm{L/h}$.

Considerando a densidade da água como $0{,}997\,\mathrm{kg/L}$ a 25 ℃, a vazão nominal do medidor Coriolis será de $13{,}27\,\mathrm{kg/s}$:

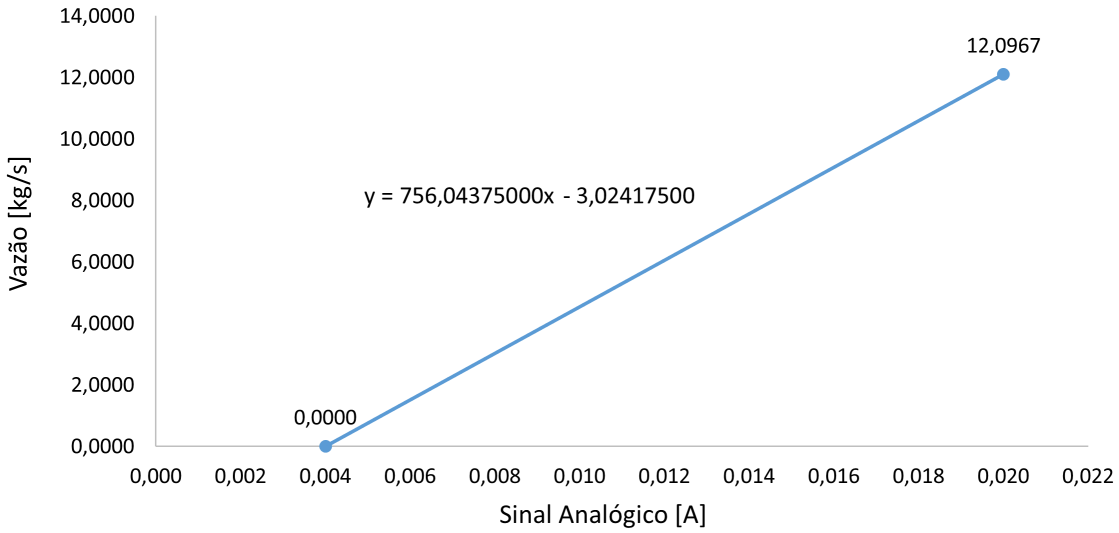
$$Q_{m\ Coriolis}^{Nom.} = 47\,900\left[\frac{\mathrm{L}}{\mathrm{h}}\right] \times 0{,}997\left[\frac{\mathrm{kg}}{\mathrm{L}}\right] \times \frac{1}{3600}\left[\frac{\mathrm{h}}{\mathrm{s}}\right] = 13{,}27\mathrm{kg/s} \tag{20}$$

Figura 35 – Conjunto do medidor de vazão coriolis e transmissor.

Fonte: Emerson (2021).

O sinal de saída do transmissor é dado no padrão de 4 a $20\,\mathrm{mA}$, com uma precisão informada pelo fabricante de $\pm 0.1\,\%$ da vazão medida. A conversão do sinal analógico para unidades de engenharia é dada por uma relação linear, onde $4\,\mathrm{mA}$ representam $0\,\mathrm{kg/s}$ e $20\,\mathrm{mA}$ indicam $12{,}0967\,\mathrm{kg/s}$, segundo calibração do fabricante. Com isso, é possível gerar a curva de calibração e o polinômio para conversão das leituras, conforme apresentado na Figura 36 e Equação 21:

Figura 36 – Curva de calibração do medidor de vazão coriolis CMF200M.



Fonte: autor (2022).

$$Q_{m\ Coriolis} = 756{,}04375 \times i_{Coriolis} - 3{,}024175 \tag{21}$$

Sendo assim, esse medidor atende o requisito *[RF02]* no ramal de água, medindo a faixa de 0 a $0{,}5\,\mathrm{kg/s}$ com uma precisão de $\pm 0.1\,\%$.

4.2.1.2 Ramal de óleo

No ramal de óleo um medidor de vazão eletromagnético da linha *Rosemount* 8700M será utilizado, evitando a perda de carga causada pela geometria de um medidor Coriolis e facilitando a limpeza do óleo no interior do equipamento. Apresentado na Figura 37, esse medidor conta com um sensor de $25{,}4\,\mathrm{mm}$ modelo 8711 e um transmissor $8732E$.

Figura 37 – Conjunto do medidor de vazão eletromagnético e transmissor.

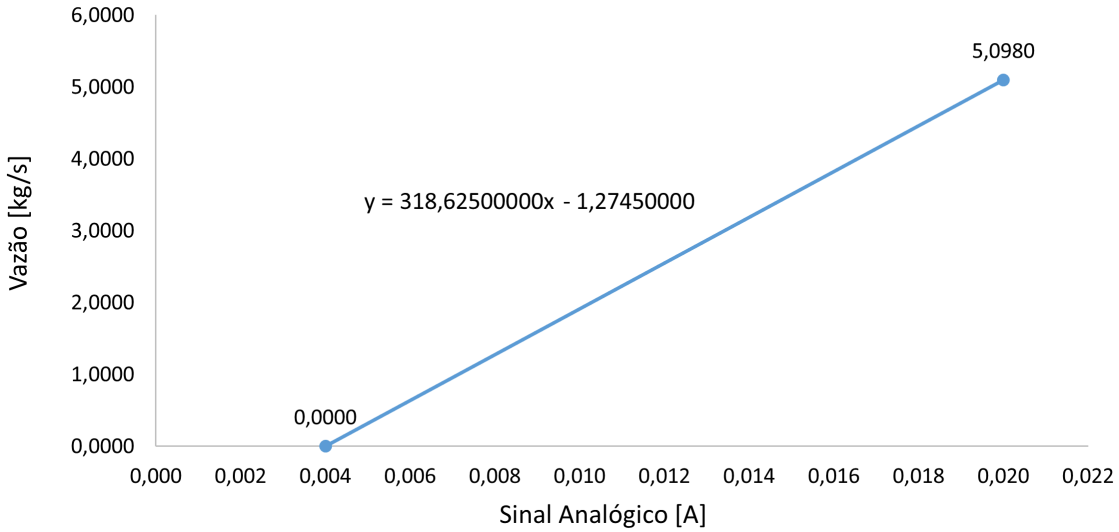
(a) Medidor de vazão modelo 8711. (b) Transmissor $8732E$ do medidor de vazão.

Fonte: adaptado de Emerson (2019).

Segundo Emerson (2019), essa linha de medidores é capaz de medir escoamentos com velocidades de 0 a $12\,\mathrm{m/s}$ com precisão de $\pm 0{,}25\,\%$. Conhecendo o diâmetro interno pelo qual o fluido escoa no sensor, o transmissor realiza a conversão da velocidade do escoamento para vazão mássica, medindo a temperatura e considerando a densidade da água para o cálculo.

Pela calibração de fábrica, a saída do transmissor é um sinal de 4 a $20\,\mathrm{mA}$, que representa a faixa de 0 a $5{,}098\,\mathrm{kg/s}$ da vazão do escoamento. Com isso, é possível obter a curva de calibração apresentada na Figura 38 e Equação 22.

Figura 38 – Curva de calibração do medidor de vazão eletromagnético *Rosemount* 8700M.



Fonte: autor (2022).

$$Q^{agua}_{m\ Eletromag} = 318{,}625 \times i_{Eletromag} - 1{,}2745 \tag{22}$$

A medição obtida com a conversão apresentada acima é referente à um escoamento de água. Para obter a vazão mássica de outro fluido, basta multiplicar o valor por sua densidade relativa ρ_r. Segundo Anton Paar (2021), o óleo crude possui densidade relativa entre de $0{,}87$ e $0{,}92$, sendo assim, sua vazão mássica pode ser determinada por:

$$Q^{oleo}_{m\ Eletromag} = \rho_r \times Q^{agua}_{m\ Eletromag} \tag{23}$$

No pior caso, a maior vazão mássica possível de ser medida no ramal de óleo será de $0{,}87 \times 5{,}098 = 4{,}435\,\mathrm{kg/s}$. Portando, a utilização desse medidor no ramal de óleo atende ao requisito *[RF02]*, medindo de 0 a $1\,\mathrm{kg/s}$ com precisão de $\pm 0{,}25\,\%$.

4.2.2 Medição de pressão

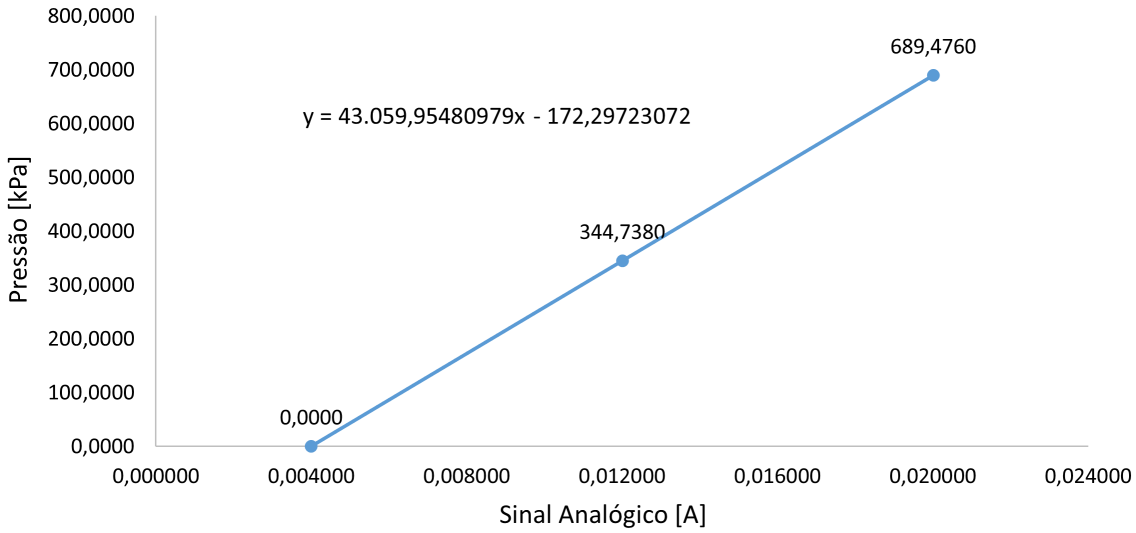
Para medir as pressões nos ramais de água e óleo, serão utilizados dois transdutores de pressão piezorresistivos modelo PX409-100GI (Figura 39), com fundo de escala de $100\,\mathrm{psi}$ ($689{,}476\,\mathrm{kPa}$). De acordo com Omega (2015), a precisão desse modelo é de $\pm 0{,}08\,\%$. Conforme requisito *[RF03]*, tal transdutor atende a faixa de medição de até $200\,\mathrm{kPa}$, superando a precisão desejada de $\pm 5\,\%$.

Figura 39 – Transdutor de pressão modelo PX409-100GI.

Fonte: adaptado de Omega (2015).

Esse transdutor deve ser alimentado com $24\,\mathrm{V}$ e apresenta um sinal de saída em corrente, de 4 a $20\,\mathrm{mA}$ (OMEGA, 2015). O fabricante informa os pontos de calibração como sendo: $[0\,\mathrm{psi}, 4{,}001\mathrm{mA}]$, $[50\,\mathrm{psi}, 12{,}008\mathrm{mA}]$ e $[100\,\mathrm{psi}, 20{,}013\mathrm{mA}]$. Convertendo as pressões para kPa, obtém-se a curva de calibração apresentada na Figura 40 e Equação 24.

Figura 40 – Curva de calibração do transdutor de pressão PX409-100GI.



Fonte: autor (2022).

$$P_{PX409} = 43.059{,}95480979 \times i_{PX409} - 172{,}29723072 \tag{24}$$

4.2.3 Medição de temperatura

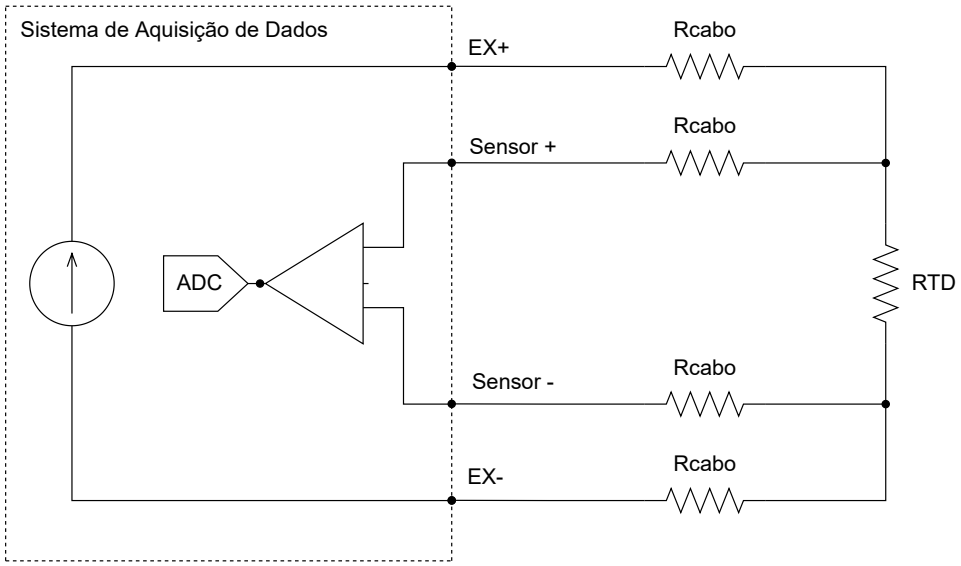
A temperatura em ambos os ramais de alimentação será medida com RTDs modelo $PT100$, fabricados com platina e apresentando resistência de $100\,\Omega$ à temperatura de 0 ℃. De acordo com Omega (2013), os limites de medição vão de -29 ℃ a 100 ℃, sendo listado como Classe A, a qual define uma precisão mínima de $\pm 0{,}35$ ℃ na faixa de 0 a 100 ℃. Portanto, atende ao requisito *[RF04]*, pois mede a faixa de temperatura especificada com uma precisão mínima de $0{,}35\,\%$.

Figura 41 – Sensor de temperatura RTD $PT100$.

Fonte: adaptado de Omega (2013).

Apresentado na Figura 41, esse sensor é construído na topologia 4 fios, mostrada na Figura 42. Dessa forma, dois fios são utilizados para injetar corrente (EX+ e EX-), enquanto os outros dois medem a queda de tensão na resistência do RTD. Com isso, não há corrente circulando pelos cabos de medição nem queda de tensão na resistência dos cabos, e a tensão medida é exclusivamente aquela apresentada nas extremidades do sensor.

Figura 42 – Topologia 4-fios para medição de sensores de temperatura RTD.



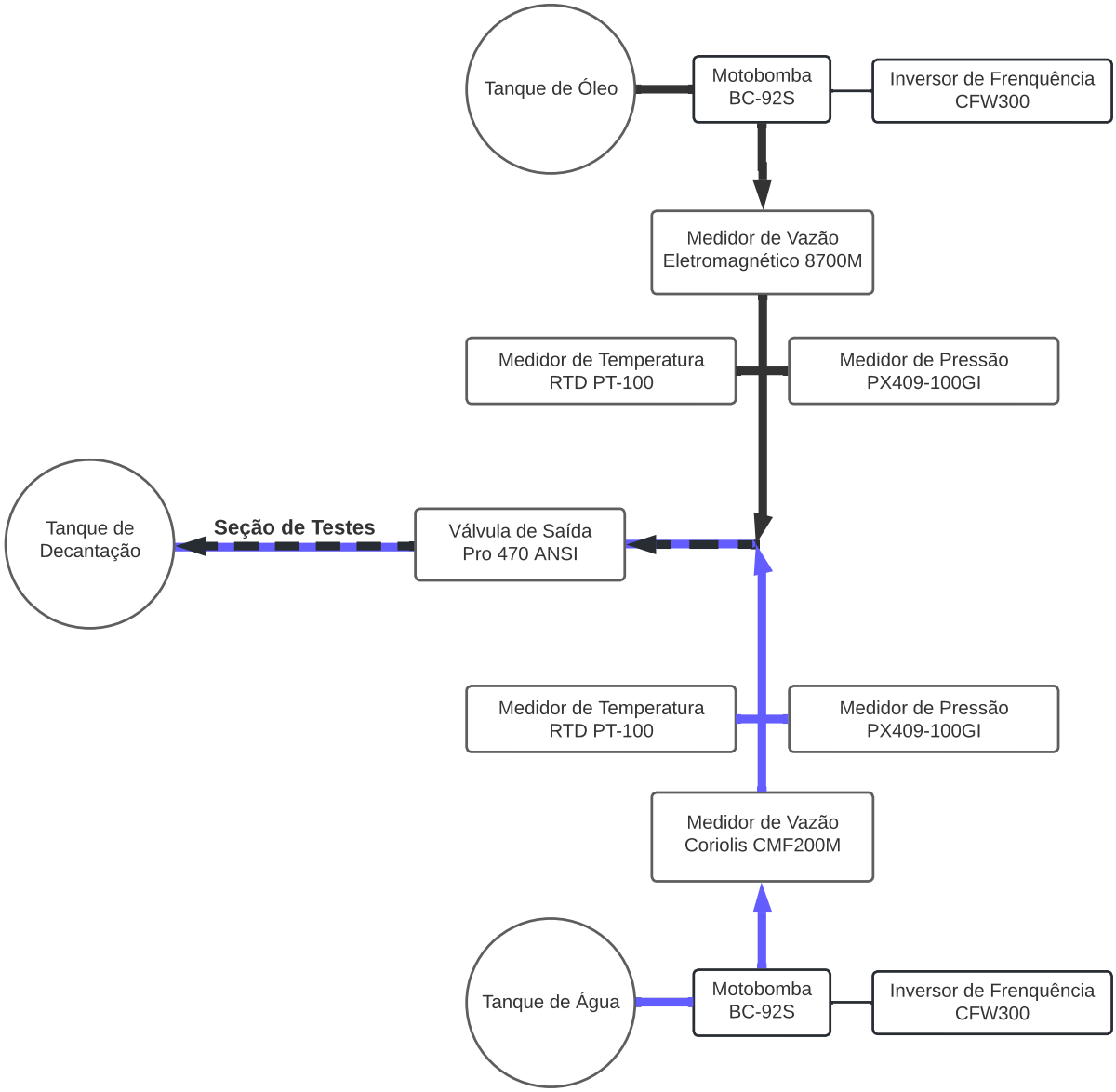
Fonte: autor (2022).

Para a conversão dos sinais de tensão para temperatura, o fabricante informa os coeficientes $A = 3{,}811 \times 10^{-3}$, $B = -580{,}191 \times 10^{-9}$ e $C = -4{,}27311 \times 10^{-12}$, utilizados nas Equações 16 e 17.

4.3 SISTEMA MECÂNICO

Com os principais componentes do sistema definidos, é possível modelar a estrutura mecânica e posicionamento dos equipamentos. O sistema mecânico deve fixar todos os instrumentos de medição e atuadores, além dos tanques para armazenamento dos fluidos e tubulações.

Figura 43 – Diagrama do posicionamento dos equipamentos.

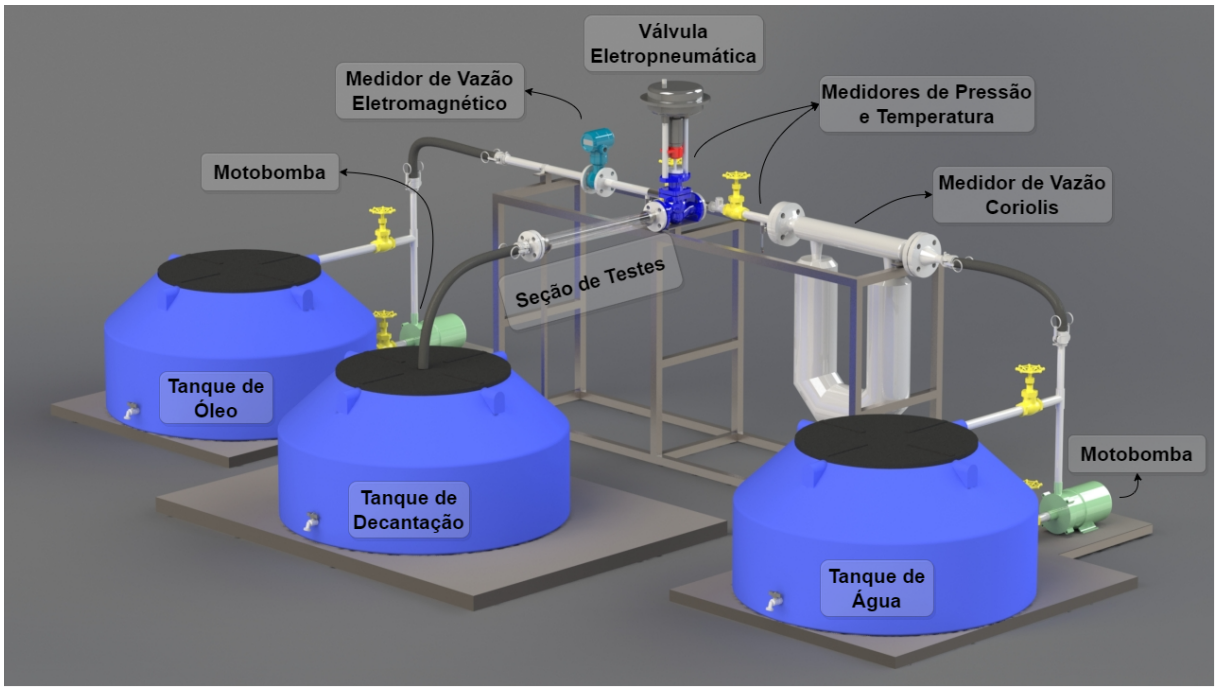


Fonte: autor (2022).

A Figura 43 mostra o diagrama estrutural da bancada. As motobombas, controladas pelos inversores de frequência, irão bombear os fluidos simultaneamente através das tubulações. Medidores de vazão são posicionados no inicio de cada escoamento, coriolis CMF200M no ramal de água e eletromagnético $8700M$ no ramal de óleo. Pressão e temperatura serão medidas no mesmo ponto da tubulação, utilizando os instrumentos PX409-100GI e RTD PT-100, respectivamente. Na saída, após a união dos escoamentos, será posicionada a válvula eletropneumática, seguida pela seção de testes e tanque de decantação.

Através do diagrama acima, foi possível modelar a estrutura mecânica. O projeto da bancada é apresentado na Figura 44.

Figura 44 – Projeto mecânico da bancada.



Fonte: autor (2022).

Para atender aos requisitos *[RF08]* e *[RF09]*, foram utilizados 3 tanques de $310\,\mathrm{L}$. A seção de testes disponibilizada após a união dos escoamentos, contém $500\,\mathrm{mm}$ de comprimento e $50\,\mathrm{mm}$ de diâmetro, atendendo ao requisito *[RNF01]*. Tubulações de PVC e aço inox com $25{,}4\mathrm{mm}$ de diâmetro foram utilizadas nos ramais de água e óleo, respectivamente. Válvulas de retenção foram posicionadas a fim de evitar o contra-fluxo nos ramais antes da emulsão. A interligação do conjunto dos tanques e motobombas com o restante da bancada irá ocorrer por meio de mangueiras flexíveis, facilitando o posicionamento e manutenção.

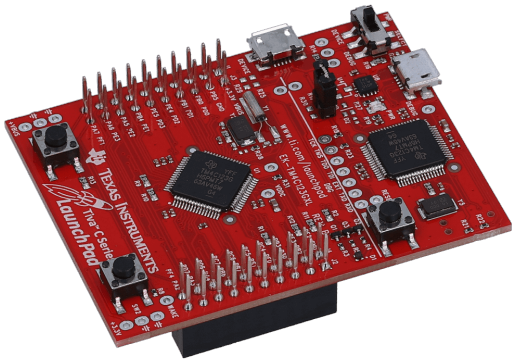
4.4 HARDWARE

O hardware deverá realizar a interface entre os equipamentos da bancada e o computador de controle que, segundo *[RF04]*, deverá ocorrer pelas portas USB. Para isso, o hardware será apresentado em duas partes, divididas entre controle dos atuadores aquisição de dados.

4.4.1 Controle dos atuadores

A integração entre o computador e os demais componentes de hardware para controle dos atuadores será realizada através da plataforma de avaliação TM4C123GH6PM, contendo o microcontrolador ARM Cortex-M4F (Figura 45). Esse microcontrolador possui 256kB de memória flash e 32kB de SRAM, disponibilizando canais de comunicação serial como UART e I2C.

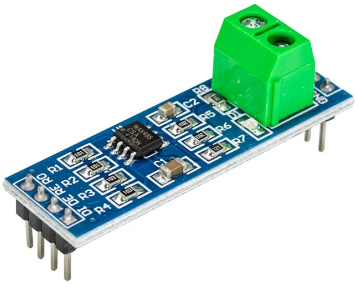
Figura 45 – Microcontrolador TIVA TM4C123GH6PM.



Fonte: Texas Instruments (2014).

Para enviar comandos aos inversores de frequência e controlar a velocidade das motobombas, será preciso enviar as mensagens do protocolo Modbus no padrão RS485. Um conversor TTL (Transistor-Transistor Level) para RS485 será utilizado para transmitir os comandos recebidos do microcontrolador para os inversores (Figura 46). Esse módulo conversor recebe as mensagens via UART, com nível de tensão de 0 a $5\,\mathrm{V}$, e as converte para RS485, como um sinal diferencial de tensão de $-5\,\mathrm{V}$ a $5\,\mathrm{V}$.

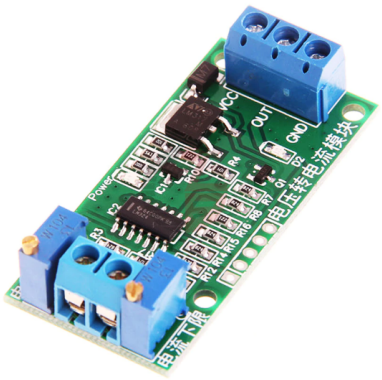
Figura 46 – Módulo conversor TTL-RS485.



Fonte: Maxim Integrated (2014).

O controle da válvula eletropneumática será através de sinais de 4 a $20\,\mathrm{mA}$. Para gerar os sinais em corrente, o hardware contará com o conversor apresentado na Figura 47. O módulo recebe níveis de tensão de 0 a $5\,\mathrm{V}$ e converte-os para a faixa de 4 a $20\,\mathrm{mA}$. A alimentação da malha de saída suporta tensões de 5 a $30\,\mathrm{V}$, que será disponibilizada por uma fonte externa de $24\,\mathrm{V}$ e $1{,}3\,\mathrm{A}$. O ajuste do ganho e offset da entrada pode ser feito pelos trimpots ao lado dos bornes de entrada, garantindo que os sinais de $0\,\mathrm{V}$ e $5\,\mathrm{V}$ correspondam efetivamente a $4\,\mathrm{mA}$ e $20\,\mathrm{mA}$.

Figura 47 – Módulo conversor 0 a $5\,\mathrm{V}$ para 4 a $20\,\mathrm{mA}$.



Fonte: PDA (2018).

Para gerar o sinal analógico de tensão na entrada do conversor, será empregado um DAC modelo MCP4725 (Figura 48). Recebendo comandos via I2C, esse componente representa a tensão de saída com uma variável de 12 bits, o que possibilita um incremento mínimo de $1{,}22\mathrm{mV}$, correspondendo a aproximadamente $0{,}02\%$ na abertura da válvula:

$$\Delta V_{min} = \frac{5\,\mathrm{V}}{2^{12} - 1} = \frac{5\,\mathrm{V}}{4095} = 1{,}22\mathrm{mV}$$

$$R_{12bits} = \frac{4095 \times R_{\%}}{100} \tag{25}$$

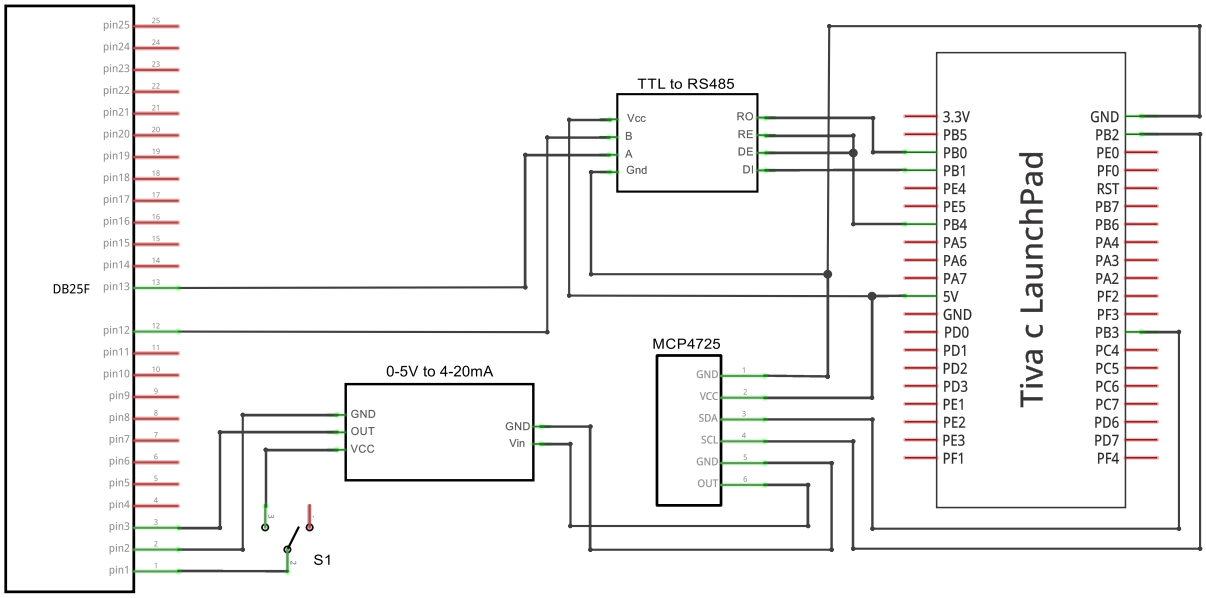
A Equação 25 mostra o cálculo para converter a referência de abertura da válvula, de 0% a 100%, para a referência de tensão em 12 bits.

Figura 48 – Conversor digital analógico MCP4725.

Fonte: Microchip (2022).

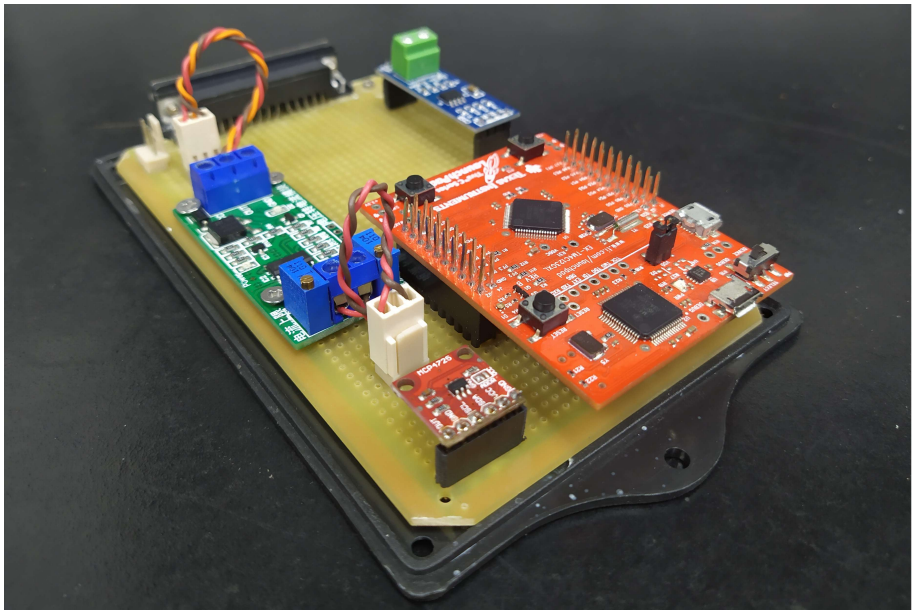
O projeto do hardware para controle dos atuadores é apresentado na Figura 49. A conexão com o computador será feita por um cabo USB, cumprindo o requisito *[RNF04]*. Os comandos recebidos pelo microcontrolador serão enviados aos componentes via comunicação UART e I2C. Após as conversões, a conexão com os equipamentos da bancada será através de um conector DB25. Uma chave on-off fará a conexão entre a fonte externa de $24\,\mathrm{V}$ e o conversor de 4 a $20\,\mathrm{mA}$.

Figura 49 – Projeto do hardware para controle dos atuadores.



Fonte: autor (2022).

Figura 50 – Montagem do hardware para controle dos atuadores.



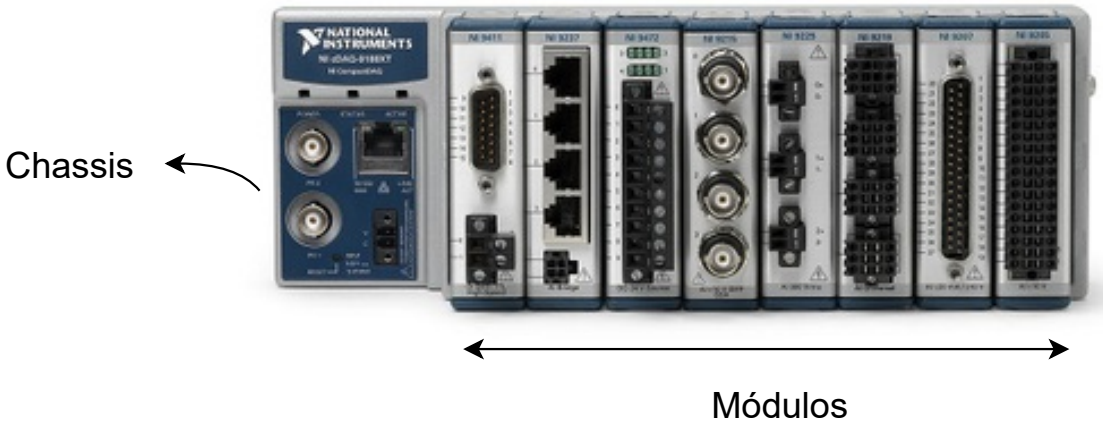
Fonte: autor (2022).

A montagem é apresentada na Figura 50 foi realizada no interior de uma caixa plástica, a fim de ser posicionada ao lado do computador de controle.

4.4.2 Aquisição de dados

A interface entre os instrumentos de medição e o computador será através da plataforma de aquisição de dados *CompactDAQ*. A plataforma se conecta ao computador por uma porta USB, e é formada por um chassis conectado a módulos intercambiáveis de aquisição de dados (Figura 51).

Figura 51 – Plataforma de aquisição de dados CompacDAQ.



Fonte: adaptado de National Instruments (2021).

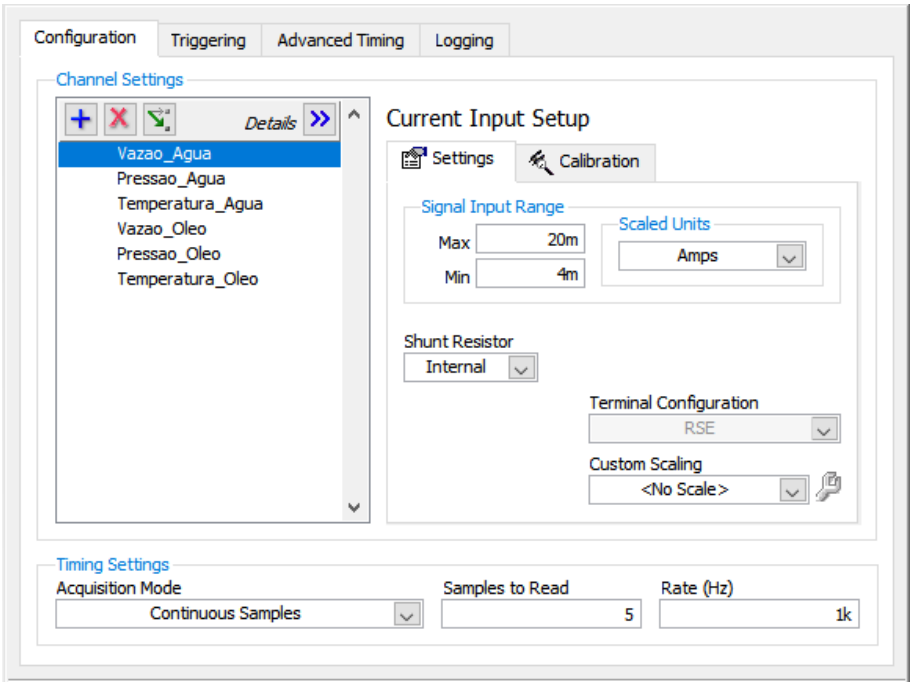
Para leitura dos sinais de corrente, provenientes dos sensores de vazão e pressão, será utilizado o módulo *NI9203*. Esse módulo implementa o circuito da Figura 11, suportando correntes na faixa de $\pm 20\,\mathrm{mA}$, contendo 8 canais de entrada e taxa de amostragem de até $200\,\mathrm{kS/s}$.

A malha de corrente das medições de pressão será alimentada pela mesma fonte de $24\,\mathrm{V}$ e $1{,}3\,\mathrm{A}$ utilizada no loop de corrente da válvula eletropneumática. A fonte de energia dos medidores de vazão é interna aos transmissores, sendo derivada da alimentação de $220\,\mathrm{V}$.

Para os RTDs, será empregado o módulo *NI9216*, específico para tais instrumentos. Possuindo a topologia apresentada na Figura 42, esse módulo suporta variações de resistência de $0\,\Omega$ a $400\,\Omega$, disponibilizando 8 canais de entrada com taxa de amostragem de até $400\,\mathrm{kS/s}$.

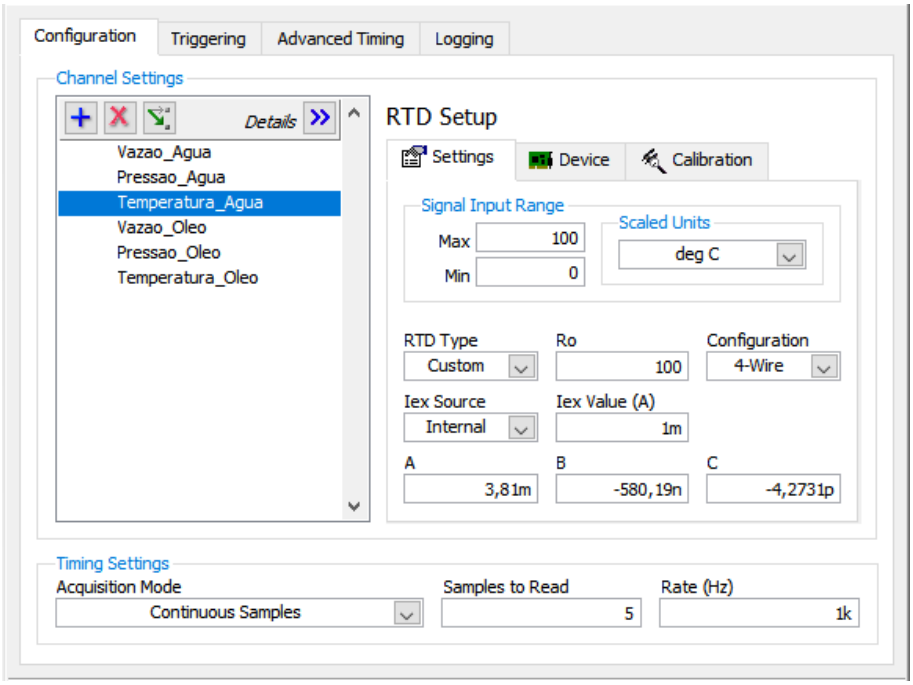
A configuração de cada entrada é feita através do software da plataforma de aquisição de dados. Neste trabalho, foi adotado como padrão a taxa de amostragem de $1\,\mathrm{kHz}$, atendendo a dinâmica do sistema, que ocorre na ordem dos segundos. Em cada canal, são realizadas cinco leituras antes enviá-las em um buffer para o LabView. Para as entradas de corrente, a configuração é apresentada na Figura 52, indicando os limites de medição de $4\,\mathrm{mA}$ a $20\,\mathrm{mA}$ e o resistor shunt, para realizar a conversão do sinal de corrente para tensão, interno ao equipamento de medição.

Figura 52 – Configuração das entradas de corrente.



Fonte: autor (2022).

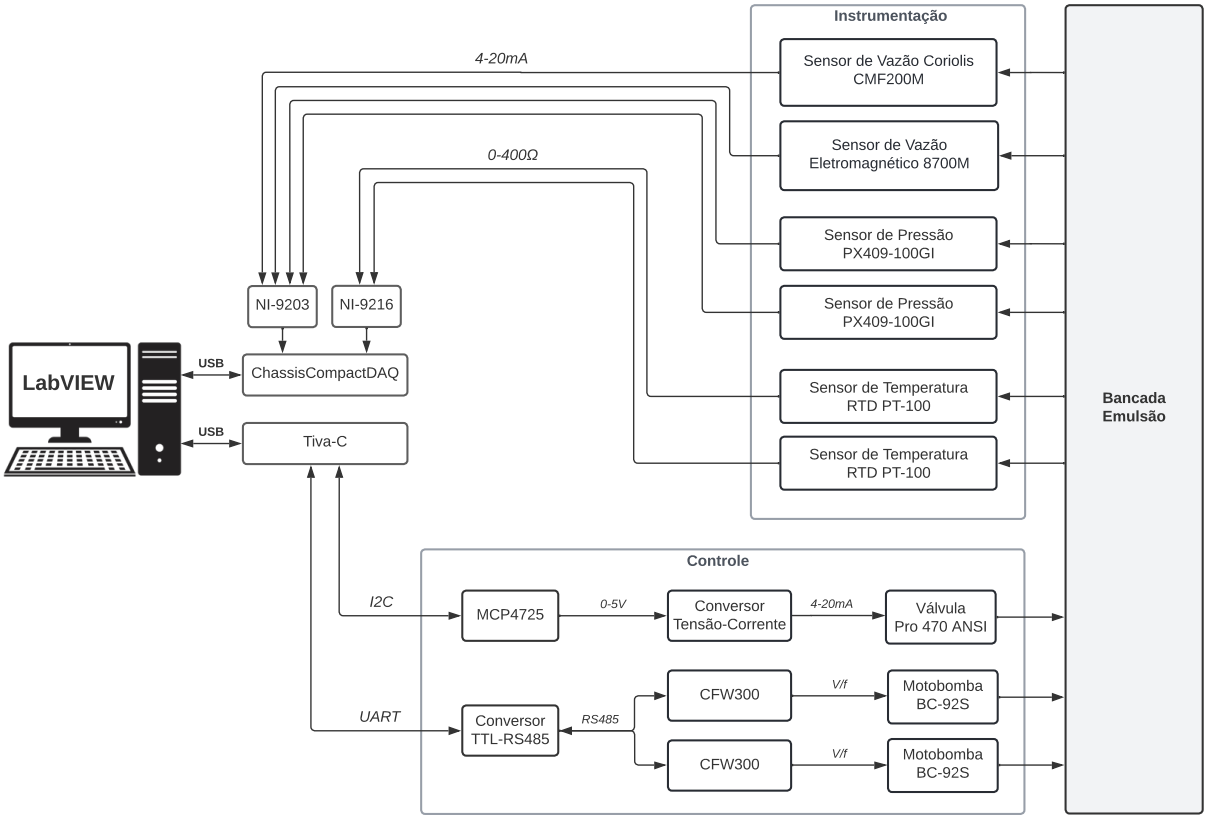
Figura 53 – Configuração das entradas de RTD.



Fonte: autor (2022).

A Figura 53 apresenta a configuração das entradas de RTD. A faixa de temperatura a ser medida é definida de 0 ℃ a 100 ℃. A topologia de 4 fios é selecionada e os parâmetros R_0, A, B e C são ajustados de acordo com o RTD *PT100* utilizado. O módulo $NI9216$ fornece a alimentação do instrumento, sendo utilizada uma corrente de excitação de $1\,\mathrm{mA}$.

Figura 54 – Diagrama das comunicações.



Fonte: autor (2022).

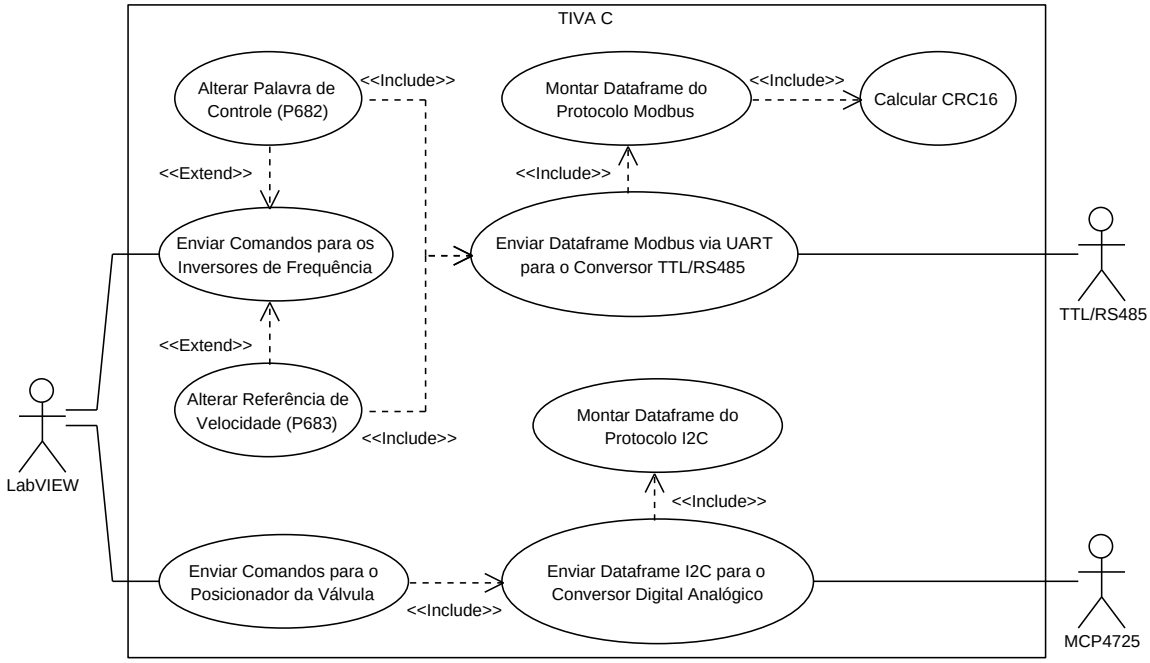
A Figura 54 apresenta uma visão geral das comunicações entre os equipamentos da bancada. As leituras serão realizadas pelo LabView, de acordo com as configurações realizadas para cada canal. Para os canais de corrente, a leitura será entregue em ampere, já a leitura dos RTDs é previamente convertida, aplicando a Equação 17, e entregue em graus Celsius.

4.5 FIRMWARE

O firmware, código em C++ embarcado no microcontrolador, terá a função de receber os comandos enviados pelo LabView, tratá-los e enviá-los aos devidos componentes a fim de controlar os atuadores e efetivar a ação desejada pelo usuário. Para mapear as possíveis interações entre o software de interface com o usuário e o firmware embarcado, a Figura 55 apresenta os possíveis casos de uso.

O LabView enviará comandos ao microcontrolador via comunicação serial, estabelecida por uma porta USB. Os comandos destinados aos inversores de frequência serão divididos entre controle do estado das motobombas, ligadas ou desligadas, e alteração da referência de velocidade, escrevendo nos registradores $P682$ e $P683$. A conversão da referência, apresentada na Equação 19, é previamente realizada pelo LabView. Em ambos os casos, será necessário calcular o CRC16, montar o dataframe do protocolo Modbus e enviá-lo ao conversor TTL/RS485 via UART.

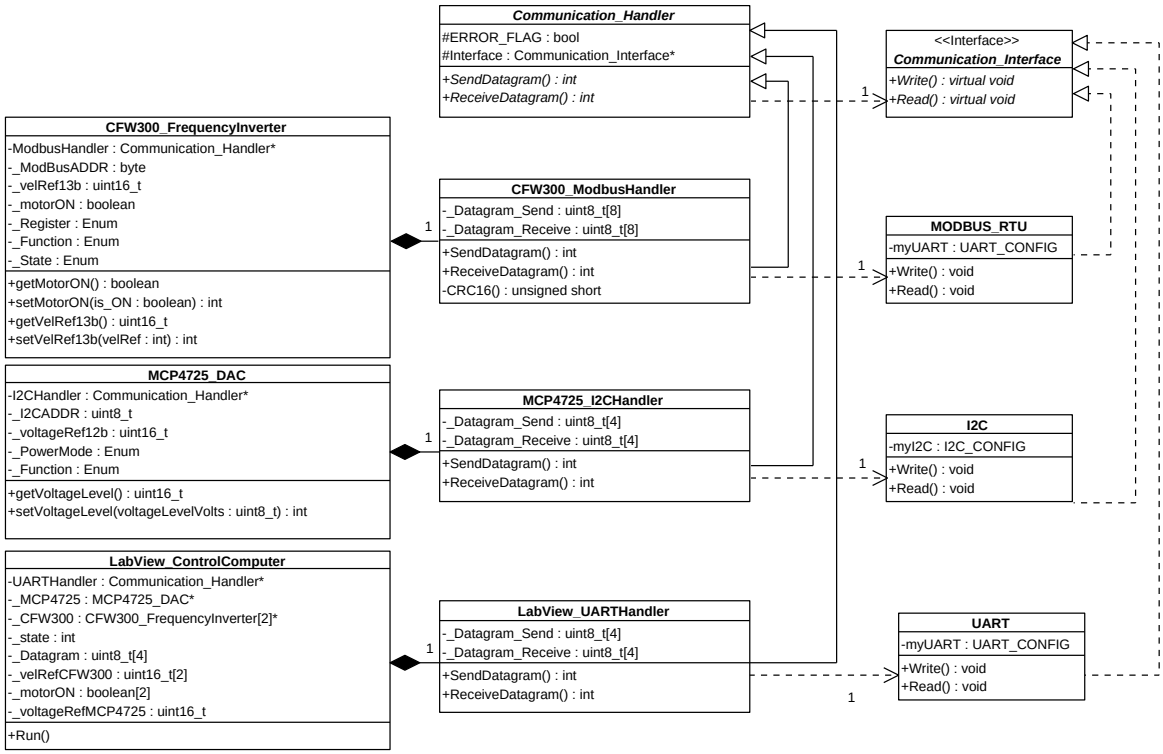
Figura 55 – Casos de uso do firmware.



Fonte: autor (2022).

Para comandar a válvula, o microcontrolador enviará via I2C a referência de tensão em 12 bits, já calculada pelo LabView com a Equação 25. Para isso é necessário montar as mensagens no formato do protocolo e transmiti-las ao MCP4725. O diagrama de classes elaborado para atender os casos casos de uso é apresentado na Figura 56.

Figura 56 – Diagrama de classes.



Fonte: autor (2022).

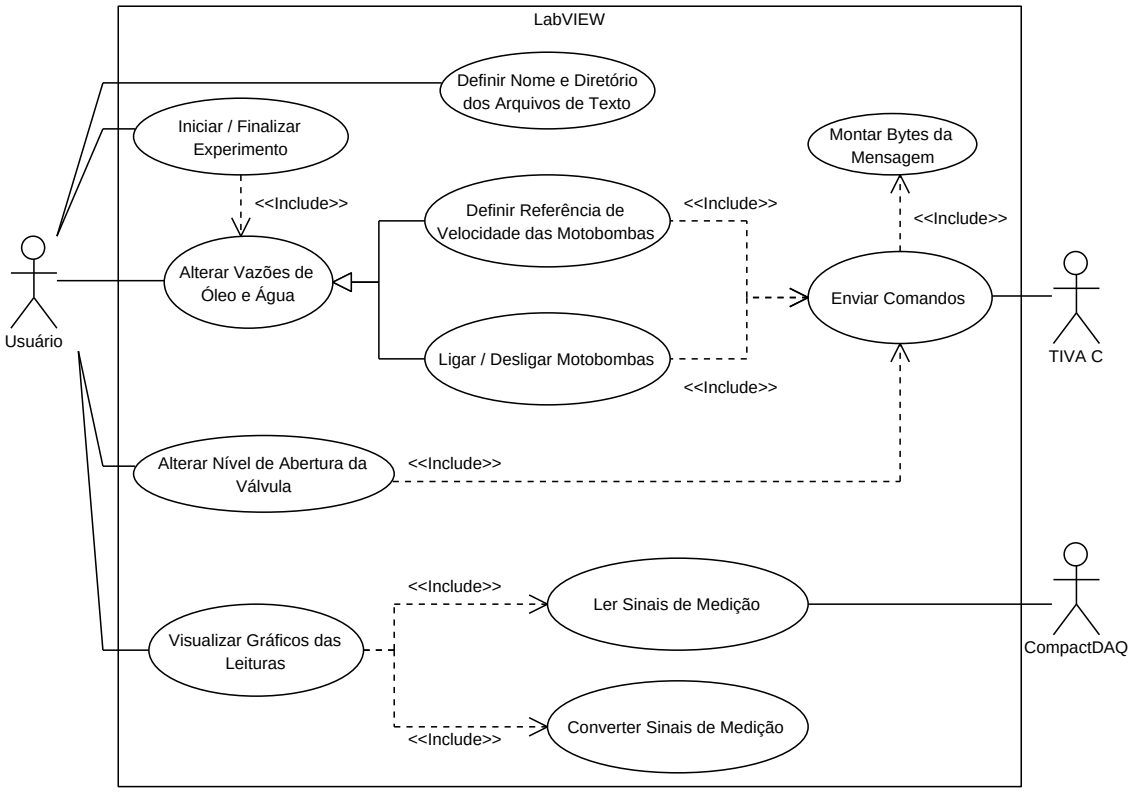
Cada agente apresentado nos casos de uso é representado por uma classe. O canal de comunicação de cada um dos agentes é representado por uma interface de comunicação, que implementa os métodos virtuais da classe base *Communication_Interface*. Essas classes lidam com as nuances da escrita e leitura nas interfaces de comunicação, como UART e I2C.

As classes derivadas da *Communication_Handler* são dependentes das interfaces de comunicação, fazendo uso de polimorfismo para invocar os métodos de leitura e escrita. Elas fornecem aos agentes a abstração dos protocolos de comunicação, montando os respectivos datagramas e, no caso dos inversores de frequência, também calculam o CRC16. As implementações são apresentadas no Apêndice A.

4.6 SOFTWARE

Da mesma forma abordada anteriormente, a Figura 57 mostra os casos de uso do software em LabView, que fará a interface com o usuário. As principais funcionalidades serão a inicialização ou finalização do experimento, visualização das leituras e alteração das vazões de água e óleo e do nível de abertura da válvula. As interações com os atuadores estão concentradas na comunicação com o microcontrolador, enquanto as medições são realizadas através da plataforma de aquisição de dados.

Figura 57 – Casos de uso do LabView.

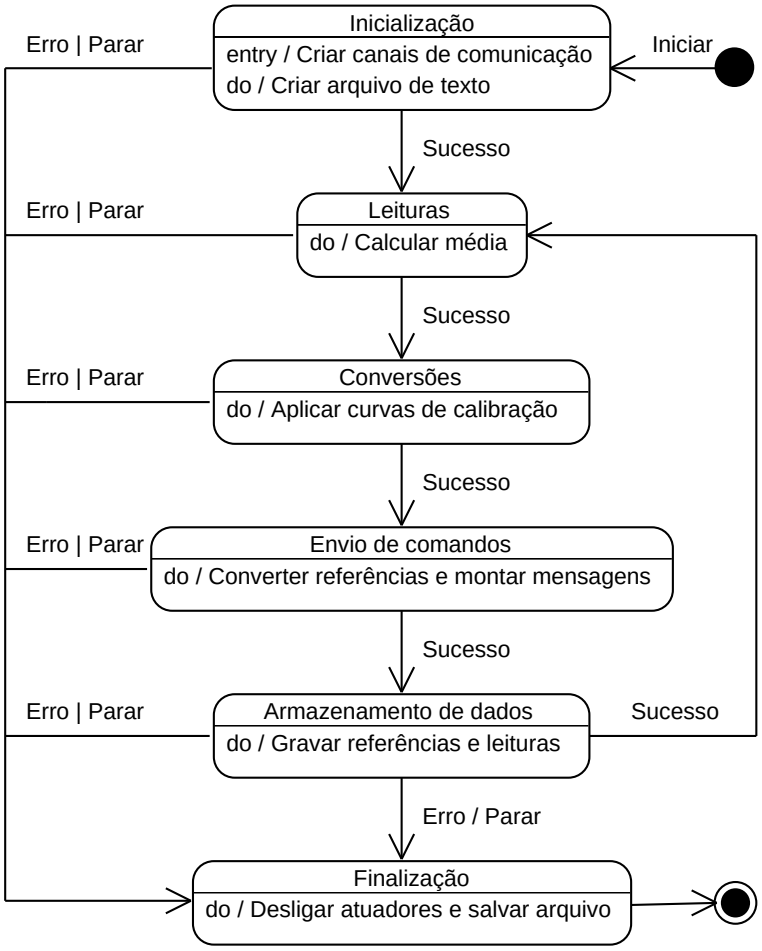


Fonte: autor (2022).

O fluxo de execução foi desenvolvido com uma máquina de estados, apresentada na Figura 58. Ao inicializar a execução, são criados os canais de comunicação e o arquivo de texto, onde serão armazenados os dados dos experimentos. As leituras são realizadas na sequência, calculando a média dos cinco valores recebidos de cada canal.

No próximo estado, as leituras são convertidas para unidades de engenharia de acordo com as curvas de calibração apresentadas, atendendo ao *[RF11]*. Em seguida, os comandos são enviados ao microcontrolador. A estrutura de mensagens utilizada entre o software e o firmware é apresentada na Figura 59.

Figura 58 – Máquina de estados do LabView.



Fonte: autor (2022).

O primeiro byte da mensagem é a função a ser executada pelo firmware. Podendo ser V, para alterar a velocidade das motobombas, L ou D para ligar ou desligar as motobombas e A para definir a abertura da válvula. O segundo campo identifica cada equipamento, sendo 1 ou 2 no caso das motobombas, e fixado em 1 para a válvula de saída. O último campo da mensagem possui dois bytes e indica o valor a ser escrito. Para as motobombas esse valor é a referência de velocidade em 13 bits (Equação 19), já para a válvula, é o valor de tensão em 12 bits (Equação 25),

escrito no MCP4725. Com isso, cumpre-se o requisito *[RF12]*.

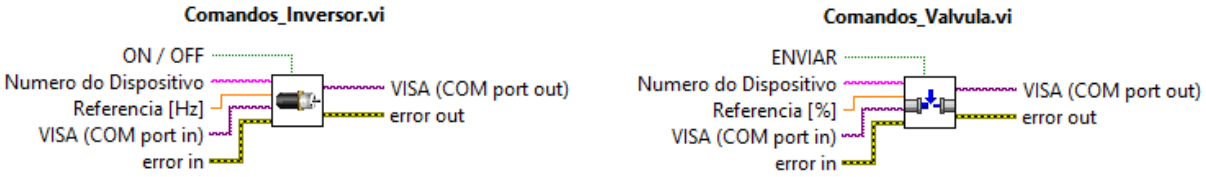
Figura 59 – Estrutura das mensagens entre o software e o firmware.

Data Frame LABVIEW -> TIVA		
Função	**Dispositivo**	**Valor**
1byte	1byte	2bytes
|0xXX|	**|0xXX|0xXX|**	**|0xXX|**
Código da funcionalidade (A, D, L, V)	Número do dispositivo (usado para distinguir dispositivos da mesma classe)	Valor a ser enviado (altera parâmetros do dispositivo)

Fonte: autor (2022).

Para cada atuador foi criado um sub-vi (Figura 60), recebendo as referências de entrada do usuário, realizando as conversões, montando as mensagens e enviando-as ao TIVA. Sua utilização é feita no estado *Envio de comandos*, apresentado parcialmente na Figura 61.

Figura 60 – SubVI's para controlar os atuadores.

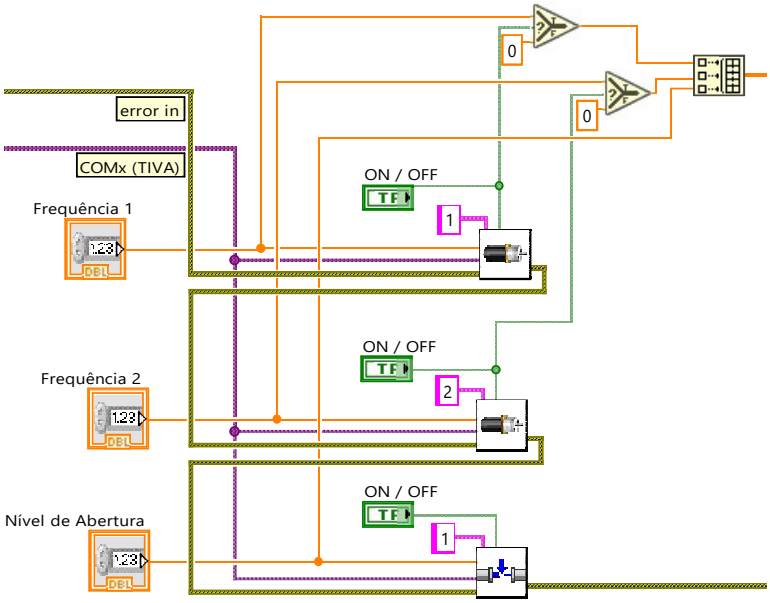


(a) Função para envio de comandos aos inversores de frequência.

(b) Função para envio de comandos à válvula.

Fonte: autor (2022).

Figura 61 – Envio de comandos aos atuadores.

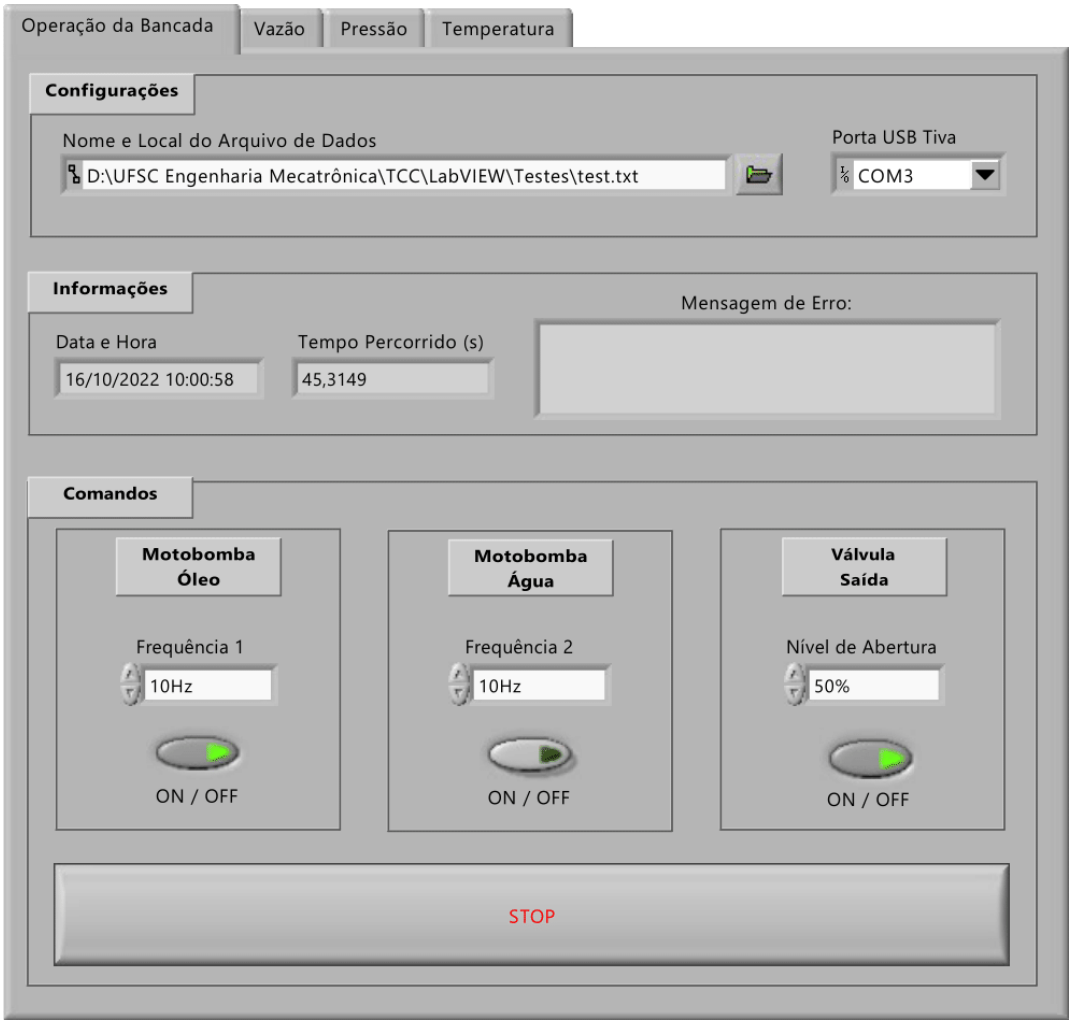


Fonte: autor (2022).

A tela de interface com o usuário é mostrada na Figura 62. No topo da tela é possível escolher o nome e diretório do arquivo de texto, atendendo ao *[RF13]* e *[RNF05]*. Outra configuração necessária é a seleção da porta USB na qual o microcontrolador está conectado. A detecção da porta conectada à plataforma de aquisição de dados é feita automaticamente pelo LabView. Sendo assim, completa-se o requisito *[RNF04]*, realizando todas as comunicações com a bancada de testes através das portas USB. Abaixo das configurações são exibidas informações como data e hora, tempo de execução do experimento e possíveis mensagens de erro.

Na aba de comandos o usuário poderá controlar o estado dos atuadores, definindo a frequência de cada motobomba e o nível de abertura da válvula. As motobombas podem ser ligadas ou desligadas através dos botões *ON/OFF*, que indicam visualmente o estado atual, atendendo o *[RNF04]*. O botão da válvula permite a desabilitação temporária do envio de comandos à válvula. Abaixo dos comandos é disponibilizado o botão para finalizar o experimento, conforme *[RNF07]*, desligando as motobombas e salvando o arquivo de dados.

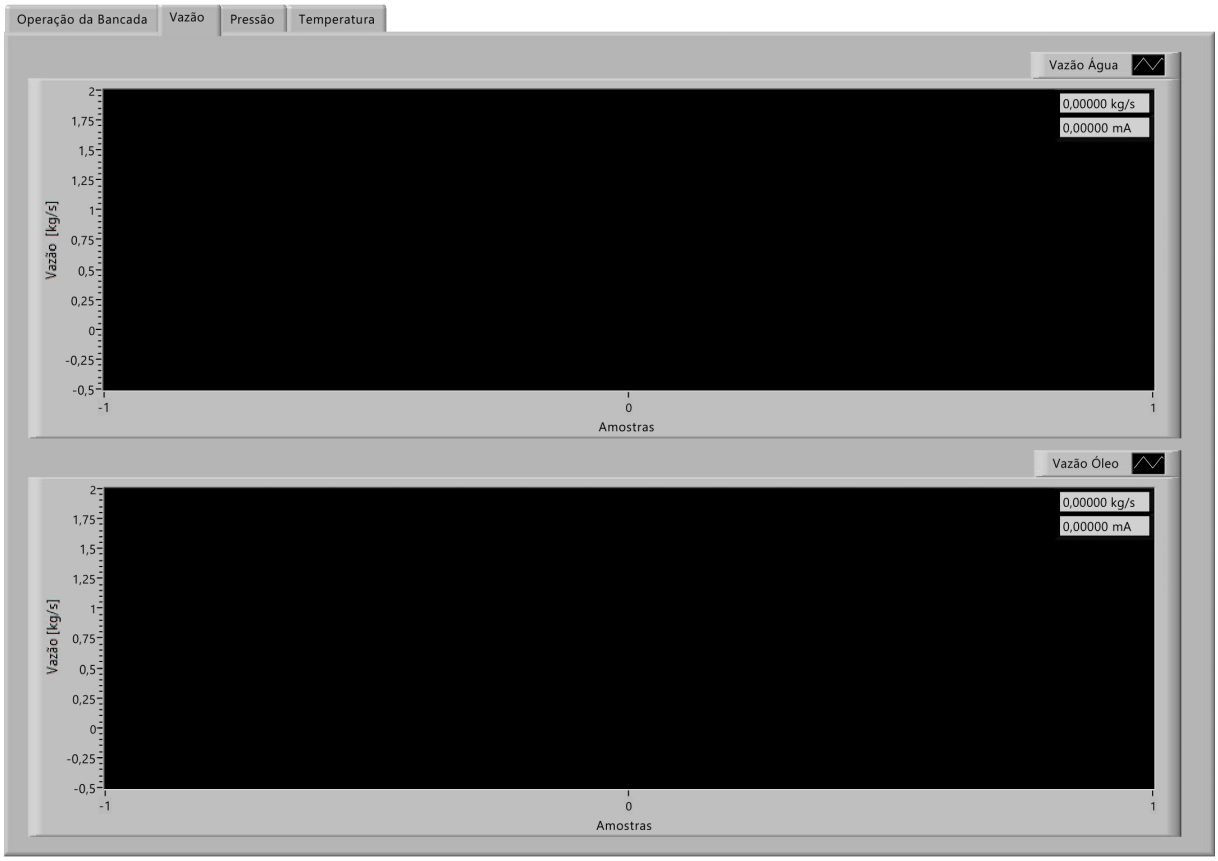
Figura 62 – Tela principal da interface gráfica com o usuário.



Fonte: autor (2022).

Por fim, os gráficos das leituras podem ser visualizados nas demais abas da interface. Separados de acordo com o tipo da medição e ramal, conforme *[RF14]*. As leituras e valores convertidos são exibitos também em um display digital no canto superior direito dos gráficos.

Figura 63 – Gráficos das medições.



Fonte: autor (2022).

A Figura 63 apresenta os gráficos de vazão. O mesmo modelo de gráfico foi utilizado nas demais abas. O desenvolvimento completo do software em LabView é apresentado no Apêndice B.

5 RESULTADOS

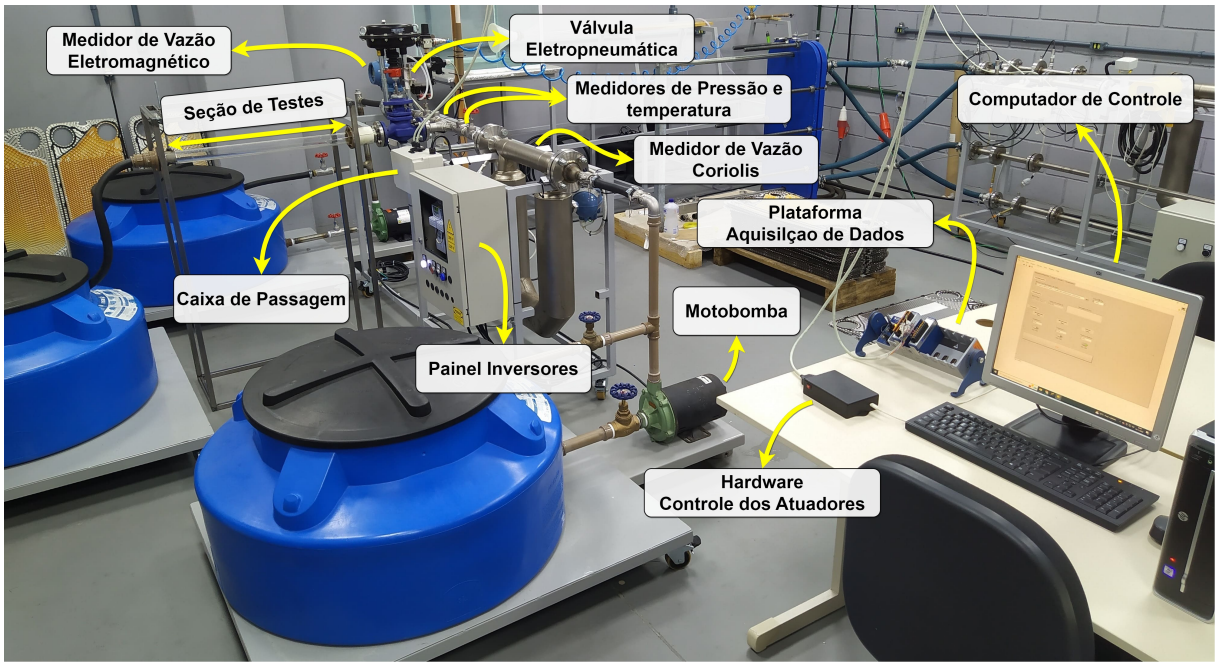
Neste capítulo serão apresentados os resultados obtidos e testes realizados para avaliar o comportamento da bancada experimental e seu desempenho em relação aos requisitos.

5.1 MONTAGEM DO SISTEMA MECÂNICO

A estrutura, tubulação e posicionamento dos equipamentos da bancada foram realizados seguindo o projeto apresentado na Seção 4.3. Para facilitar o posicionamento e movimentação, rodízios foram adicionados na base das estruturas. A conexão entre as plataformas que contém os tanques e motobombas e o restante da bancada ocorre por meio de mangueiras flexíveis, conforme previsto. Tubulações de PVC foram utilizadas entre a motobomba e o tanque de água. No ramal de óleo, somente tubulações de aço inox.

Também foram fixadas à estrutura da bancada o painel elétrico, contendo os inversores, e uma caixa de passagem, para realizar as conexões com o hardware. Dois cabos manga de 20 vias 26 AWG (American Wire Gauge) fazem a interligação entre os equipamentos da bancada e os componentes de hardware, apresentados na Seção 4.4 e posicionados ao lado do computador de controle.

Figura 64 – Montagem da bancada experimental.



Fonte: autor (2022).

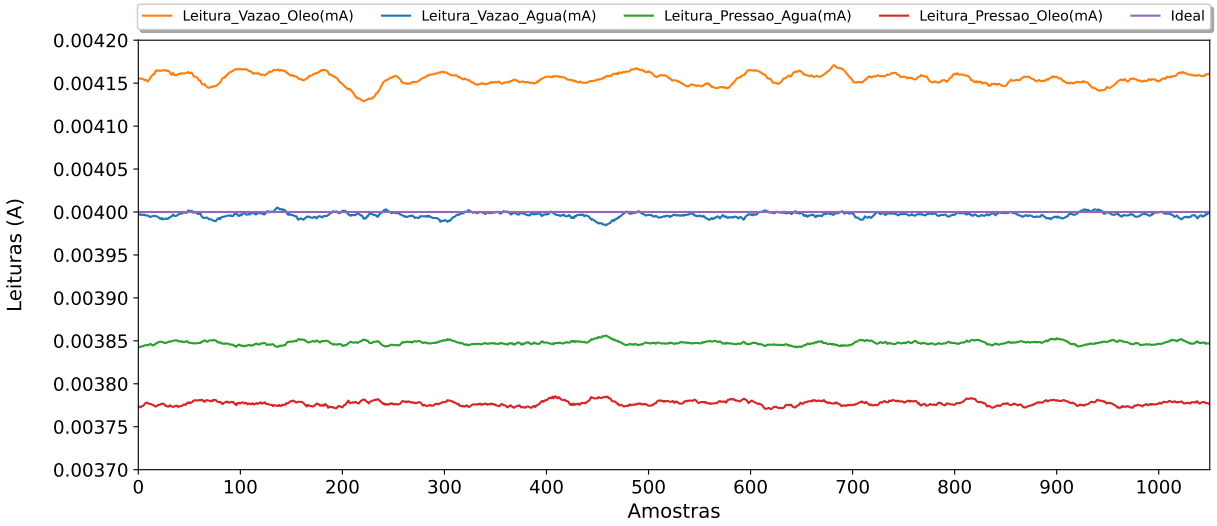
A Figura 64 apresenta uma visão geral da bancada experimental.

5.2 AQUISIÇÃO DE DADOS

Testes preliminares com os instrumentos de medição indicaram erros sistemáticos nas leituras, com offsets entre os valores lidos e aqueles indicados pelos fabricantes no ponto zero das medições. Para corrigir tais erros, foram coletadas 1000 amostras dos medidores de vazão e pressão em condições nulas de operação, ou seja, sem vazão e com pressão atmosférica. Cada amostra corresponde à média de 5 leituras recebidas da plataforma CompacDAQ.

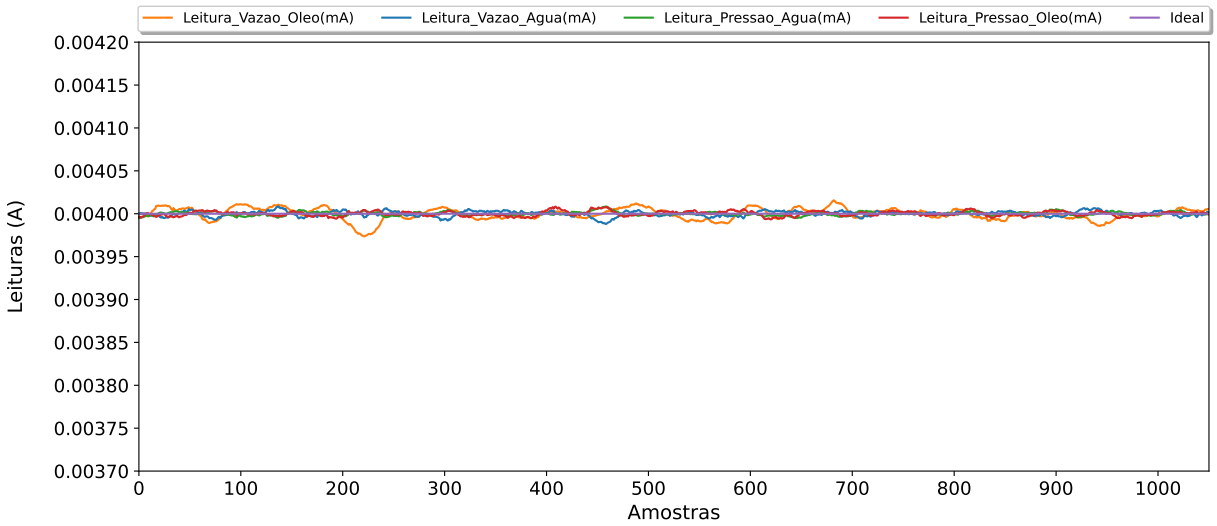
A diferença entre a média do conjunto de amostras e o valor de $4\,\mathrm{mA}$, esperado no ponto zero das medições, indica o bias a ser corrigido nas leituras. A correção foi realizada no tratamento dos dados com a Equação 26, onde S_A^* é a leitura corrigida.

Figura 65 – Medições de vazão e pressão no ponto zero antes da correção do bias.



Fonte: autor (2022).

Figura 66 – Medições de vazão e pressão no ponto zero após a correção do bias.



Fonte: autor (2022).

As Figuras 65 e 66 apresentam as medições antes e após a correção dos erros sistemáticos. A Tabela 3 mostra a média e desvio padrão das 1000 amostras para cada instrumento de medição e o respectivo bias, que foi corrigido nos demais testes.

Tabela 3 – Parâmetros das medições em condições nulas de operação.

Instrumento de medição	Média (A)	Desvio Padrão (A)	Bias (A)
Eletromagnético	$4{,}15539 \times 10^{-3}$	$6{,}72 \times 10^{-6}$	$155{,}39 \times 10^{-6}$
Coriolis	$3{,}99654 \times 10^{-3}$	$2{,}92 \times 10^{-6}$	$-3{,}45 \times 10^{-6}$
PX409 Ramal Água	$3{,}84759 \times 10^{-3}$	$2{,}07 \times 10^{-6}$	$-152{,}40 \times 10^{-6}$
PX409 Ramal Óleo	$3{,}77720 \times 10^{-3}$	$2{,}73 \times 10^{-6}$	$-222{,}80 \times 10^{-6}$

Fonte: autor (2022).

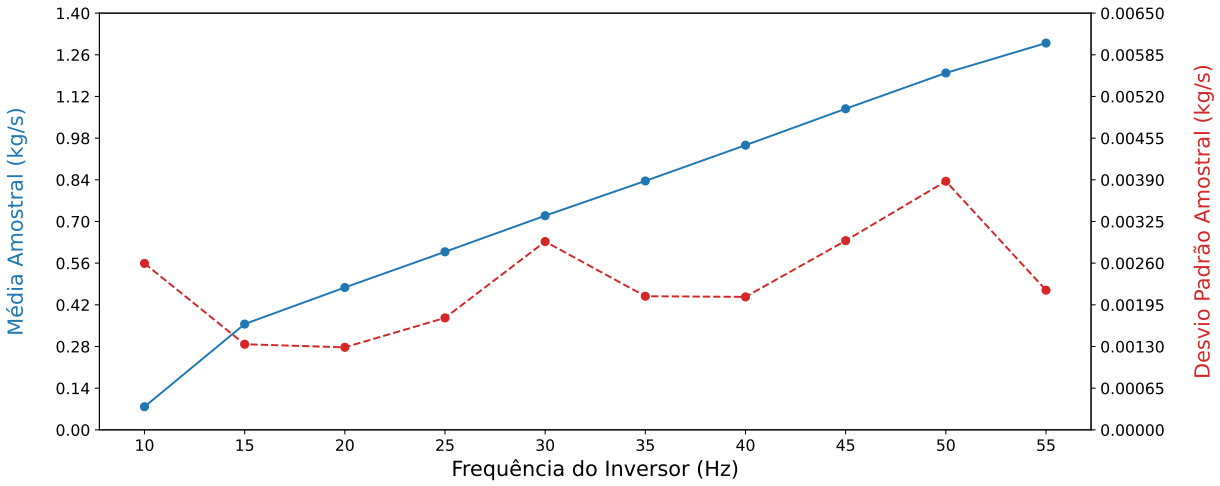
$$S_A^* = S_A - Bias \tag{26}$$

As medições dos RTDs são enviadas ao LabView previamente convertidas pelo sistema de aquisição para graus Célsius, com a Equação 17. Por não dispor de um método mais preciso para medir e controlar a temperatura, não foram realizadas as correções do possível bias dos RTDs.

Análises em relação à precisão das medições de vazão foram realizadas ao longo da faixa de operação das motobombas. Utilizando água em ambos os ramais, foram coletadas 1000 amostras em 10 pontos de operação, variando a frequência dos inversores de $10\,\mathrm{Hz}$ a $55\,\mathrm{Hz}$ com incrementos de $5\,\mathrm{Hz}$. Sendo $10\,\mathrm{Hz}$ a frequência mínima na qual a motobomba é capaz de gerar vazões pela tubulação e $55\,\mathrm{Hz}$ a frequência máxima de operação adotada nos testes.

As leituras passam pela correção do bias, com a Equação 26, e são convertidas para unidades de engenharia pelas Equações 21 e 22.

Figura 67 – Média e desvio padrão das medições realizadas pelo medidor de vazão eletromagnético no ramal de óleo.

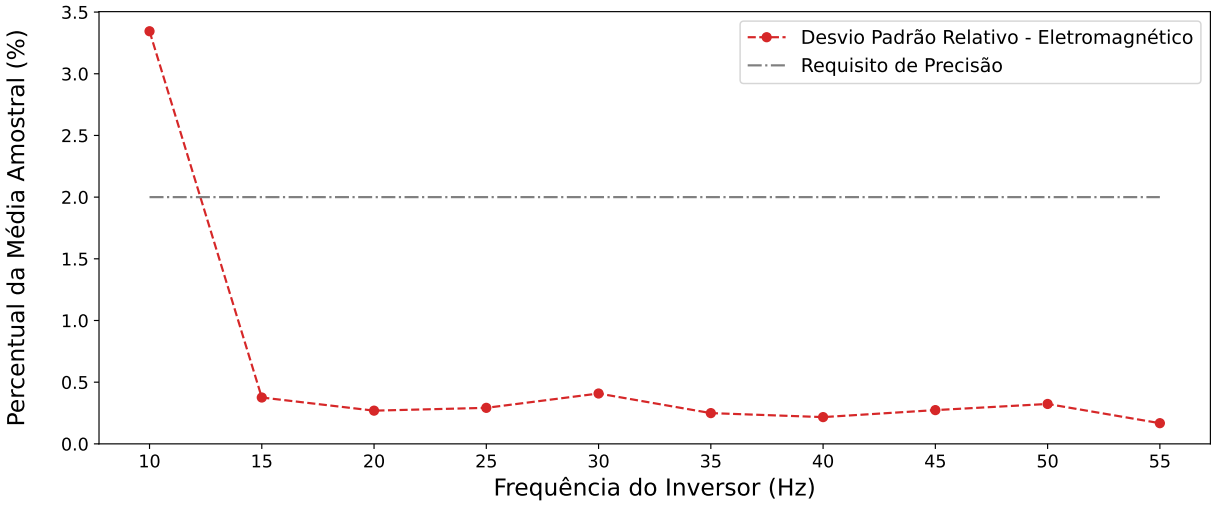


Fonte: autor (2022).

Para o medidor de vazão eletromagnético, os resultados são apresentados na Figura 67, onde a média e o desvio padrão são mostrados em cada ponto de operação. É possível notar que o ganho do sistema possui valores distintos ao longo da faixa de operação, sendo $55{,}54 \times 10^{-3}\,(\mathrm{kg/s})/\mathrm{Hz}$, entre $10\,\mathrm{Hz}$ e $15\,\mathrm{Hz}$, e $23{,}79 \times 10^{-3}\,(\mathrm{kg/s})/\mathrm{Hz}$ nos demais pontos.

Já a Figura 68 apresenta o desvio padrão como percentual da média amostral. A comparação entre a precisão obtida e a especificada pelo *[RF02]*, de $\pm 2\%$, mostra que, a partir do ponto de operação à $15\,\mathrm{Hz}$, a precisão satisfaz o requisito, atingindo um desvio padrão máximo de $\sigma_{max} = 0{,}41\%$, em $30\,\mathrm{Hz}$. Considerando o pior caso, as medições realizadas nesta faixa de operação terão uma precisão de $\pm 2\sigma_{max} = \pm 0{,}82\%$ em 95% dos casos, conforme Figura 6.

Figura 68 – Desvio padrão relativo das medições realizadas pelo medidor de vazão eletromagnético.

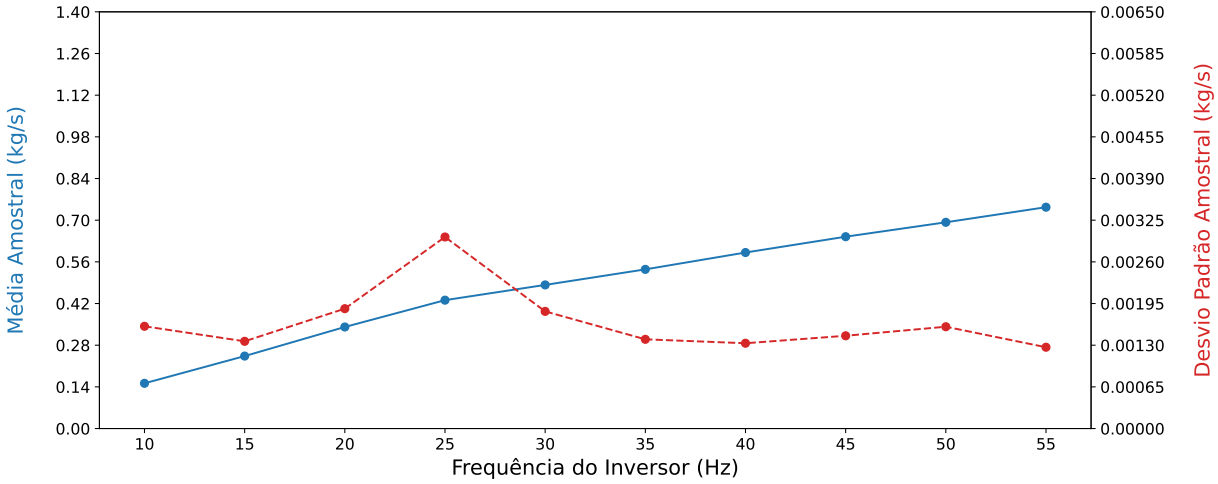


Fonte: autor (2022).

Os resultados obtidos com o medidor de vazão coriolis são apresentados nas Figuras 69 e 70. É possível notar um aumento no desvio padrão em torno do ponto de operação à $25\,\mathrm{Hz}$, juntamente com uma alteração no ganho do sistema, passando de $17{,}37 \times 10^{-3}\,(\mathrm{kg/s})/\mathrm{Hz}$ para $10{,}48 \times 10^{-3}\,(\mathrm{kg/s})/\mathrm{Hz}$. A maior vazão obtida no ramal de água, à $55\,\mathrm{Hz}$, foi de $0{,}7436\,\mathrm{kg/s}$, enquanto no ramal de óleo foi de $1{,}2994\,\mathrm{kg/s}$. A principal diferença entre os dois ramais é a perda de carga causada pela geometria do medidor de vazão coriolis.

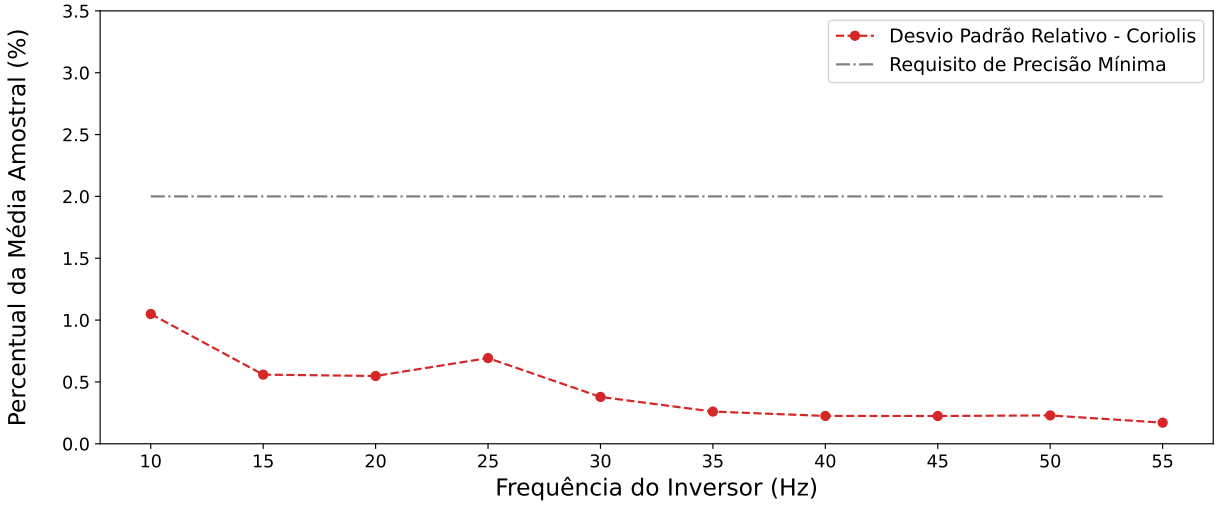
Na Figura 70 é mostrado o desvio padrão percentual em relação à média. Em baixas vazões o coriolis apresenta uma precisão maior que o eletromagnético, atingindo um desvio padrão de $1{,}05\%$ em $10\,\mathrm{Hz}$, enquanto o eletromagnético, $3{,}34\%$. Na faixa de $15\,\mathrm{Hz}$ a $55\,\mathrm{Hz}$, o maior desvio padrão é de $\sigma_{max} = 0{,}69\%$, em $25\,\mathrm{Hz}$. Sendo assim, no pior caso, a precisão em 95% das medições será de $\pm 2\sigma_{max} = \pm 1{,}38\%$, atingindo a especificação do requisito *[RF02]*.

Figura 69 – Média e desvio padrão das medições realizadas pelo medidor de vazão coriolis no ramal de água.



Fonte: autor (2022).

Figura 70 – Desvio padrão relativo das medições realizadas pelo medidor de vazão coriolis.



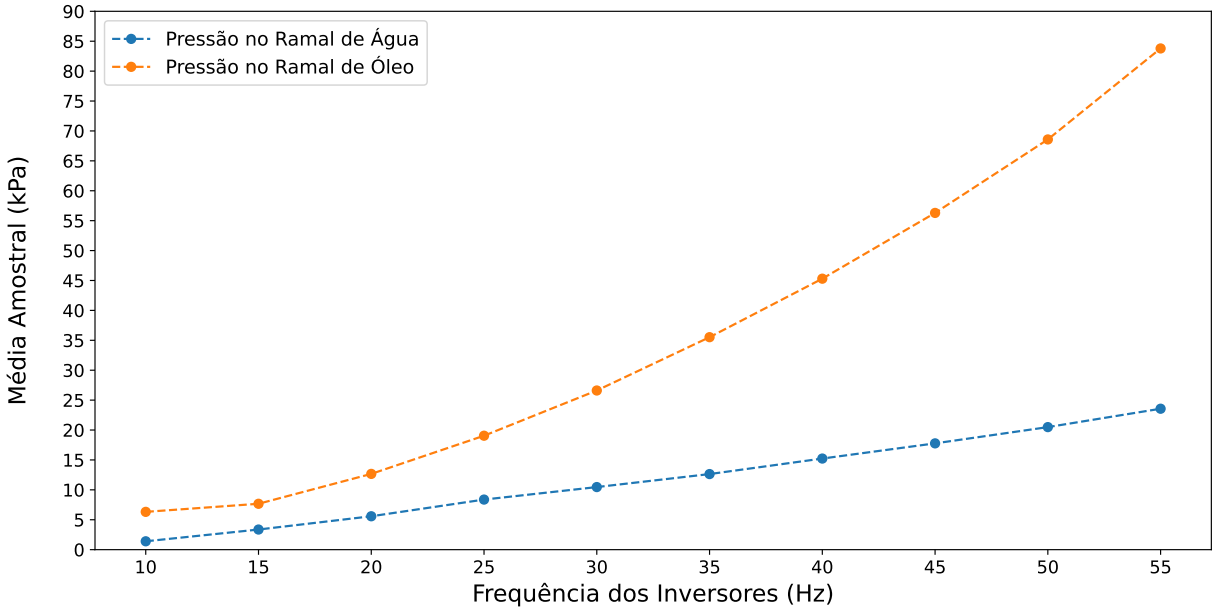
Fonte: autor (2022).

Com isso, é possível confirmar também o cumprimento do requisito *[RF01]*, atingindo vazões de $0{,}1519\,\mathrm{kg/s}$ a $0{,}7436\,\mathrm{kg/s}$ no ramal de água e $0{,}0776\,\mathrm{kg/s}$ a $1{,}2994\,\mathrm{kg/s}$ no ramal de óleo. A diferença entre as vazões máximas obtidas e aquela estimada na Seção 4.1.1, de $1{,}69\,\mathrm{kg/s}$, ocorre devido a fatores não considerados inicialmente, como geometria das tubulações, mangueiras e conexões utilizadas, as quais influenciam na perda de carga do sistema.

O incremento mínimo na vazão pode ser avaliado com relação ao ganho do sistema, sendo $55{,}54 \times 10^{-3}\,(\mathrm{kg/s})/\mathrm{Hz}$ o maior ganho observado. Como a interface com o usuário possibilita incrementos de $0{,}5\,\mathrm{Hz}$ na referência de frequência dos inversores, o incremento mínimo obtido, no pior caso, será de $0{,}028\,\mathrm{kg/s}$, sendo menor que o incremento máximo de $0{,}050\,\mathrm{kg/s}$ estipulado inicialmente. Portanto, o requisito *[RF01]* foi satisfeito em sua totalidade.

Para avaliar o desempenho dos transdutores de pressão, suas medições foram obtidas juntamente com os testes anteriores. Ao passo que a vazão aumenta, também se eleva a pressão no interior da tubulação. Em cada ponto de operação, 1000 amostras foram coletadas, corrigindo o bias e convertendo-as para kPa com a Equação 24. A Figura 71 apresenta a média das medições ao longo da faixa de operação.

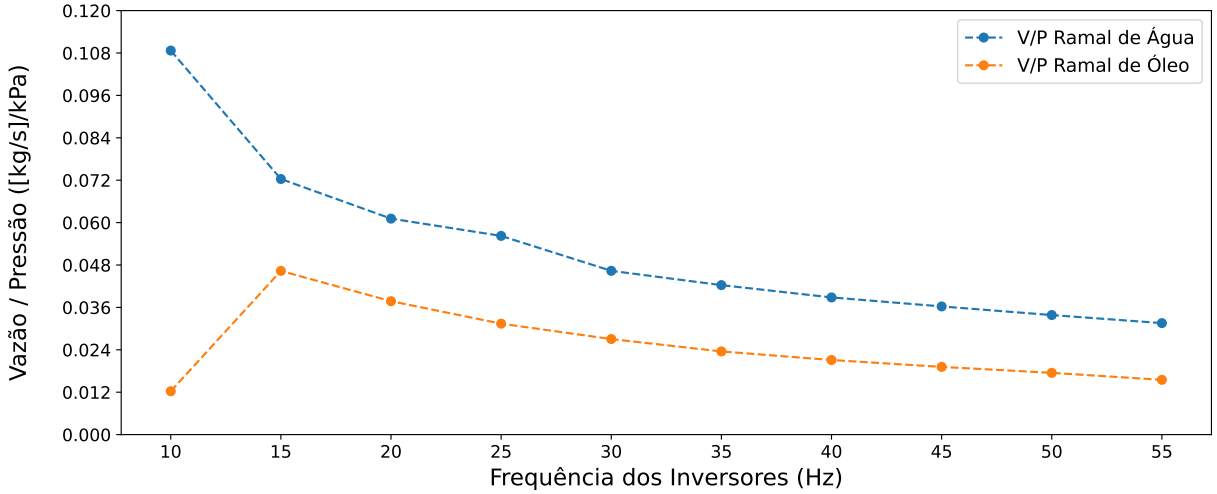
Figura 71 – Média das medições realizadas pelos transdutores de pressão nos ramais de água e óleo.



Fonte: autor (2022).

A diferença entre as pressões no ramal de água e óleo para o mesmo ponto de operação ocorre devido à diferença de vazão entre eles. Analisando a relação entre vazão e pressão para cada ramal, conforme Figura 72, é notado o mesmo comportamento em ambos os ramais a partir de $15\,\mathrm{Hz}$.

Figura 72 – Relação entre vazão e pressão nos ramais de água e óleo.

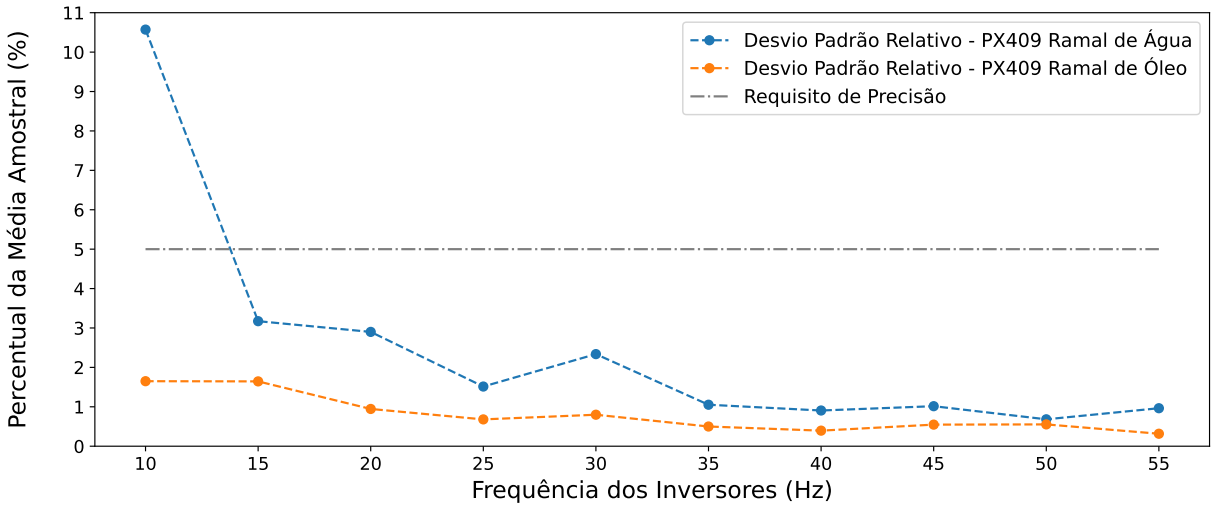


Fonte: autor (2022).

Na faixa de $15\,\mathrm{Hz}$ a $55\,\mathrm{Hz}$ o maior desvio padrão foi de $\pm 3{,}17\%$ no ramal de água e $\pm 1{,}65\%$ no ramal de óleo. Com isso, as menores precisões esperadas para esta faixa de operação são de $\pm 6{,}34\%$ e $\pm 3{,}30\%$ nos ramais de água e óleo, respectivamente. Se aproximando da precisão de 5% definida pelo requisito *[RF03]*.

Devido à baixa pressão no ramal de água aos $10\,\mathrm{Hz}$ a precisão foi afetada, com um desvio padrão de $\pm 10{,}57\%$ nesse ponto. Diferentes padrões de escoamento podem ocorrer no interior das tubulações ao longo da faixa de operação, como estratificado, ondulado e pistonado. A pressão é afetada pelo comportamento de cada padrão, afetando a dispersão das amostras e precisão das medições.

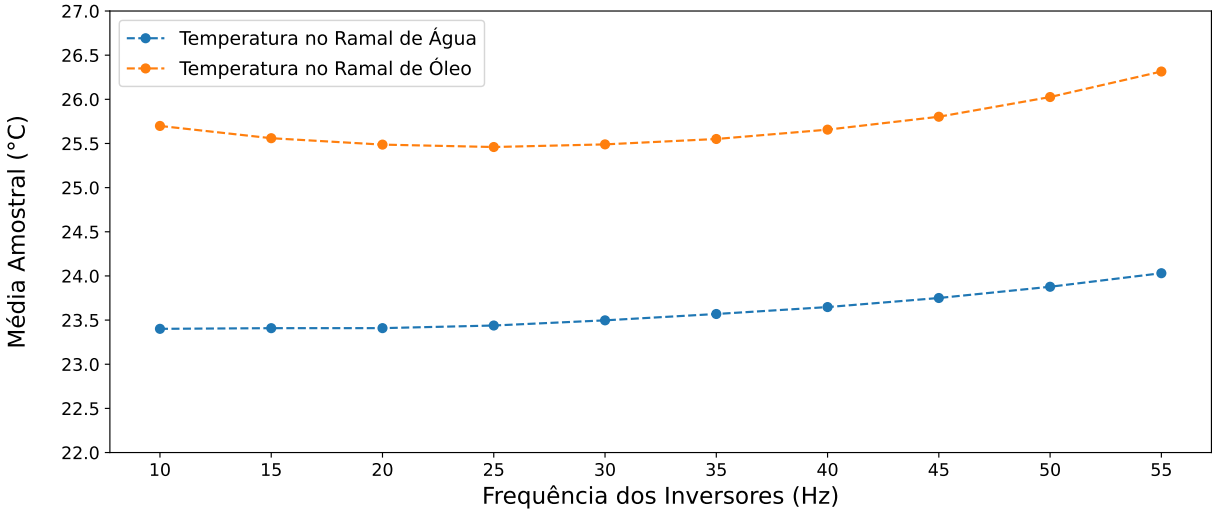
Figura 73 – Desvio padrão relativo das medições realizadas pelos transdutores de pressão.



Fonte: autor (2022).

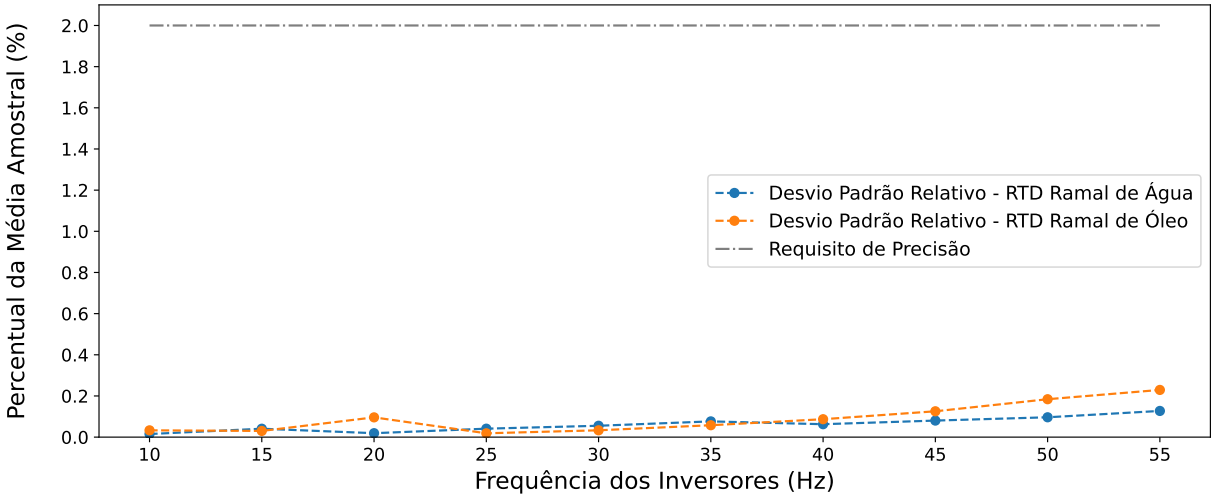
Para as medições de temperatura a mesma análise foi realizada, o resultado é apresentado nas Figuras 74 e 75.

Figura 74 – Média das medições realizadas pelos RTDs nos ramais de água e óleo.



Fonte: autor (2022).

Figura 75 – Desvio padrão relativo das medições realizadas pelos RTDs.



Fonte: autor (2022).

A discrepância de temperatura entre os ramais é devido à variação da temperatura ambiente, uma vez que os testes em cada ramal foram realizados em horários diferentes. A variação da temperatura ao longo dos pontos de operação é causada pelo aquecimento do fluido ao passar pela motobomba. O maior desvio padrão obtido foi de $0{,}23\%$ no ramal de óleo aos $55\,\mathrm{Hz}$ e $26{,}31\,^{\circ}\mathrm{C}$. Assim, atinge-se uma precisão mínima de $0{,}46\%$ em 95% das medições, atendendo ao requisito *[RF04]*.

5.3 FUNCIONAMENTO DA BANCADA

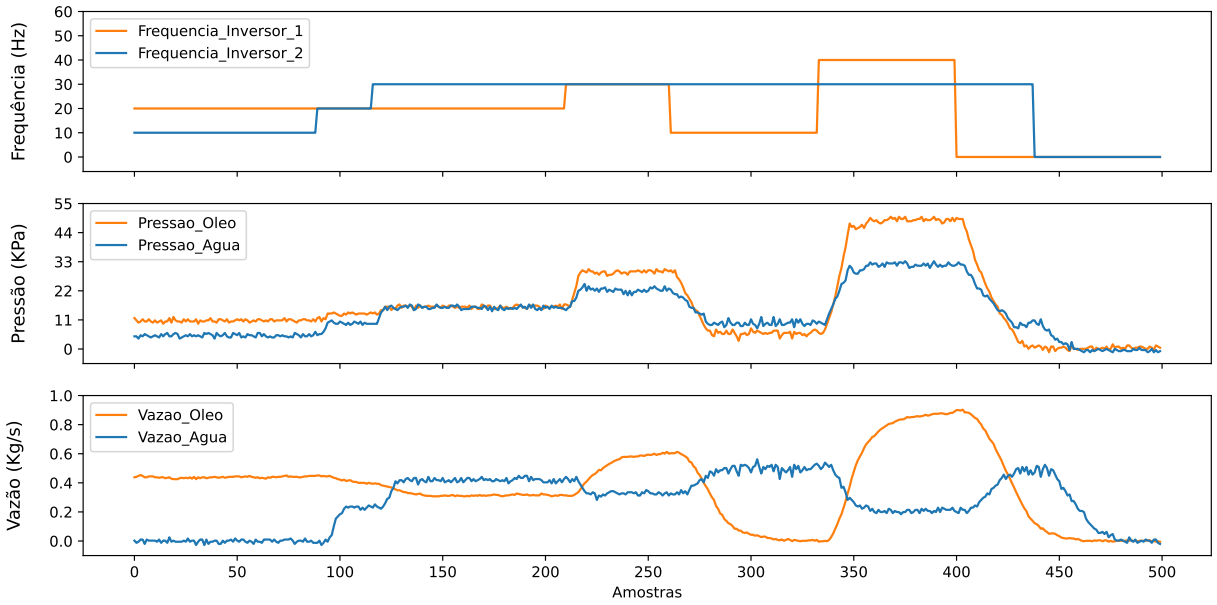
Os testes a seguir foram realizados a fim de demonstrar a interação entre as variáveis e efeitos dos atuadores no comportamento da bancada. Primeiramente, foi avaliada a relação entre as vazões e pressões de cada ramal. Para isso, a válvula de saída foi mantida completamente aberta e as frequências dos inversores foram excursionadas na faixa de $10\,\mathrm{Hz}$ a $40\,\mathrm{Hz}$. A Figura 76 apresenta os resultados obtidos.

Iniciando com frequências de $10\,\mathrm{Hz}$ e $20\,\mathrm{Hz}$ nos ramais de água e óleo, respectivamente, observa-se que a vazão e pressão no ramal de óleo são mais elevadas que no ramal de água, como esperado. Ao elevar sucessivamente a frequência da motobomba no ramal de água para $20\,\mathrm{Hz}$ e $30\,\mathrm{Hz}$, a vazão e pressão desse ramal aumentam, ao passo que a vazão no ramal de óleo diminui. Nota-se também um aumento da pressão no ramal de óleo, que ocorre devido à união dos ecoamentos monofásicos em uma única tubulação.

O mesmo comportamento se repete ao longo do teste, demonstrando a influencia mútua entre as variáveis de pressão e vazão de cada ramal. A cada aumento na frequência de referência de uma das motobombas, a vazão e pressão nesse ramal se intensificam, elevando também a pressão no ramal oposto e diminuindo sua vazão. O oposto também ocorre. Ao diminuir uma das vazões, a pressão em ambos os ramais

decai, e a vazão no ramal oposto aumenta.

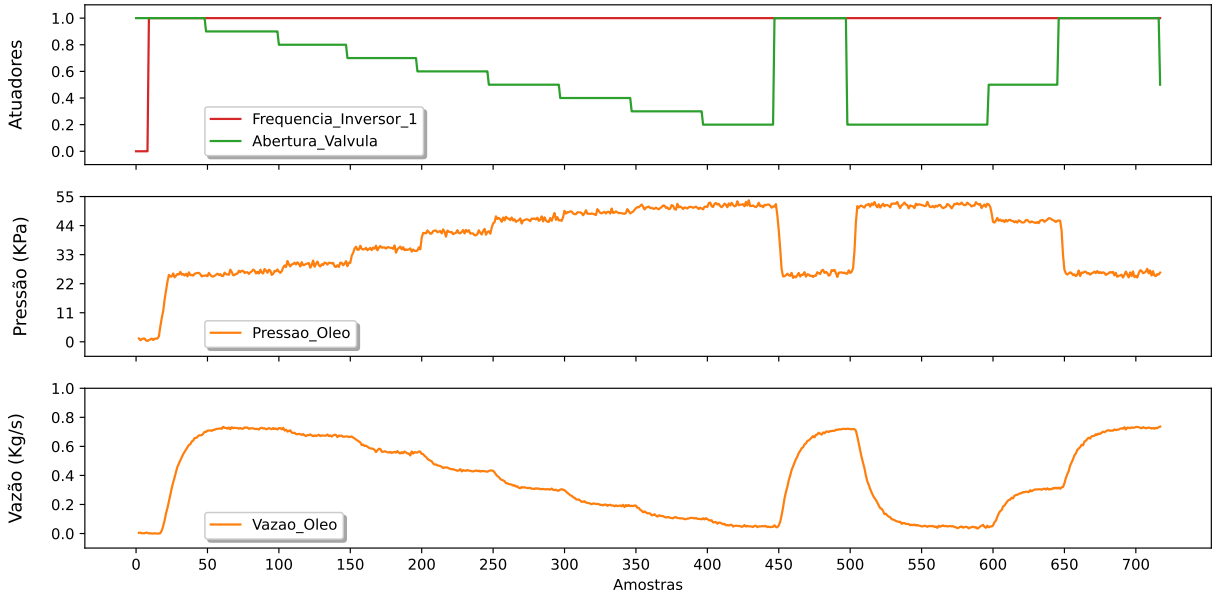
Figura 76 – Influencia mútua entre as vazões e pressões de cada ramal.



Fonte: autor (2022).

Por fim, a influência da válvula de saída foi observada. Utilizando o ramal de óleo, a frequência do inversor foi mantida constante em $30\,\mathrm{Hz}$ e a abertura da válvula foi ajustada de 100% a 20%. Os resultados obtidos são apresentados na Figura 77, onde os valores de referência da válvula e do inversor são normalizados, sendo os valores máximos de 100% e $30\,\mathrm{Hz}$, respectivamente.

Figura 77 – Efeito da variação da válvula de saída.



Fonte: autor (2022).

Como esperado, ao passo que restringe-se a passagem do escoamento, a pressão no interior da tubulação aumenta e a vazão diminui. O nível máximo de pressão,

obtido com a válvula em 20%, foi de $52\,\mathrm{kPa}$. Ao abrir completamente a válvula, a pressão retorna para o patamar inicial, de $25\,\mathrm{kPa}$. A vazão varia entre $0{,}047\,\mathrm{kg/s}$ e $0{,}720\,\mathrm{kg/s}$.

A Tabela 4 apresenta as precisões observadas, considerando o pior caso, em cada instrumento de medição a partir do ponto de operação de $15\,\mathrm{Hz}$.

Tabela 4 – Parâmetros das medições em condições nulas de operação.

Instrumento de medição	**Modelo**	**Precisão**
Medidor de Vazão Eletromagnético	8700M	$\pm 0{,}82\%$
Medidor de Vazão Coriolis	CMF200M	$\pm 1{,}38\%$
Medidor de Pressão (Ramal Água)	PX409-100GI	$\pm 6{,}34\%$
Medidor de Pressão (Ramal Óleo)	PX409-100GI	$\pm 3{,}30\%$
Medidor de Temperatura (Ramal Água)	RTD PT100	$\pm 0{,}26\%$
Medidor de Temperatura (Ramal Óleo)	RTD PT100	$\pm 0{,}46\%$

Fonte: autor (2022).

Sendo assim, os testes realizados apresentam o desempenho da bancada experimental de emulsões em relação aos requisitos estabelecidos, sendo cumpridos total ou parcialmente de acordo com a faixa de operação. O pior caso foi na medição das pressões no ramal de água, onde a precisão nos pontos de operação abaixo de $20\,\mathrm{Hz}$ foi inferior àquela desejada. Contudo, tais incertezas podem estar relacionadas às variações de pressão causadas pelos perfis de escoamento e transições entre eles ao longo das tubulações. O comportamento dinâmico da bancada também pôde ser observado, havendo influência mútua entre as variáveis de pressão e vazão de cada ramal.

6 CONCLUSÕES

Este trabalho teve como objetivo o desenvolvimento de uma bancada experimental que emulasse as condições de operação de instrumentos multifásicos, proporcionando um ambiente de testes onde o pesquisador pudesse controlar e monitorar as principais variáveis do processo de emulsificação entre água e óleo. Para isso, foram levantados os requisitos necessários, construídos os sistemas mecânico e eletrônico, e desenvolvidos os softwares para controle dos atuadores e interface com o usuário.

Iniciando com o levantamento das características necessárias para atender a finalidade da bancada, foram definidos 20 requisitos, divididos entre 13 requisitos funcionais e 7 não-funcionais. Com isso, foi possível estabelecer uma visão geral sobre o sistema, compreendendo as funcionalidades que deveriam existir e os componentes necessários para sua implementação.

A elaboração da bancada iniciou-se com a escolha dos atuadores necessários para gerar as vazões de água e óleo, assim como a válvula para regulação da turbulência após a união dos escoamentos. Com isso, definiu-se o modelo das motobombas e da válvula eletropneumática utilizada na saída. Inversores de frequência também foram empregados, realizando o controle da velocidade das motobombas.

Os instrumentos de medição aplicados para medir a vazão, pressão e temperatura em cada um dos ramais também foram definidos, assim como suas respectivas curvas de calibração. A modelagem do sistema mecânico foi então realizada, contemplando o posicionamento dos atuadores, instrumentos de medição e tanques de armazenamento.

O hardware foi modelado a fim de possibilitar o controle dos atuadores remotamente através de um computador. Para isso, o microcontrolador TIVA-C foi utilizado, recebendo os comandos do LabView via comunicação UART pela porta USB. Os comandos são tratados e distribuídos para cada módulo adicional, responsáveis pela comunicação com os atuadores. Para os inversores de frequência, os comandos são enviados via protocolo Modbus, passando por um conversor TTL-RS485. Já para o posicionador da válvula eletropneumática, foi utilizado um conversor de $0\,\mathrm{V}$ a $5\,\mathrm{V}$ para $4\,\mathrm{mA}$ a $20\,\mathrm{mA}$ juntamente com um DAC, enviando os sinais pela malha de corrente.

O firmware embarcado no microcontrolador foi projetado para atender a comunicação com cada dispositivo. O tratamento das mensagens e conversão dos comandos para a devida referência é realizado internamente em cada classe, modularizando o código e facilitando adaptações. Os protocolos de comunicação UART e I2C são disponibilizados pela biblioteca driverlib, juntamente demais funções para

controle dos periféricos do microcontrolador. As mensagens do protocolo Modbus e o CRC são montados internamente, de acordo com o dispositivo de destino e comando a ser enviado.

No software em LabView, que faz a interface com o usuário, o painel principal disponibiliza as funções para comando dos atuadores, escolha do arquivo de texto a ser gerado e seleção da porta USB na qual está conectado o microcontrolador. Os gráficos das medições são apresentados em abas de acordo com a grandeza física medida, conforme previsto.

Por fim, a bancada cumpre o propósito previsto inicialmente, disponibilizando as funcionalidades necessárias para gerar escoamentos multifásicos e emulsões entre água e óleo crude, medindo e armazenado os dados de vazão, pressão e temperatura.

6.1 TRABALHOS FUTUROS

Comandos para ler o status dos inversores e possíveis mensagens de erros internos podem ser implementados, informando ao usuário o diagnóstico desses dispositivos. Configurações de valores de segurança, como pressão e vazão máxima permitidas, também podem ser adicionados na interface com o usuário. A correção do bias, que foi realizada no tratamento e análise dos dados, pode ser incluída diretamente na conversão das leituras no LabView.

Atualmente, o controle da vazão é realizado em malha aberta, ajustando a frequência dos inversores. A estimação das plantas e projeto de um controlador para cada ramal, tendo em vista que possuem características dinâmicas distintas, melhoraria o desempenho do sistema, atingindo e mantendo a referência de vazão escolhida, apesar das perturbações. O controle em malha fechada também iria minimizar os efeitos apresentados na Figura 76, mitigando a alteração da vazão de um ramal em função da vazão no ramal oposto.

Para um melhor ajuste na abertura da válvula, um sensor de distância pode ser posicionado em sua haste, possibilitando o controle em malha fechada. A estrutura da bancada, como cabeamento e conexões com o hardware, permitem tal melhoria.

REFERÊNCIAS

AGUIRRE, L. **Fundamentos de Instrumentação**. São Paulo: Pearson Education do Brasil, 2013.

ANALOG DEVICES. **UART**: a hardware communication protocol understanding universal asynchronous receiver/transmitter. Wilmington, Delaware, 2020.

ANTON PAAR. **Viscosity of Crude Oil**: Description and classification overview. Graz, Áustria, 2021.

BAKER, R. **Flow measurement handbook**: industrial designs, operating principles, performance, and applications. New York: Cambridge University Press, 2016.

BERMO - ARMATUREN GROUP. **Válvula ARI-STEVI**: Pro 470 ansi. Blumenau, Santa Catarina, 2019.

BRENNEN, C. E. **Fundamentals of multiphase flow**. Pasadena: Cambridge University Press, 2005.

CROWDER, J. A.; HOFF, C. W. **Requirements engineering**: laying a firm foundation. Larkspur: Springer, 2022.

EMERSON ELECTRIC CO. **Plataforma do medidor de vazão magnético Rosemount 8700M**: Ficha de dados do produto. St. Louis, Missouri, 2019.

EMERSON ELECTRIC CO. **Medidores de vazão e densidade Coriolis ELITE Micro Motion**: Ficha de dados do produto. St. Louis, Missouri, 2021.

FAGHRI, A.; ZHANG, Y. **Fundamentals of multiphase heat transfer and flow**. Columbia: Springer, 2020.

FALCONE, G. et al. **Multiphase flow metering**: principles and applications. Amsterdam: Elsevier, 2009.

FINGAS, M. Water-in-oil emulsions: Formation and prediction. **Journal of Petroleum Science Research**, v. 3, p. 38–51, jan. 2014.

FOROUZAN, B. **Comunicação de Dados e Redes de Computadores**. Porto Alegre: AMGH Editora Ltda, 2009.

FRANKLIN ELECTRIC INDÚSTRIA DE MOTOBOMBAS S.A. **Centrífugas Monoestágio BC-92**: tabela de seleção. Joinville, 2022.

FRENZEL, L. **Handbook of Serial Communications Interfaces**: a comprehensive compendium of serial digital input/output. Waltham: Elsevier Science, 2016.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE Std 1233**: Guide for developing system requirements specifications. Nova York, 1996.

INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. **IEEE 830**: Recommended practice for software requirements specifications. Nova York, 1998.

INTERNATIONAL ORGANIZATION FOR STANDARDIZATION. **ISO/IEC/IEEE29148**: Systems and software engineering - life cycle processes - requirements engineering. Vernier, 2018.

JY VáLVULAS. **Válvula de controle**. Sorocaba, São Paulo, 2020.

KHAN, B. et al. Basics of pharmaceutical emulsions: A review. **African Journal of Pharmacy and Pharmacology**, v. 5, p. 25–36, dez. 2011.

KUROSE, J.; ROSS, K. **Redes de Computadores e a Internet**: uma abordagem top-down. São Paulo: Pearson Education do Brasil, 2013.

MICROCHIP TECHNOLOGY. **MCP4725**: 12-bit digital-to-analog converter with eeprom memory. Chandler, Arizona, 2022.

MODBUS ORGANIZATION. **Modicon Modbus Protocol Reference Guide**: Pi–mbus–300 rev. j. Andover, Massachusetts, 2006.

MODBUS ORGANIZATION. **Modbus Application Protocol Specification**: V1.1b3. Andover, Massachusetts, 2012.

MURATA POWER SOLUTIONS. **DMS APPLICATION NOTE**: 4-20ma current loop primer. Mansfield, Massachusetts, 2015.

NATIONAL INSTRUMENTS. **Design Validation Test System Build Guide for CompactDAQ**. Austin, Texas, 2021.

NATIONAL INSTRUMENTS. **Fundamentals, System Design, and Setup for the 4 to 20mA Current Loop**. Austin, Texas, 2022.

NILSSON, J.; RIEDEL, S. **Circuitos Elétricos**. São Paulo: Pearson Education do Brasil, 2016.

NORTHROP, R. **Introduction to Instrumentation and Measurements**. Boca Raton: Taylor & Francis, 2014.

NXP SEMICONDUCTORS. **I2C-bus specification and user manual**: Rev. 7.0. Eindhoven, Netherlands, 2021.

OMEGA ENGINEERING. **Sensores RTD para Imersão Ultraprecisos**: Série pm. Campinas, São Paulo, 2013.

OMEGA ENGINEERING. **Transdutores de pressão de alta exatidão**: Série px409. Campinas, São Paulo, 2015.

PDACONTROL. **Test module 0-10V to 4-20mA**. Jefferson City, Missouri, 2018.

PELGROM, M. **Analog-to-Digital Conversion**. Stiphout: Springer, 2021.

TEXAS INSTRUMENTS. **Low-Power, Slew-Rate-Limited RS-485/RS-422 Transceivers**. San José, Califórnia, 2014.

TEXAS INSTRUMENTS. **Tiva TM4C123GH6PM Microcontroller**. Austin, Texas, 2014.

VAZQUEZ, C.; SIMÕES, G. **Engenharia de Requisitos**: software orientado ao negócio. São Paulo: Brasport, 2016.

WAHAIBI, T. et al. Basics of pharmaceutical emulsions: A review. **Journal of Petroleum Science and Engineering**, v. 57, p. 334–346, jan. 2007.

WEBSTER, J.; EREN, H. **Measurement, Instrumentation, and Sensors Handbook**: spatial, mechanical, thermal, and radiation measurement. Boca Raton: CRC Press, 2018.

WEG AUTOMAçãO. **Inversor de Frequência CFW300 V3.1X**: Manual de programação. Jaraguá do Sul, Santa Catarina, 2021.

WONG, S.; LIM, J.; DOL, S. Crude oil emulsion: A review on formation, classification and stability of water-in-oil emulsions. **Journal of Petroleum Science and Engineering**, v. 135, p. 498–504, jan. 2015.

APÊNDICE A - IMPLEMENTAÇÕES DO FIRMWARE EM C++

main.cpp1

```
#include "board_configs.h"
#include "user_configs.h"
#include "Communication_Interfaces.hpp"
#include "Communication_Handlers.hpp"
#include "Devices.hpp"

int main(void) {
    MCP4725_DAC                 *MCP4725;
    CFW300_FrequencyInverter    *CFW300_1;
    CFW300_FrequencyInverter    *CFW300_2;
    LabView_ControlComputer     *LabViewControl;

    MCP4725  = new MCP4725_DAC(MCP4725_ADDR0);
    CFW300_1 = new CFW300_FrequencyInverter(ADDR_CFW300_1);
    CFW300_2 = new CFW300_FrequencyInverter(ADDR_CFW300_2);

    LabViewControl = new LabView_ControlComputer(MCP4725,
                                                 CFW300_1,
                                                 CFW300_2);

    LabViewControl->Run();

    delete MCP4725;
    delete CFW300_1;
    delete CFW300_2;
    delete LabViewControl;

    return 0;
}
```

board_config.h

```
#ifndef BOARD_CONFIGS_H_
#define BOARD_CONFIGS_H_

#include <stdint.h>
#include <stdbool.h>
#include <stdarg.h>
#include "inc/hw_ints.h"
```

```
8  #include "inc/hw_memmap.h"
9  #include "inc/hw_types.h"
10 #include "inc/hw_i2c.h"
11 #include "inc/hw_gpio.h"
12 #include "driverlib/gpio.h"
13 #include "driverlib/interrupt.h"
14 #include "driverlib/pin_map.h"
15 #include "driverlib/sysctl.h"
16 #include "driverlib/uart.h"
17 #include "driverlib/i2c.h"
18
19 // TYPES USED TO CONFIGURE PERIPHERALS
20
21 //UART PERIPHERAL CONFIG
22 typedef struct UART_PERIPHERAL_CFG {
23     uint32_t SYSCTL_PERIPH_UART;
24     uint32_t SYSCTL_PERIPH_GPIO;
25     uint32_t GPIO_PORT_BASE;
26     uint32_t UART_BASE;
27     uint32_t INT_UART;
28     uint32_t GPIO_P_URX;
29     uint32_t GPIO_P_UTX;
30     uint32_t GPIO_PIN_x;
31     uint32_t GPIO_PIN_y;
32 } UART_PERIPHERAL_CFG;
33
34 //UART USER CONFIG
35 typedef struct UART_USER_CFG {
36     uint32_t BAUD_RATE;
37     uint32_t UART_CONFIG_WLEN;
38     uint32_t UART_CONFIG_STOP;
39     uint32_t UART_CONFIG_PAR;
40 } UART_USER_CFG;
41
42 //UART CONFIG
43 typedef struct UART_CFG {
44     UART_PERIPHERAL_CFG PERIPHERAL_CFG;
45     UART_USER_CFG USER_CFG;
46 } UART_CFG;
47
48 //I2C PERIPHERAL CONFIG
49 typedef struct I2C_PERIPHERAL_CFG {
50     uint32_t SYSCTL_PERIPH_I2C;
51     uint32_t SYSCTL_PERIPH_GPIO;
52     uint32_t GPIO_PORT_BASE;
53     uint32_t I2C_BASE;
54     uint32_t GPIO_P_I2CSCL;
```

```
55      uint32_t GPIO_P_I2CSDA;
56      uint32_t GPIO_PIN_SCL;
57      uint32_t GPIO_PIN_SDA;
58  } I2C_PERIPHERAL_CFG;
59
60  //I2C USER CONFIG
61  typedef struct I2C_USER_CFG {
62      bool BAUD_RATE_HIGH_SPEED; //True: 400kbps, False: 100kbps
63  } I2C_USER_CFG;
64
65  //I2C CONFIG
66  typedef struct I2C_CFG {
67      I2C_PERIPHERAL_CFG PERIPHERAL_CFG;
68      I2C_USER_CFG USER_CFG;
69  } I2C_CFG;
70
71  //PIN STRUCT TO ENABLE TX IN RS485 MODULE
72  typedef struct PIN {
73      uint32_t GPIO_PORT_BASE;
74      uint32_t GPIO_PIN;
75  } PIN;
76
77  // STANDARD PERIPHERAL CONFIGS TO TIVA-C (dont modify)
78  //Copy this values into CFG structs with USER_CFG,
79  //the structures below are not referenced.
80
81  const UART_PERIPHERAL_CFG UART0_CFG = {
82      .SYSCTL_PERIPH_UART     = SYSCTL_PERIPH_UART0,
83      .SYSCTL_PERIPH_GPIO     = SYSCTL_PERIPH_GPIOA,
84      .GPIO_PORT_BASE         = GPIO_PORTA_BASE,
85      .UART_BASE              = UART0_BASE,
86      .INT_UART               = INT_UART0,
87      .GPIO_P_URX             = GPIO_PA0_U0RX,
88      .GPIO_P_UTX             = GPIO_PA1_U0TX,
89      .GPIO_PIN_x             = GPIO_PIN_0,
90      .GPIO_PIN_y             = GPIO_PIN_1,
91  };
92
93  const UART_PERIPHERAL_CFG UART1_CFG = {
94      .SYSCTL_PERIPH_UART     = SYSCTL_PERIPH_UART1,
95      .SYSCTL_PERIPH_GPIO     = SYSCTL_PERIPH_GPIOB,
96      .GPIO_PORT_BASE         = GPIO_PORTB_BASE,
97      .UART_BASE              = UART1_BASE,
98      .INT_UART               = INT_UART1,
99      .GPIO_P_URX             = GPIO_PB0_U1RX,
100     .GPIO_P_UTX             = GPIO_PB1_U1TX,
101     .GPIO_PIN_x             = GPIO_PIN_0,
```

```
102     .GPIO_PIN_y             = GPIO_PIN_1,
103 };
104
105 const I2C_PERIPHERAL_CFG I2C0_CFG = {
106     .SYSCTL_PERIPH_I2C      = SYSCTL_PERIPH_I2C0,
107     .SYSCTL_PERIPH_GPIO     = SYSCTL_PERIPH_GPIOB,
108     .GPIO_PORT_BASE         = GPIO_PORTB_BASE,
109     .I2C_BASE               = I2C0_BASE,
110     .GPIO_P_I2CSCL          = GPIO_PB2_I2C0SCL,
111     .GPIO_P_I2CSDA          = GPIO_PB3_I2C0SDA,
112     .GPIO_PIN_SCL           = GPIO_PIN_2,
113     .GPIO_PIN_SDA           = GPIO_PIN_3,
114 };
115
116 #endif /* BOARD_CONFIGS_H_ */
```

user_config.h

```
1  #ifndef USER_CONFIGS_H_
2  #define USER_CONFIGS_H_
3
4  #include <stdint.h>
5  #include <stdbool.h>
6  #include <stdarg.h>
7  #include "inc/hw_ints.h"
8  #include "inc/hw_memmap.h"
9  #include "inc/hw_types.h"
10 #include "inc/hw_i2c.h"
11 #include "inc/hw_gpio.h"
12 #include "driverlib/gpio.h"
13 #include "driverlib/interrupt.h"
14 #include "driverlib/pin_map.h"
15 #include "driverlib/sysctl.h"
16 #include "driverlib/uart.h"
17 #include "driverlib/i2c.h"
18
19 #include "board_configs.h"
20
21 //Copy values from xxx_PERIPHERAL_CFG in board_configs.h
22 //to initialize xxx_CFG struct
23
24 //CFW300 UART ------------------------------------------------------
25 const UART_CFG UART_CFW300 = {
26     //Board parameters to config UART_CFW300 communication
27     .PERIPHERAL_CFG = {
28         .SYSCTL_PERIPH_UART     = SYSCTL_PERIPH_UART1,
```

```
29          .SYSCTL_PERIPH_GPIO     = SYSCTL_PERIPH_GPIOB,
30          .GPIO_PORT_BASE         = GPIO_PORTB_BASE,
31          .UART_BASE              = UART1_BASE,
32          .INT_UART               = INT_UART1,
33          .GPIO_P_URX             = GPIO_PB0_U1RX,
34          .GPIO_P_UTX             = GPIO_PB1_U1TX,
35          .GPIO_PIN_x             = GPIO_PIN_0,
36          .GPIO_PIN_y             = GPIO_PIN_1,
37      },
38      //User parameters to config UART_CFW300 communication
39      .USER_CFG = {
40          .BAUD_RATE              = 76800,
41          .UART_CONFIG_WLEN       = UART_CONFIG_WLEN_8,
42          .UART_CONFIG_STOP       = UART_CONFIG_STOP_ONE,
43          .UART_CONFIG_PAR        = UART_CONFIG_PAR_EVEN,
44      },
45  };
46
47  const PIN ENABLE_TX_PB4 = {
48      .GPIO_PORT_BASE             = GPIO_PORTB_BASE,
49      .GPIO_PIN                   = GPIO_PIN_4,
50  };
51
52  #define ADDR_CFW300_1  0x01
53  #define ADDR_CFW300_2  0x02
54
55  //Delay at end of telegram
56  //(Datasheet Modbus RTU CFW300 pg17: 76800kbps 2,005ms)
57  #define MODBUS_END_FRAME_US    2100    //[microseconds]
58  //-------------------------------------------------------------
59
60  //MCP4725 I2C ---------------------------------------------------
61  const I2C_CFG I2C_MCP4725 = {
62      //Board parameters to config I2C_MCP4725 communication
63      .PERIPHERAL_CFG = {
64          .SYSCTL_PERIPH_I2C      = SYSCTL_PERIPH_I2C0,
65          .SYSCTL_PERIPH_GPIO     = SYSCTL_PERIPH_GPIOB,
66          .GPIO_PORT_BASE         = GPIO_PORTB_BASE,
67          .I2C_BASE               = I2C0_BASE,
68          .GPIO_P_I2CSCL          = GPIO_PB2_I2C0SCL,
69          .GPIO_P_I2CSDA          = GPIO_PB3_I2C0SDA,
70          .GPIO_PIN_SCL           = GPIO_PIN_2,
71          .GPIO_PIN_SDA           = GPIO_PIN_3,
72      },
73      //User parameters to config I2C_MCP4725 communication
74      .USER_CFG = {
75          .BAUD_RATE_HIGH_SPEED   = true,
```

```
76      },
77  };
78
79  //Tiva already put the last bit in I2C:
80  //FirstByte = (ADDR << 1) | 0b(RW))
81  #define MCP4725_ADDR0    0x60 //|1|1|0|0|0|0|0|
82  #define MCP4725_ADDR1    0x62 //|1|1|0|0|0|1|0|
83  //---------------------------------------------------------------------
84
85
86  //LABVIEW UART ---------------------------------------------------------
87  const UART_CFG UART_LABVIEW = {
88      //Board parameters to config UART_LABVIEW
89      .PERIPHERAL_CFG = {
90          .SYSCTL_PERIPH_UART     = SYSCTL_PERIPH_UART0,
91          .SYSCTL_PERIPH_GPIO     = SYSCTL_PERIPH_GPIOA,
92          .GPIO_PORT_BASE         = GPIO_PORTA_BASE,
93          .UART_BASE              = UART0_BASE,
94          .INT_UART               = INT_UART0,
95          .GPIO_P_URX             = GPIO_PA0_U0RX,
96          .GPIO_P_UTX             = GPIO_PA1_U0TX,
97          .GPIO_PIN_x             = GPIO_PIN_0,
98          .GPIO_PIN_y             = GPIO_PIN_1,
99      },
100     //User parameters to config UART_LABVIEW
101     .USER_CFG = {
102         .BAUD_RATE               = 9600,
103         .UART_CONFIG_WLEN        = UART_CONFIG_WLEN_8,
104         .UART_CONFIG_STOP        = UART_CONFIG_STOP_ONE,
105         .UART_CONFIG_PAR         = UART_CONFIG_PAR_NONE,
106     },
107 };
108
109 #define Labview_nCommands         4 //Number of valid commands
110 #define Labview_dataframeSize     4 //Number of bytes received from LabView
111
112 static const char ValidCommands[] = {'A', 'D', 'L', 'V'};
113 //---------------------------------------------------------------------
114
115 #endif /* USER_CONFIGS_H_ */
```

Communication_Interfaces.hpp

```
1  #ifndef COMMUNICATION_INTERFACES_HPP_
2  #define COMMUNICATION_INTERFACES_HPP_
3
```

```
4  /*C++ Section*/
5  #ifdef __cplusplus
6
7  #include <stdint.h>
8  #include <stdbool.h>
9  #include <stdarg.h>
10 #include "inc/hw_ints.h"
11 #include "inc/hw_memmap.h"
12 #include "inc/hw_types.h"
13 #include "inc/hw_i2c.h"
14 #include "inc/hw_gpio.h"
15 #include "driverlib/gpio.h"
16 #include "driverlib/interrupt.h"
17 #include "driverlib/pin_map.h"
18 #include "driverlib/sysctl.h"
19 #include "driverlib/uart.h"
20 #include "driverlib/i2c.h"
21
22 #include "board_configs.h"
23
24 //Abstract base class
25 class Communication_Interface {
26 public:
27     virtual void Write(uint8_t *, int) = 0;
28     virtual void Read(uint8_t *, int) = 0;
29 };
30
31 //Modbus interface
32 class MODBUS_RTU : public Communication_Interface {
33 public:
34     MODBUS_RTU(UART_CFG myUART, PIN myPIN);
35     void Write(uint8_t *data, int len) override;
36     void Read(uint8_t *data, int len) override;
37 private:
38     UART_CFG    _UART;
39     PIN         _ENABLE_TX_PIN;
40 };
41
42 //I2C interface
43 class I2C : public Communication_Interface {
44 public:
45     I2C(I2C_CFG myI2C);
46     void Write(uint8_t *data, int len) override;
47     void Read(uint8_t *data, int len) override;
48 private:
49     I2C_CFG _I2C;
50 };
```

```
51
52 //UART interface
53 class UART : public Communication_Interface {
54 public:
55     UART(UART_CFG myUART);
56     void Write(uint8_t *data, int len) override;
57     void Read(uint8_t *data, int pos) override;
58 private:
59     UART_CFG _UART;
60 };
61
62 #endif /*C++ Section*/
63 #endif /* COMMUNICATION_INTERFACES_HPP_ */
```

Communication_Interfaces.cpp

```
1  #include "Communication_Interfaces.hpp"
2
3  void EnableUART(UART_CFG UARTx, bool enable_interrupt = false) {
4      //Enable peripheral UARTx
5      SysCtlPeripheralEnable(UARTx.PERIPHERAL_CFG.SYSCTL_PERIPH_UART);
6      //Enable peripheral GPIOx with the pins of UARTx
7      SysCtlPeripheralEnable(UARTx.PERIPHERAL_CFG.SYSCTL_PERIPH_GPIO);
8      //Configure Pin as RX of UARTx
9      GPIOPinConfigure(UARTx.PERIPHERAL_CFG.GPIO_P_URX);
10     //Configure Pin as TX of UARTx
11     GPIOPinConfigure(UARTx.PERIPHERAL_CFG.GPIO_P_UTX);
12     //Configure Pins as RX and TX of UARTx
13     GPIOPinTypeUART(UARTx.PERIPHERAL_CFG.GPIO_PORT_BASE,
14                     UARTx.PERIPHERAL_CFG.GPIO_PIN_x |
15                     UARTx.PERIPHERAL_CFG.GPIO_PIN_y);
16
17     //User configs
18     UARTConfigSetExpClk(UARTx.PERIPHERAL_CFG.UART_BASE,
19                         SysCtlClockGet(),
20                         UARTx.USER_CFG.BAUD_RATE,
21                         (UARTx.USER_CFG.UART_CONFIG_WLEN    |
22                         UARTx.USER_CFG.UART_CONFIG_STOP     |
23                         UARTx.USER_CFG.UART_CONFIG_PAR      ));
24
25     if(enable_interrupt) {
26         IntMasterEnable(); //Enable processor interrupts
27         IntEnable(UARTx.PERIPHERAL_CFG.INT_UART); //Enable UART interrupt
28         //Enable RX and TX interrupts
29         UARTIntEnable(UARTx.PERIPHERAL_CFG.UART_BASE,
30                       UART_INT_RX | UART_INT_RT);
```

```
31      }
32  }
33
34  void EnableI2C(I2C_CFG I2Cx) {
35      //Enable I2C module
36      SysCtlPeripheralEnable(I2Cx.PERIPHERAL_CFG.SYSCTL_PERIPH_I2C);
37      //Reset module
38      SysCtlPeripheralReset(I2Cx.PERIPHERAL_CFG.SYSCTL_PERIPH_I2C);
39      //Enable GPIO peripheral that contains I2Cx
40      SysCtlPeripheralEnable(I2Cx.PERIPHERAL_CFG.SYSCTL_PERIPH_GPIO);
41      //Configure the pin muxing for I2Cx functions on ports.
42      GPIOPinConfigure(I2Cx.PERIPHERAL_CFG.GPIO_P_I2CSCL);
43      GPIOPinConfigure(I2Cx.PERIPHERAL_CFG.GPIO_P_I2CSDA);
44      //Select the I2C function for these pins.
45      GPIOPinTypeI2CSCL(I2Cx.PERIPHERAL_CFG.GPIO_PORT_BASE,
46                        I2Cx.PERIPHERAL_CFG.GPIO_PIN_SCL);
47      GPIOPinTypeI2C(I2Cx.PERIPHERAL_CFG.GPIO_PORT_BASE,
48                        I2Cx.PERIPHERAL_CFG.GPIO_PIN_SDA);
49
50      //Enable and initialize the I2Cx master module.
51      I2CMasterInitExpClk(I2Cx.PERIPHERAL_CFG.I2C_BASE,
52                          SysCtlClockGet(),
53                          I2Cx.USER_CFG.BAUD_RATE_HIGH_SPEED);
54      //Clear I2Cx FIFOs
55      HWREG(I2Cx.PERIPHERAL_CFG.I2C_BASE + I2C_O_FIFOCTL) = 80008000;
56  }
57
58  //MODBUS_RTU ------------------------------------------------------------
59  MODBUS_RTU::MODBUS_RTU(UART_CFG myUART, PIN myPIN) {
60          _UART           = myUART;
61          _ENABLE_TX_PIN  = myPIN;
62
63          EnableUART(_UART);
64
65          //Enable GPIO pin to output
66          GPIOPinTypeGPIOOutput(_ENABLE_TX_PIN.GPIO_PORT_BASE,
67                                _ENABLE_TX_PIN.GPIO_PIN);
68  }
69
70  void MODBUS_RTU::Write(uint8_t *data, int len) {
71      //HIGH to enable transmission
72      GPIOPinWrite(_ENABLE_TX_PIN.GPIO_PORT_BASE,
73                   _ENABLE_TX_PIN.GPIO_PIN, (uint8_t)16);
74
75      int i = 0;
76      for(i=0; i < len; i++){
77          UARTCharPut(_UART.PERIPHERAL_CFG.UART_BASE, data[i]);
```

```
78      }
79  }
80
81  void MODBUS_RTU::Read(uint8_t *data, int len) {
82      int i = 0;
83      for(i=0; i < len; i++){
84          data[i] = UARTCharGet(_UART.PERIPHERAL_CFG.UART_BASE);
85      }
86  }
87  //---------------------------------------------------------------------
88
89  //I2C -----------------------------------------------------------------
90  I2C::I2C(I2C_CFG myI2C){
91      _I2C = myI2C;
92
93      EnableI2C(_I2C);
94  }
95
96  void I2C::Write(uint8_t *data, int len) {
97      //Tell the master module what address it will place on the bus
98      I2CMasterSlaveAddrSet(_I2C.PERIPHERAL_CFG.I2C_BASE, data[0], false);
99      //Put first data to be sent into FIFO
100     I2CMasterDataPut(_I2C.PERIPHERAL_CFG.I2C_BASE, data[1]);
101
102     //If there is only one byte of data, use the single send I2C function
103     if(len == 2) {
104         //Initiate send of data from the MCU
105         I2CMasterControl(_I2C.PERIPHERAL_CFG.I2C_BASE,
106                          I2C_MASTER_CMD_SINGLE_SEND);
107         //Wait until MCU is done transferring
108         while(I2CMasterBusy(_I2C.PERIPHERAL_CFG.I2C_BASE));
109     }
110
111     //Otherwise, start transmission of multiple bytes on the I2C bus
112     else {
113         //Initiate send of data from the MCU
114         I2CMasterControl(_I2C.PERIPHERAL_CFG.I2C_BASE,
115                          I2C_MASTER_CMD_BURST_SEND_START);
116         // Wait until MCU is done transferring
117         while(I2CMasterBusy(_I2C.PERIPHERAL_CFG.I2C_BASE));
118
119         int i = 2;
120
121         //Send num_of_args-2 bytes, using the BURST_SEND_CONT command
122         while(i != len-1) {
123             //Put next byte of data into I2C FIFO
124             I2CMasterDataPut(_I2C.PERIPHERAL_CFG.I2C_BASE, data[i++]);
```

```
125             //Send next data that was just placed into FIFO
126             I2CMasterControl(_I2C.PERIPHERAL_CFG.I2C_BASE,
127                              I2C_MASTER_CMD_BURST_SEND_CONT);
128             //Wait until MCU is done transferring
129             while(I2CMasterBusy(_I2C.PERIPHERAL_CFG.I2C_BASE));
130         }
131 
132         //Put last byte of data into I2C FIFO
133         I2CMasterDataPut(_I2C.PERIPHERAL_CFG.I2C_BASE, data[i]);
134         //send next data that was just placed into FIFO
135         I2CMasterControl(_I2C.PERIPHERAL_CFG.I2C_BASE,
136                          I2C_MASTER_CMD_BURST_SEND_FINISH);
137         //Wait until MCU is done transferring
138         while(I2CMasterBusy(_I2C.PERIPHERAL_CFG.I2C_BASE));
139     }
140 }
141 
142 void I2C::Read(uint8_t *data, int len) {}
143 //---------------------------------------------------------------------
144 
145 
146 //UART ----------------------------------------------------------------
147 UART::UART(UART_CFG myUART) {
148     _UART = myUART;
149     EnableUART(_UART, true);
150 }
151 
152 void UART::Write(uint8_t *data, int len) { //TBD }
153 
154 void UART::Read(uint8_t *data, int pos) {
155     //Get interrupt status
156     uint32_t ui32Status = UARTIntStatus(_UART.PERIPHERAL_CFG.UART_BASE,
157                                         true);
158     //Clear the asserted interrupts
159     UARTIntClear(_UART.PERIPHERAL_CFG.UART_BASE, ui32Status);
160 
161     //Asserts that are chars to read
162     if(!UARTCharsAvail(_UART.PERIPHERAL_CFG.UART_BASE)) {
163         data[pos] = 0;
164         return;
165     }
166 
167     data[pos] = UARTCharGet(_UART.PERIPHERAL_CFG.UART_BASE);
168 }
169 //---------------------------------------------------------------------
```

Communication_Handlers.hpp

```cpp
#ifndef COMMUNICATION_HANDLERS_HPP_
#define COMMUNICATION_HANDLERS_HPP_

/*C++ Section*/
#ifdef __cplusplus

#include <stdint.h>
#include <stdbool.h>
#include <stdarg.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_i2c.h"
#include "inc/hw_gpio.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/uart.h"
#include "driverlib/i2c.h"

#include "Communication_Interfaces.hpp"
#include "user_configs.h"

//Abstract base class
class Communication_Handler {
protected:
    bool ERROR_FLAG;
    Communication_Interface *Interface;

public:
    virtual int SendDatagram(uint8_t, uint8_t, uint16_t, uint16_t) = 0;
    virtual int ReceiveDatagram(uint8_t *, int) = 0;
};

//Modbus Handler
class CFW300_ModbusHandler : public Communication_Handler {
public:
    CFW300_ModbusHandler();
    ~CFW300_ModbusHandler();
    int SendDatagram(uint8_t addr, uint8_t func,
                     uint16_t reg, uint16_t value) override;
    int ReceiveDatagram(uint8_t *data, int len) override;
private:
```

```
45      uint8_t _Datagram_Send[8];
46      uint8_t _Datagram_Receive[8];
47      unsigned short CRC16(unsigned char *puchMsg, unsigned short usDataLen);
48  };
49
50  //I2C Handler
51  class MCP4725_I2CHandler : public Communication_Handler {
52  public:
53      MCP4725_I2CHandler();
54      ~MCP4725_I2CHandler();
55      int SendDatagram(uint8_t addr, uint8_t WRITE_MODE,
56                       uint16_t PWR_DOWN_MODE, uint16_t ref) override;
57      int ReceiveDatagram(uint8_t *data, int len) override;
58  private:
59      uint8_t _Datagram_Send[4];
60      uint8_t _Datagram_Receive[4];
61  };
62
63  //UART Handler
64  class LabView_UARTHandler : public Communication_Handler {
65  public:
66      LabView_UARTHandler();
67      ~LabView_UARTHandler();
68      int SendDatagram(uint8_t TBD1, uint8_t TBD2,
69                       uint16_t TBD3, uint16_t TBD4) override;
70      int ReceiveDatagram(uint8_t *data, int len) override;
71  private:
72      uint8_t _Datagram_Send[Labview_dataframeSize];
73      uint8_t _Datagram_Receive[Labview_dataframeSize];
74  };
75
76  #endif /*C++ Section*/
77  #endif /* COMMUNICATION_HANDLERS_HPP_ */
```

Communication_Handlers.cpp

```
1   #include "Communication_Handlers.hpp"
2   #include "Communication_Interfaces.hpp"
3   #include "user_configs.h"
4
5   extern const char ValidCommands[Labview_nCommands];
6   volatile bool DataframeIsFull = false;
7   volatile uint8_t ReceivedCommands[Labview_dataframeSize];
8
9   /* Table of CRC values for -highorder byte */
10  static unsigned char auchCRCHi[] = {
```

```
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x00, 0xC1, 0x81, 0x40, 0x01, 0xC0, 0x80, 0x41,
0x01, 0xC0, 0x80, 0x41, 0x00, 0xC1, 0x81, 0x40 };
/* Table of CRC values for -loworder byte */
static char auchCRCLo[] = {
0x00, 0xC0, 0xC1, 0x01, 0xC3, 0x03, 0x02, 0xC2,
0xC6, 0x06, 0x07, 0xC7, 0x05, 0xC5, 0xC4, 0x04,
0xCC, 0x0C, 0x0D, 0xCD, 0x0F, 0xCF, 0xCE, 0x0E,
0x0A, 0xCA, 0xCB, 0x0B, 0xC9, 0x09, 0x08, 0xC8,
0xD8, 0x18, 0x19, 0xD9, 0x1B, 0xDB, 0xDA, 0x1A,
0x1E, 0xDE, 0xDF, 0x1F, 0xDD, 0x1D, 0x1C, 0xDC,
0x14, 0xD4, 0xD5, 0x15, 0xD7, 0x17, 0x16, 0xD6,
0xD2, 0x12, 0x13, 0xD3, 0x11, 0xD1, 0xD0, 0x10,
0xF0, 0x30, 0x31, 0xF1, 0x33, 0xF3, 0xF2, 0x32,
0x36, 0xF6, 0xF7, 0x37, 0xF5, 0x35, 0x34, 0xF4,
0x3C, 0xFC, 0xFD, 0x3D, 0xFF, 0x3F, 0x3E, 0xFE,
0xFA, 0x3A, 0x3B, 0xFB, 0x39, 0xF9, 0xF8, 0x38,
0x28, 0xE8, 0xE9, 0x29, 0xEB, 0x2B, 0x2A, 0xEA,
```

```
58 0xEE, 0x2E, 0x2F, 0xEF, 0x2D, 0xED, 0xEC, 0x2C,
59 0xE4, 0x24, 0x25, 0xE5, 0x27, 0xE7, 0xE6, 0x26,
60 0x22, 0xE2, 0xE3, 0x23, 0xE1, 0x21, 0x20, 0xE0,
61 0xA0, 0x60, 0x61, 0xA1, 0x63, 0xA3, 0xA2, 0x62,
62 0x66, 0xA6, 0xA7, 0x67, 0xA5, 0x65, 0x64, 0xA4,
63 0x6C, 0xAC, 0xAD, 0x6D, 0xAF, 0x6F, 0x6E, 0xAE,
64 0xAA, 0x6A, 0x6B, 0xAB, 0x69, 0xA9, 0xA8, 0x68,
65 0x78, 0xB8, 0xB9, 0x79, 0xBB, 0x7B, 0x7A, 0xBA,
66 0xBE, 0x7E, 0x7F, 0xBF, 0x7D, 0xBD, 0xBC, 0x7C,
67 0xB4, 0x74, 0x75, 0xB5, 0x77, 0xB7, 0xB6, 0x76,
68 0x72, 0xB2, 0xB3, 0x73, 0xB1, 0x71, 0x70, 0xB0,
69 0x50, 0x90, 0x91, 0x51, 0x93, 0x53, 0x52, 0x92,
70 0x96, 0x56, 0x57, 0x97, 0x55, 0x95, 0x94, 0x54,
71 0x9C, 0x5C, 0x5D, 0x9D, 0x5F, 0x9F, 0x9E, 0x5E,
72 0x5A, 0x9A, 0x9B, 0x5B, 0x99, 0x59, 0x58, 0x98,
73 0x88, 0x48, 0x49, 0x89, 0x4B, 0x8B, 0x8A, 0x4A,
74 0x4E, 0x8E, 0x8F, 0x4F, 0x8D, 0x4D, 0x4C, 0x8C,
75 0x44, 0x84, 0x85, 0x45, 0x87, 0x47, 0x46, 0x86,
76 0x82, 0x42, 0x43, 0x83, 0x41, 0x81, 0x80, 0x40 };
77
78
79 //CFW300 ------------------------------------------------------------
80 unsigned short CFW300_ModbusHandler::CRC16(unsigned char *puchMsg,
81                                            unsigned short usDataLen)
82 {
83     unsigned char uchCRCHi = 0xFF; // high byte of CRC initialized
84     unsigned char uchCRCLo = 0xFF; // low byte of CRC initialized
85     unsigned uIndex; //will index into CRC lookup table */
86     while (usDataLen--) { // pass through message buffer
87         uIndex = uchCRCLo ^ *puchMsg++; // calculate the CRC
88         uchCRCLo = uchCRCHi ^ auchCRCHi[uIndex];
89         uchCRCHi = auchCRCLo[uIndex];
90     }
91     return (uchCRCHi << 8 | uchCRCLo);
92 }
93
94 CFW300_ModbusHandler::CFW300_ModbusHandler() {
95     Interface = new MODBUS_RTU(UART_CFW300, ENABLE_TX_PB4);
96 }
97 CFW300_ModbusHandler::~CFW300_ModbusHandler() {
98     delete Interface;
99 }
100
101 int CFW300_ModbusHandler::SendDatagram(uint8_t addr, uint8_t func,
102                                        uint16_t reg, uint16_t val) {
103     _Datagram_Send[0] = addr;                 //Device address
104     _Datagram_Send[1] = func;                 //Function
```

```
105     _Datagram_Send[2] = (reg & 0xFF00) >> 8; //8 most significant bits
106     _Datagram_Send[3] = (reg & 0x00FF);      //8 least significant bits
107     _Datagram_Send[4] = (val & 0xFF00) >> 8; //8 most significant bits
108     _Datagram_Send[5] = (val & 0x00FF);      //8 least significant bits
109 
110     uint16_t CRC = CRC16(_Datagram_Send, 6); //CRC16 based on message
111 
112     _Datagram_Send[6] = (CRC & 0x00FF);      //8 least significant bits
113     _Datagram_Send[7] = (CRC & 0xFF00) >> 8; //8 most significant bits
114 
115     Interface->Write(_Datagram_Send, 8);
116 
117     //Delay in microsseconds
118     //(Argument in number of timer loops. Each loop takes 3 clock cycles)
119     SysCtlDelay((MODBUS_END_FRAME_US*SysCtlClockGet()) / (3 * 1000000));
120 
121     //check returned datagram
122     //if(TBD){
123         //return 0;
124     //}
125 
126     return 1;
127 }
128 
129 int CFW300_ModbusHandler::ReceiveDatagram(uint8_t *data, int len) {
130     uint16_t CRC; //CRC received
131 
132     //Assert len
133     if (len > Labview_dataframeSize)
134         return 0;
135 
136     //Clear previous Datagram
137     uint8_t i;
138     for(i = 0; i < 8; i++)
139         _Datagram_Receive[i] = 0;
140 
141     //Read bytes from UART
142     Interface->Read(_Datagram_Receive, len);
143 
144     for(i = 0; i < len; i++)
145         data[i] = _Datagram_Receive[i]; //Datagram received
146 
147     //Check message
148     //ERROR: If received function is different from the sent function
149     if(_Datagram_Receive[0] != _Datagram_Send[0])
150         return 0;
151 
```

```
152     //CRC Received (last 2 bytes)
153     CRC = (_Datagram_Receive[len-2] << 8) | _Datagram_Receive[len-1];
154 
155     //ERROR: CRC mismatch
156     if(CRC != CRC16(_Datagram_Receive, len-2))
157         return 0;
158 
159     //If function and CRC match, return success
160     return 1;
161 }
162 //---------------------------------------------------------------------
163 
164 //MCP4725 ------------------------------------------------------------
165 MCP4725_I2CHandler::MCP4725_I2CHandler() {
166     Interface = new I2C(I2C_MCP4725);
167 }
168 
169 MCP4725_I2CHandler::~MCP4725_I2CHandler() {
170     delete Interface;
171 }
172 
173 int MCP4725_I2CHandler::SendDatagram(uint8_t addr, uint8_t WRITE_MODE,
174                                      uint16_t PWR_DOWN_MODE,
175                                      uint16_t ref) {
176     if (ref > 4095)
177         ref = 4095;
178 
179     _Datagram_Send[0] = addr;
180     _Datagram_Send[1] = (WRITE_MODE << 4) | PWR_DOWN_MODE;
181     //Split integer reference into 2 bytes
182     _Datagram_Send[2] =  (ref & 0x0FF0)  >> 4;
183     //Last 4 bits are unused - forced to be 0
184     _Datagram_Send[3] = ((ref & 0x000F)  << 4) & 0x00F0;
185 
186     Interface->Write(_Datagram_Send, 4);
187 
188     //check returned datagram
189         //if(TBD){
190             //return 0;
191         //}
192 
193     return 1;
194 }
195 
196 int MCP4725_I2CHandler::ReceiveDatagram(uint8_t *data, int len) {
197     return -1;
198 }
```

```
199
200 //LabView ------------------------------------------------
201 LabView_UARTHandler::LabView_UARTHandler() {
202     Interface = new UART(UART_LABVIEW);
203 }
204 LabView_UARTHandler::~LabView_UARTHandler() {
205     delete Interface;
206 }
207
208 int LabView_UARTHandler::SendDatagram(uint8_t TBD1, uint8_t TBD2,
209                                       uint16_t TBD3, uint16_t TBD4) {
210     return -1;
211 }
212
213 int LabView_UARTHandler::ReceiveDatagram(uint8_t *data, int len) {
214     int i;
215     if(DataframeIsFull) {
216         for(i = 0; i < Labview_dataframeSize; i++) {
217             _Datagram_Receive[i]    = ReceivedCommands[i];
218             data[i]                 = ReceivedCommands[i];
219         }
220         DataframeIsFull = false;
221         return Labview_dataframeSize;
222     }
223     return 0; //If dataframe is incomplete
224 }
225
226
227 extern "C" void InterruptHandler(void) {
228     if(DataframeIsFull)
229             return;
230
231     static uint8_t RxData[Labview_dataframeSize];
232     static uint8_t pos = 0;
233     bool initDataFrame = false;
234
235     UART intUART(UART_LABVIEW);
236     intUART.Read(RxData, pos);    //Read from UART
237
238     int i = 0;
239     //Only assembly dataframe when receive
240     //the initial dataframe char (ValidCommands)
241     for(i = 0; i < Labview_nCommands; i++) {
242         if(RxData[0] == ValidCommands[i]) {
243             initDataFrame = true;
244             pos++;
245         }
```

```
246     }
247 
248     //Make the interrupt waits until receive a valid start char
249     if(!initDataFrame) {
250         DataframeIsFull = false;
251         RxData[0] = 0;
252         pos = 0;
253     }
254 
255     //When reaches the standard size of dataframe
256     if(pos == Labview_dataframeSize){
257         //Signals there is a complete dataframe available
258         DataframeIsFull = true;
259 
260         //Pass the RxData to ReceivedCommands and clear the buffer
261         for(i = 0; i < Labview_dataframeSize; i++){
262             ReceivedCommands[i] = RxData[i];
263             RxData[i] = 0;
264         }
265         pos = 0;
266     }
267 }
```

Devices.hpp

```
1  #ifndef DEVICES_HPP_
2  #define DEVICES_HPP_
3  
4  #include <stdint.h>
5  #include <stdbool.h>
6  #include <stdarg.h>
7  #include "inc/hw_ints.h"
8  #include "inc/hw_memmap.h"
9  #include "inc/hw_types.h"
10 #include "inc/hw_i2c.h"
11 #include "inc/hw_gpio.h"
12 #include "driverlib/gpio.h"
13 #include "driverlib/interrupt.h"
14 #include "driverlib/pin_map.h"
15 #include "driverlib/sysctl.h"
16 #include "driverlib/uart.h"
17 #include "driverlib/i2c.h"
18 
19 #include "board_configs.h"
20 #include "user_configs.h"
21 #include "Communication_Handlers.hpp"
```

```
22
23 /*C++ Section*/
24 #ifdef __cplusplus
25
26 class CFW300_FrequencyInverter {
27 public:
28     CFW300_FrequencyInverter(uint8_t ADDR);
29     ~CFW300_FrequencyInverter();
30     bool getMotorON();
31     int setMotorON(bool is_ON);
32     uint16_t getVelRef13b();
33     int setVelRef13b(int velRef);
34 private:
35     Communication_Handler* ModbusHandler;
36     uint8_t _ModBusADDR;
37     uint16_t _velRef13b;
38     bool _motorON;
39
40     enum _Register {
41         P680 = 0x02A8, //RO  - Device status monitoring
42         P681 = 0x02A9, //RO  - Frequency reference in 13 bits
43         P682 = 0x02AA, //R/W - Control word via Serial
44         P683 = 0x02AB, //R/W - Frequency reference via Serial (0-8192)
45     };
46     enum _Function {
47         WRITE = 0x06,
48         READ  = 0x03,
49     };
50     enum _State {
51         ON  = 0x0017, //00010111 (P682)
52         OFF = 0x0016, //00010110 (P682)
53     };
54 };
55
56 class MCP4725_DAC {
57 public:
58     MCP4725_DAC(uint8_t ADDR);
59     ~MCP4725_DAC();
60     uint16_t getVoltageLevel();
61     int setVoltageLevel(uint8_t voltageRef12b);
62 private:
63     Communication_Handler* I2CHandler;
64     uint8_t _I2CADDR;
65     uint16_t _voltageRef12b;
66
67     //Power Down Selection |X=0|PD1|PD0|X=0|
68     //In the power-down mode:
```

```
69      //VOUT is off and most of internal circuits are disabled.
70      enum _PowerMode {
71          //Normal Mode (X = 0, PD1 = 0, PD0 = 0, X = 0)
72          NORMAL_MODE     = 0x0000,
73          //1K  Ohms resistor to ground (X = 0, PD1 = 0, PD0 = 1, X = 0)
74          PULL_DOWN_1K    = 0x0002,
75          //100K Ohms resistor to ground (X = 0, PD1 = 1, PD0 = 0, X = 0)
76          PULL_DOWN_100K  = 0x0004,
77          //500K Ohms resistor to ground (X = 0, PD1 = 1, PD0 = 1, X = 0)
78          PULL_DOWN_500K  = 0x0006,
79      };
80
81      enum _Function {
82          //Write DAC Register only : (C2 = 0, C1 = 1, C0 = 0, X = 0)
83          DAC_ONLY        = 0x00,
84          //Write DAC Register and EEPROM: (C2 = 0, C1 = 1, C0 = 1, X = 0)
85          DAC_EEPROM      = 0x06,
86      };
87  };
88
89  class LabView_ControlComputer {
90  public:
91      LabView_ControlComputer(MCP4725_DAC *MCP4725,
92                              CFW300_FrequencyInverter *CFW300_1,
93                              CFW300_FrequencyInverter *CFW300_2);
94      ~LabView_ControlComputer();
95      void Run();
96  private:
97      int ReceiveLabviewCommands();
98      int SendResponseMessage();
99
100     uint8_t     _Datagram[Labview_dataframeSize];
101     uint16_t    _velRefCFW300[2];
102     bool        _motorON[2];
103     uint16_t    _voltageRefMCP4725;
104
105     Communication_Handler *UARTHandler;
106
107     //Devices
108     MCP4725_DAC                 *_MCP4725;
109     CFW300_FrequencyInverter    *_CFW300[2];
110 };
111
112
113 #endif /*C++ Section*/
114 #endif /* DEVICES_HPP_ */
```

Devices.cpp

```cpp
#include "Devices.hpp"

//CFW300
CFW300_FrequencyInverter::CFW300_FrequencyInverter(uint8_t ADDR) {
    ModbusHandler = new CFW300_ModbusHandler();
    _ModBusADDR = ADDR;
    _velRef13b = 0;

    setVelRef13b(0);
    setMotorON(false);
}

CFW300_FrequencyInverter::~CFW300_FrequencyInverter() {
    delete ModbusHandler;
}

bool CFW300_FrequencyInverter::getMotorON() {
    return _motorON;
}

int CFW300_FrequencyInverter::setMotorON(bool is_ON) {
    //Prevent useless messages
   if (is_ON == _motorON)
       return 1;

    _motorON = is_ON;

    int res = 0;

    //Set state of motor in P682
    if(_motorON)
        res = ModbusHandler->SendDatagram(_ModBusADDR, _Function::WRITE,
                                          _Register::P682, _State::ON);
    else
        res = ModbusHandler->SendDatagram(_ModBusADDR, _Function::WRITE,
                                          _Register::P682, _State::OFF);

    return res;
}

uint16_t CFW300_FrequencyInverter::getVelRef13b() {
    return _velRef13b;
}

```

```
45 int CFW300_FrequencyInverter::setVelRef13b(int velRef) {
46     //Prevent useless messages
47     if (velRef == _velRef13b)
48         return 1;
49 
50     _velRef13b = velRef;
51 
52     return ModbusHandler->SendDatagram(_ModBusADDR, _Function::WRITE,
53                                        _Register::P683, velRef);
54 }
55 //---------------------------------------------------------------------------
56 
57 //MCP4725
58 MCP4725_DAC::MCP4725_DAC(uint8_t ADDR){
59     _I2CADDR = ADDR;
60     I2CHandler = new MCP4725_I2CHandler();
61     _voltageRef12b = 4095;
62 
63     //Write in DAC_EEPROM mode to to keep the valve open in case of failure
64     //EEPROM value is load at startup of MCP4725
65     I2CHandler->SendDatagram(_I2CADDR, _Function::DAC_EEPROM,
66                              _PowerMode::NORMAL_MODE, _voltageRef12b);
67 }
68 
69 MCP4725_DAC::~MCP4725_DAC() {
70     delete I2CHandler;
71 }
72 
73 uint16_t MCP4725_DAC::getVoltageLevel() {
74     return _voltageRef12b;
75 }
76 
77 int MCP4725_DAC::setVoltageLevel(uint8_t voltageRef12b) {
78     //Prevent useless messages
79     if (voltageRef12b == _voltageRef12b)
80         return 1;
81 
82     _voltageRef12b = voltageRef12b;
83 
84     //Write in DAC_ONLY mode and return state
85     return I2CHandler->SendDatagram(_I2CADDR, _Function::DAC_ONLY,
86                                     _PowerMode::NORMAL_MODE,
87                                     _voltageRef12b);
88 }
89 //---------------------------------------------------------------------------
90 
91 //LabView
```

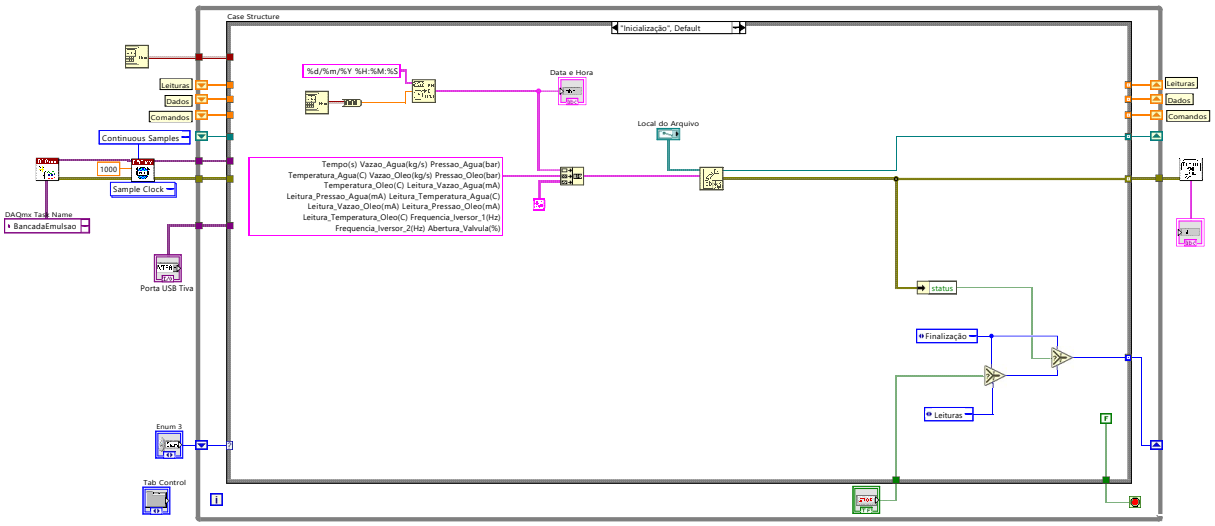
```
92 LabView_ControlComputer::LabView_ControlComputer(MCP4725_DAC
93                                                 *MCP4725,
94                                                 CFW300_FrequencyInverter
95                                                 *CFW300_1,
96                                                 CFW300_FrequencyInverter
97                                                 *CFW300_2) {
98     UARTHandler = new LabView_UARTHandler();
99 
100    _MCP4725    = MCP4725;
101    _CFW300[0]  = CFW300_1;
102    _CFW300[1]  = CFW300_2;
103 
104    _velRefCFW300[0]    = 0;
105    _velRefCFW300[1]    = 0;
106    _motorON[0]         = false;
107    _motorON[1]         = false;
108    _voltageRefMCP4725  = 4095;
109 }
110 
111 LabView_ControlComputer::~LabView_ControlComputer() {
112    delete UARTHandler;
113 }
114 
115 int LabView_ControlComputer::ReceiveLabviewCommands() {
116    char command, device, aux1, aux2;
117 
118    //Wait until get a complete datagram
119    while(!UARTHandler->ReceiveDatagram(_Datagram, Labview_dataframeSize));
120 
121    command     = _Datagram[0];
122    device      = _Datagram[1];
123    aux1        = _Datagram[2];
124    aux2        = _Datagram[3];
125 
126    //Only changes the control variables according to the received command
127    switch (command) {
128        //Command D: Turn Off CFW300_x inverter
129        case 'D':
130            if (device == '1')
131                _motorON[0] = false; //Turn Off Motor 1
132           else if (device == '2')
133                _motorON[1] = false; //Turn Off Motor 2
134 
135            break;
136 
137        //Command D: Turn Off CFW300_x inverter
138        case 'L':
```

```
139             if (device == '1')
140                 _motorON[0] = true; //Turn On Motor 1
141             else if (device == '2')
142                 _motorON[1] = true; //Turn On Motor 2
143 
144             break;
145 
146         //Command V: Set velref of CFW300_x inverter
147         case 'V':
148             if (device == '1')
149                 //Update Velref Motor 1
150                 _velRefCFW300[0] = (aux1 << 8) | aux2;
151             else if (device == '2')
152                 //Update Velref Motor 2
153                 _velRefCFW300[1] = (aux1 << 8) | aux2;
154 
155             break;
156 
157         //Command A: Set VoltageRef of MCP4725
158         case 'A':
159             //Update VoltageRef MCP4725 2
160             _voltageRefMCP4725 = (aux1 << 8) | aux2;
161             break;
162 
163         //If the command is not recognized, return fail
164         default:
165             return 0;
166     }
167 
168     return 1;
169 }
170 
171 void LabView_ControlComputer::Run() {
172     while(1) {
173         //Wait until get valid dataframe with valid command
174         while(ReceiveLabviewCommands() == 0);
175 
176         int i = 0;
177 
178         //Send all command variables,
179         //treatments to avoid repeated messages are performed
180         //internally in devices
181         _MCP4725->setVoltageLevel(_voltageRefMCP4725);
182         for(i = 0; i < 2; i++) {
183             _CFW300[i]->setVelRef13b(_velRefCFW300[i]);
184             _CFW300[i]->setMotorON(_motorON[i]);
185         }
```

```
186     }
187 }
188
189 int LabView_ControlComputer::SendResponseMessage() { return -1; }
190 //-----------------------------------------------------------------------
```

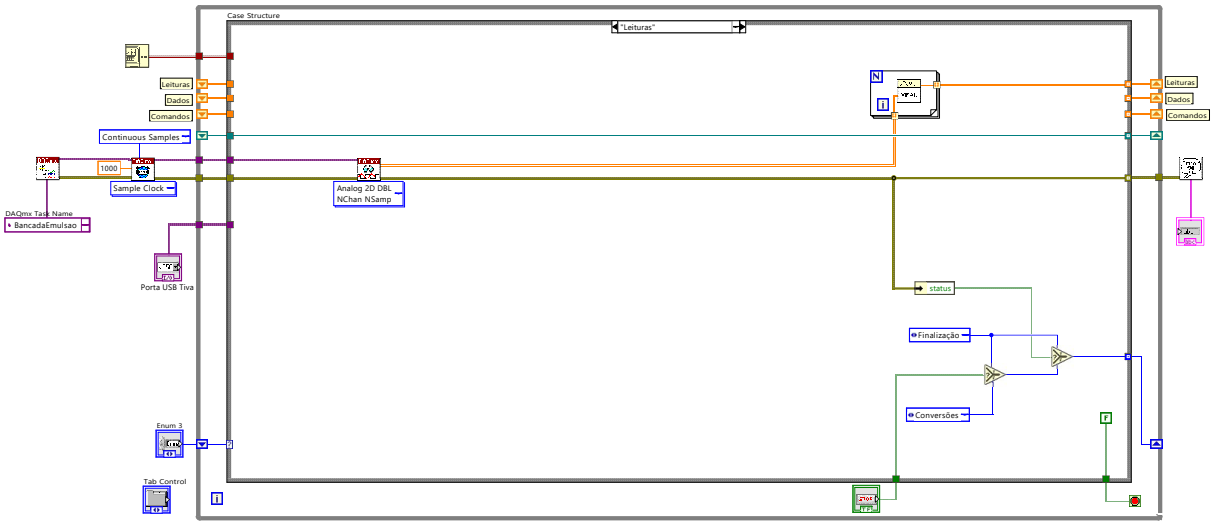
APÊNDICE B - IMPLEMENTAÇÕES DO SOFTWARE NO LABVIEW

Figura 78 – Inicialização.



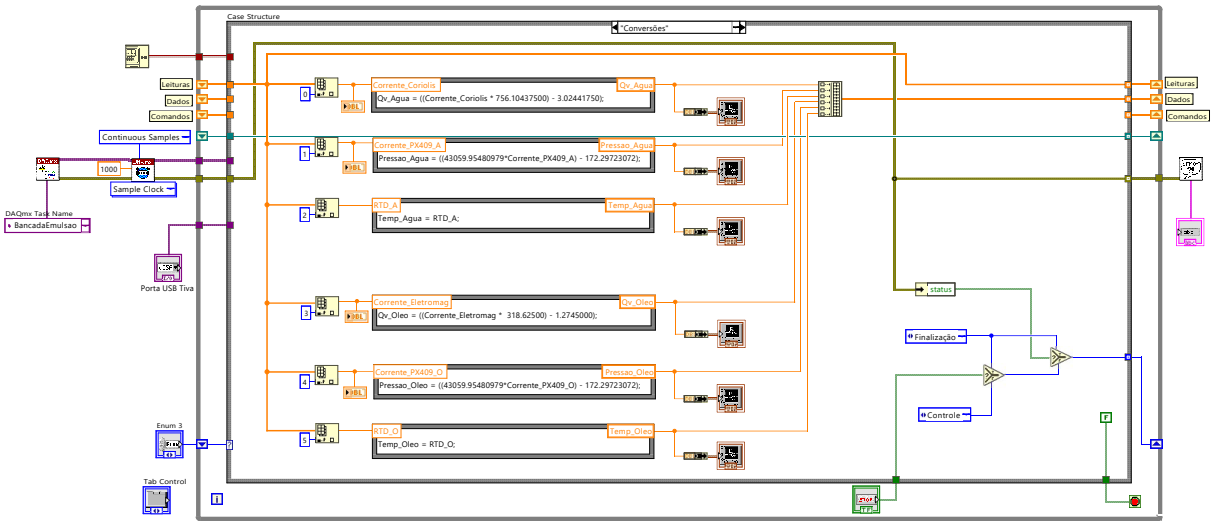
Fonte: autor (2022).

Figura 79 – Leituras.



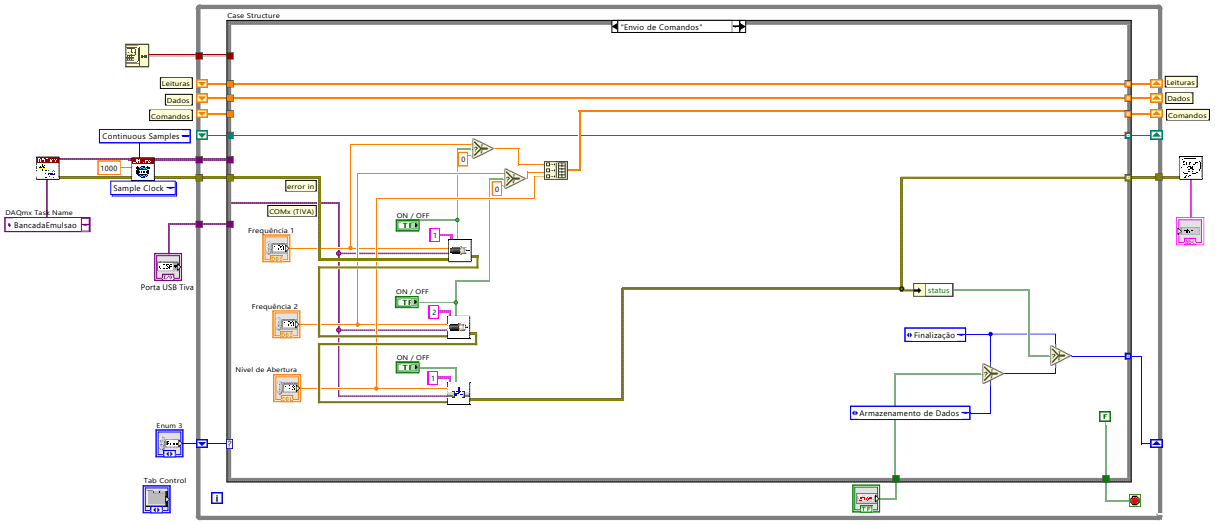
Fonte: autor (2022).

Figura 80 – Conversões.



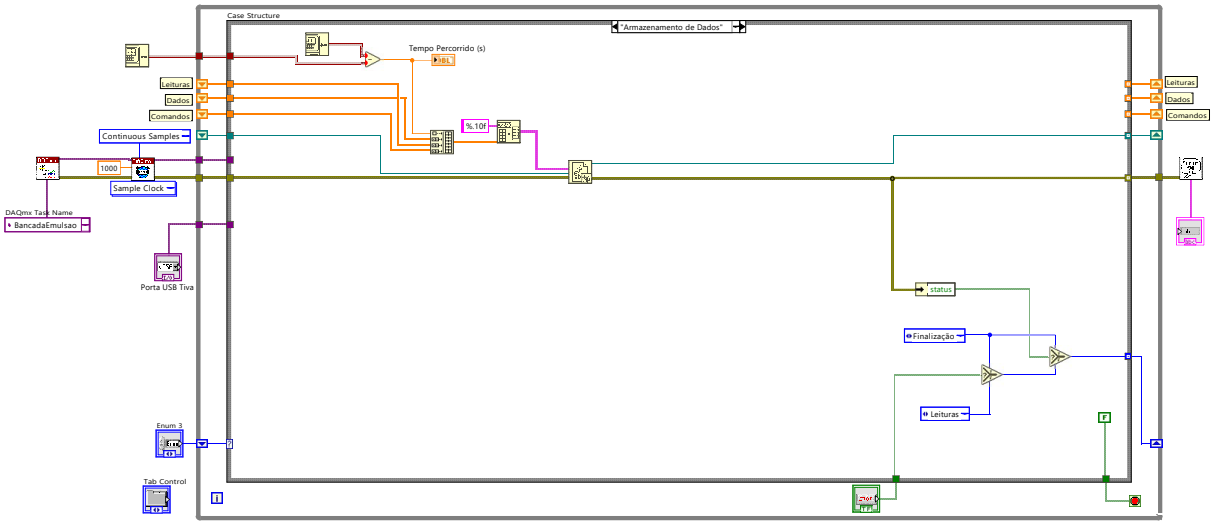
Fonte: autor (2022).

Figura 81 – Envio de Comandos.



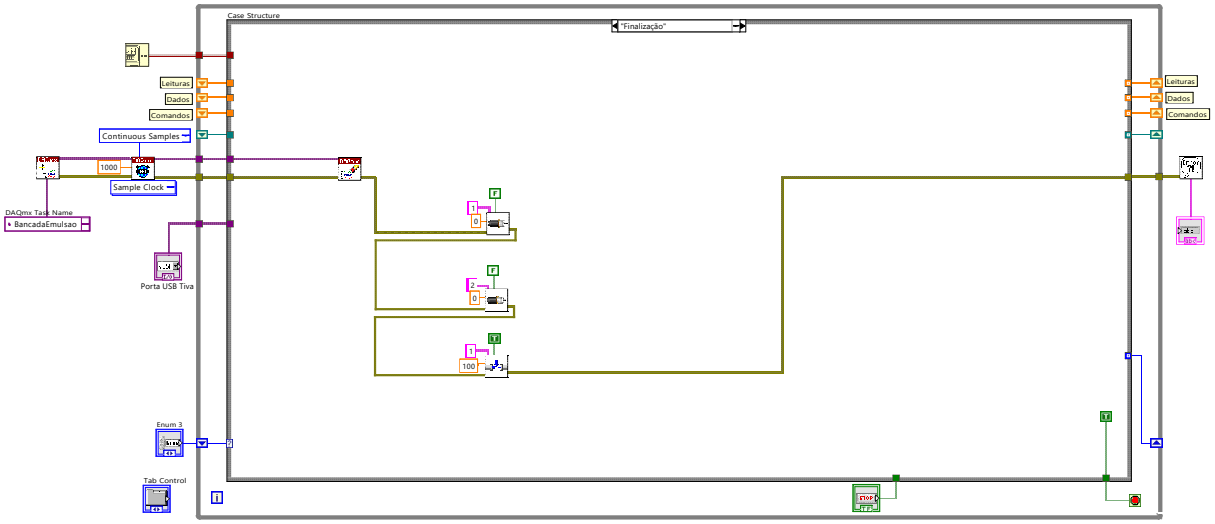
Fonte: autor (2022).

Figura 82 – Armazenamento de Dados.



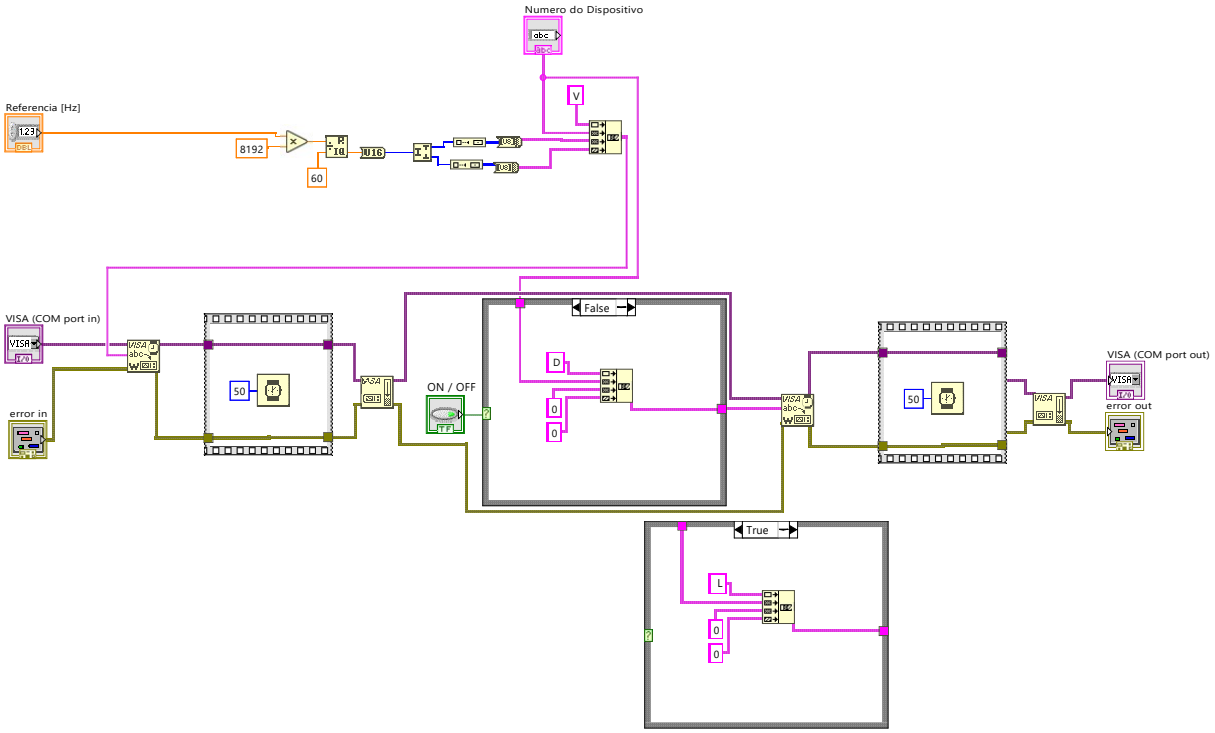
Fonte: autor (2022).

Figura 83 – Finalização.



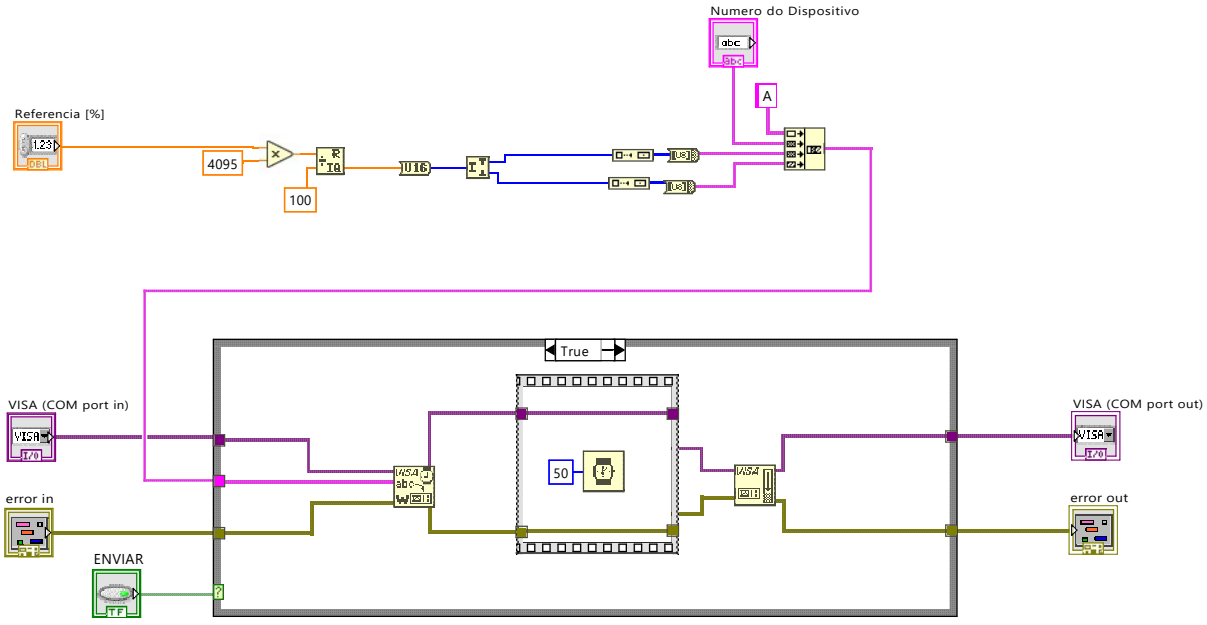
Fonte: autor (2022).

Figura 84 – Comandos Inversores.



Fonte: autor (2022).

Figura 85 – Comandos Válvula.



Fonte: autor (2022).