UNIVERSIDADE DO VALE DO RIO DOS SINOS - UNISINOS
UNIDADE ACADÊMICA DE GRADUAÇÃO
CURSO DE ENGENHARIA MECÂNICA

JONATAS SOARES NOLASCO

DESENVOLVIMENTO DE HARDWARE, SOFTWARE E INTERFACE PARA CONTROLE E MONITORAMENTO DE SISTEMAS HIDRÁULICOS PROPORCIONAIS

São Leopoldo
2022

JONATAS SOARES NOLASCO

DESENVOLVIMENTO DE HARDWARE, SOFTWARE E INTERFACE PARA CONTROLE E MONITORAMENTO DE SISTEMAS HIDRÁULICOS PROPORCIONAIS

Trabalho de Conclusão de Curso apresentado como requisito parcial para obtenção do título de Bacharel em Engenharia Mecânica, pelo Curso de Engenharia Mecânica da Universidade do Vale do Rio dos Sinos – UNISINOS.

Orientador: Prof. Ms. Daniel Faistauer

São Leopoldo

2022

AGRADECIMENTOS

Primeiramente agradeço a Deus pela oportunidade recebida. Agradeço em especial ao meu pai José E. A. Nolasco, por tudo que me possibilitou sendo metalúrgico, um trabalhador de excelência, a minha mãe Maria Adelia S. Nolasco, por todos os conselhos e amparo nos momentos mais difíceis. Gostaria de agradecer também todo o apoio que tive das minhas irmãs Andressa e Vanessa.

Agradeço todo apoio da minha esposa Bruna Milanez, pela paciência e compreensão em todos os momentos difíceis ao longo do curso.

Aos meus amigos e colegas de trabalho, pelos conselhos e apoios, principalmente nas minhas escolhas profissionais.

Gostaria de agradecer a instituição de ensino Unisinos, toda sua infraestrutura, qualidade de ensino e professores.

Um agradecimento especial ao Prof. Ms. Daniel Faistauer pelo tempo dedicado, conselhos e a amizade desenvolvida, tanto no decorrer das disciplinas lecionadas quanto durante o trabalho de conclusão do curso.

RESUMO

A hidráulica tem papel primordial em soluções de movimentações de cargas e está presente em diversas áreas, principalmente, na agricultura e nas construções civil e naval. Entre tantas aplicações, os sistemas automatizados ganham ênfase, se tornando cada vez mais próximos a um movimento humano. Neste contexto, o trabalho consiste em desenvolver sistemas que abrangem vastas aplicações, com a finalidade de unir alto desempenho e baixo custo, tendo em vista que sistemas com controladores lógicos programáveis (CLP), voltados para o mercado, possuem custo elevado. Os *softwares* desenvolvidos possuem interfaces para acionamento e monitoramento em tempo real e foram introduzidos e aplicados para Arduino Uno, que é uma ferramenta muito utilizada em prototipagem de sistemas automatizados. São apresentados diagramas hidráulicos que requerem controles proporcionais, seus respectivos programas e suas interfaces de controle, além de abordar aplicações que exigem sistemas sofisticados utilizando técnicas de programação e o controle Proporcional Integral Derivativo (PID), apresentados ao longo do trabalho.

Palavras-chaves: automação; Arduino; supervisório; sistemas hidráulicos.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVIATURAS

ASCII	American Standard Code for Information Interchange
BASIC	Beginners All-purpose Symbolic Instruction Code
C	Linguagem de Programação Compilada
C++	Linguagem de Programação Multiplataforma
CLP	Controlador lógico programável
CNC	Comando Numérico Computadorizado
COM	Porta de Comunicação
CPU	Central Processing Unit
DOS	Ambiente monousuário e mono processamento
EEPROM	Electrically Erasable Programmable Read-Only Memory
EPROM	Erasable Programmable Read-Only Memory
FBD	Function Block Dia-gram
HMI	Human Machine Interface
I/O	Input/Output
IDE	Integrated Development Environment
IEC	International Electrotechnical Commission
IHM	Interface Homem Máquina
IL	Instruction List
ISO	International Organization for Standardization
KD	Ganho Derivativo
KI	Ganho Integral
KP	Ganho Proporcional
LD	Ladder Diagram
LPT	Terminal de Impressão
MODBUS	Protocolo de comunicação de dados
OSI	Open Systems Interconnection
PC	Personal Computer
PID	Proporcional Integral Derivativo
PIDT1	Proporcional Integral Derivativo
PLC	Programable Logic Controller
POU	Program Organization Unit

PWM	Pulse Width Modulation
RAM	Memória de Acesso Aleatória
RLL	Relay Ladder Logic
ROM	Memória Somente para Leitura
RTU	Remote Terminal Unit
SCADA	Supervisory Control and Data Acquisition
SDCD	Sistemas Digitais de Controle Distribuído
SFC	Sequential Function Chart
ST	Structured Text
UNISINOS	Universidade do Vale do Rio dos Sinos
VT-HACD	Controle Digital em Malha Fechada
VT-HMC	Controlador Digital com Controle de Eixo Integrado

LISTA DE SÍMBOLOS

A1	Área 1	[m²]
A2	Área 2	[m²]
F0	Força Inicial	[N]
F1	Força 1	[N]
F2	Força 2	[N]
kHz	Quilohertz	[kHz]
L	Distância	[m]
mA	Miliampere	[mA]
P	Pressão	[MPa]
VAC	Tensão em corrente alternada	[V]
VDC	Tensão em corrente contínua	[V]
X	Distância	[m]

SUMÁRIO

1. INTRODUÇÃO

Com a elevada necessidade de tecnologias para movimentos e aplicação de cargas, o sistema hidráulico passa por evoluções constantes nos tipos de acionamentos. Inicialmente manuais, o sistema partiu para evoluções onde inseriu-se acionamentos por pilotagem hidráulica e pneumática, sendo aplicados posteriormente acionamentos por solenoides.

Sistemas com acionamentos por solenoides estão em contínua evolução, pois possibilitam diversos tipos de sinais eletroeletrônicos como sistemas liga/desliga e proporcionais. Em paralelo, o Arduino é uma tecnologia acessível que contribui na tecnologia de acionamentos, pois por linguagem de programação pode executar diversos tipos de sistemas de automação.

1.1 SITUAÇÃO PROBLEMA

O tema do presente trabalho consiste em uma alternativa para o alto custo em sistemas de acionamentos automatizados para sistemas hidráulicos. O problema engloba buscar, através do uso do Arduino e sua linguagem de programação, alternativas para substituir acionadores de mercado.

1.2 OBJETIVOS

O objetivo geral deste trabalho é desenvolver, através do Arduino, um sistema alternativo para controle em eixo hidráulico, além de verificar através de linguagem de programação os diversos efeitos sob o sistema hidráulico.

1.2.1 Objetivos Específicos

Esse trabalho tem como objetivo desenvolver um produto para acionamento e controle de sistemas hidráulicos contemplando:

a) Um *hardware* com Arduino, fonte, amplificadores e conectores;

b) Programações para acionamento de eletroválvulas digitais e analógicas (válvulas direcionais, alívio e controle de vazão);

c) Interface para controle e monitoramento do sistema hidráulico;

d) Diagramas hidráulicos com controle proporcional;

e) Validação de resultados através de testes;

1.3 DELIMITAÇÃO DO TEMA

Esse trabalho consiste no desenvolvimento de um produto de baixo custo e alto desempenho, que possibilite automatizar e monitorar sistemas hidráulicos, através de um *hardware* e programas para diversos tipos de aplicações que necessitem de controle proporcional.

1.4 PROBLEMA

Maquinários agrícolas, minerações, pavimentações e processos industriais a cada dia se modernizam e implementam novas funções que simulam a capacidade motora de uma pessoa, tanto em articulação quanto em sensibilidade. Visando a modernização e proporcionalidade percebe-se a necessidade de sistemas que possibilitem a sensibilização do sistema hidráulico. Entretanto, sistemas proporcionais tem um custo elevado se comparado a sistemas de acionamento liga/desliga.

1.5 JUSTIFICATIVA

Devido à alta competitividade no mercado em relação a redução de custo, o intuito deste trabalho é desenvolver um sistema alternativo que visa suprir a necessidade de sistemas de automação de alto desempenho e que tenham um custo competitivo.

Com a disponibilidade da infraestrutura da instituição UNISINOS, será possível simular diversas condições de trabalho, podendo comparar sistemas de acionamentos de mercado com o sistema alternativo proposto neste trabalho.

1.6 ESTRUTURA DO TRABALHO

O trabalho será dividido em seis capítulos, onde:

O capítulo 1 contempla a parte introdutória, situação problema, objetivos, delimitação do tema e justificativa.

O capítulo 2 se refere à fundamentação teórica, agregando o conteúdo base de estudo e a obtenção de dados para o trabalho.

O capítulo 3 mostrará as ferramentas para o desenvolvimento de programas para controle proporcional de sistemas hidráulicos.

O capítulo 4 exibirá as interfaces de controle, o *hardware,* a programação para o Arduino e os diagramas hidráulicos.

O capítulo 5 apresentará a análise do funcionamento do produto desenvolvido e os resultados iniciais obtidos.

No capítulo 6 é exposta a conclusão obtida ao término deste trabalho e sugestões para trabalhos futuros.

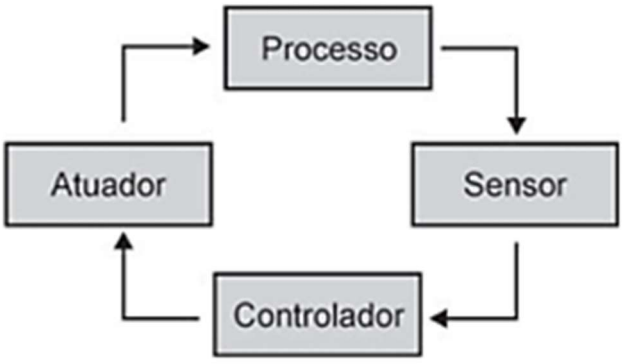
2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta os principais referenciais teóricos estudados nesta pesquisa, para que posteriormente possa fundamentar o desenvolvimento do projeto e a análise dos dados coletados.

2.1 AUTOMAÇÃO

De acordo com Silveira (2012, p. 20), a automação é um conceito e um conjunto de técnicas por meio das quais se constroem sistemas ativos capazes de atuar com uma eficiência ótima pelo uso de informações recebidas do meio sobre o qual atuam. Com base nessas informações, o sistema calcula a ação corretiva mais apropriada para a execução da ação, sendo esta, uma característica de sistemas em malha fechada conhecidos como sistemas de realimentação, ou seja, aquele que mantém uma relação expressa entre o valor de saída e o da entrada de referência do processo. Essa relação entrada/saída serve para corrigir eventuais valores na saída que estejam fora dos valores desejados. Para tanto, são utilizados controladores que por meio da execução algorítmica de um programa ou circuito eletrônico comparam o valor atual com o valor desejado, efetuando o cálculo para ajuste e correção. O valor desejado também é conhecido na literatura inglesa como *setpoint*. A Figura 1 apresenta o diagrama de blocos de um sistema de automação, mostrando o fluxo de lógica.

Figura 1 – Diagrama de blocos de um sistema de automação.



Fonte: Silveira (2012, p. 20).

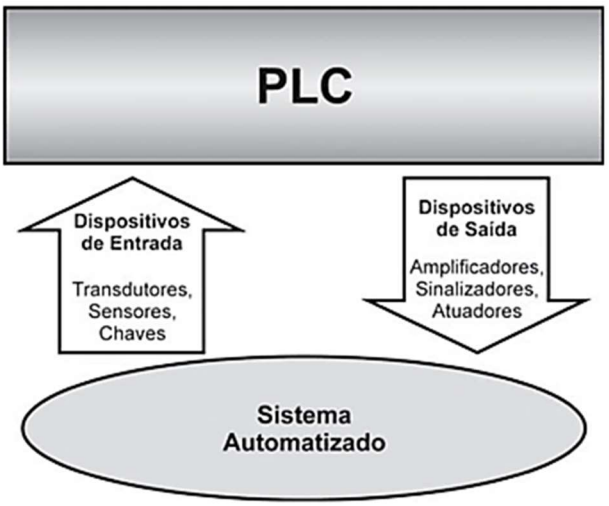
De acordo com Scheffer (2018, p.14), um sistema de controle é um sistema projetado para adequar um ou mais sinais a padrões de projeto previamente determinados.

Segundo Silveira (2012, p. 21), muitas das aplicações existentes destinadas ao controle de processos se mostram insatisfatórias, pois dentro de um curto período, existe a necessidade de amostrar o sinal a ser controlado e de obter uma alta velocidade de resposta. Basta um atraso na realimentação do sistema e os novos dados vão gerar uma solução de controle baseada em valores passados. O problema será tão maior quanto maior for o seu atraso. Seu estudo e determinação são feitos pela análise dinâmica do processo. Tais problemas existem e são geralmente encontrados em sistemas de controle em tempo real.

2.2 CONTROLADOR LÓGICO PROGRAMÁVEL (CLP)

O PLC (*Programable Logic Controller*) ou CLP (Controlador lógico programável) é definido como um computador industrial com capacidade de armazenar instruções de funções de controle, além de realizar operações lógicas, aritméticas, e comunicação em rede. A Figura 2 apresenta uma aplicação genérica do CLP.

Figura 2 – Aplicação genérica do Controlador Lógico Programável.



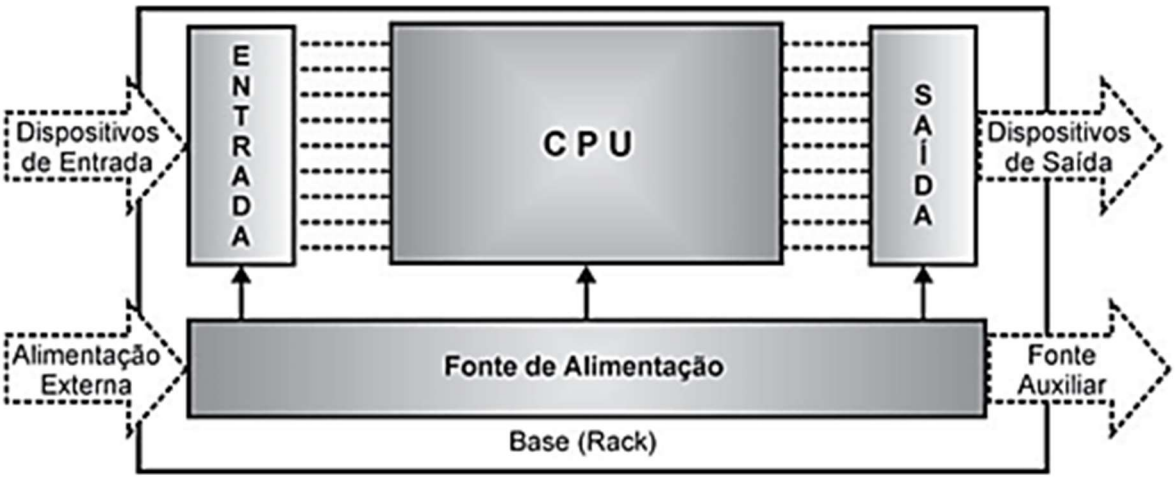
Fonte: Georgini (2016).

De acordo com Georgini (2016, p.48), os principais blocos que compões um PLC são:

- CPU (*Central Processing Unit* - Unidade Central de Processamento): compreende o processador (microprocessador, microcontrolador ou processador dedicado), o sistema de memória (ROM e RAM) e os circuitos auxiliares de controle;
- Circuitos/Módulos de I/O (*Input/Output* - Entrada/Saída): podem ser discretos (sinais digitais: 12VDC, 110VAC, contatos normalmente abertos, contatos normalmente fechados) ou analógicos (sinais analógicos: 4-20mA, 0-10VDC, termopar);
- Fonte de alimentação: responsável pela tensão de alimentação fornecida à CPU e aos Circuitos/Módulos de I/O. Em alguns casos, proporciona saída auxiliar (baixa corrente);
- Base ou Rack: Proporciona conexão mecânica ou elétrica entre a CPU, os Módulos de I/O e a Fonte de Alimentação. Contém barramento de comunicação entre eles onde os sinais de dados, endereço, controle e tensão de alimentação estão presentes;
- Circuitos/Módulos Especiais: contém contador rápido (5kHz, 10kHz, 100kHz ou mais), interrupção por *hardware*, controlador de temperatura, controlador PID, processadores (transmissão via rádio, posicionamento de eixos, programação BASIC, sintetizador de voz, entre outros) e comunicação em rede, entre outros componentes.

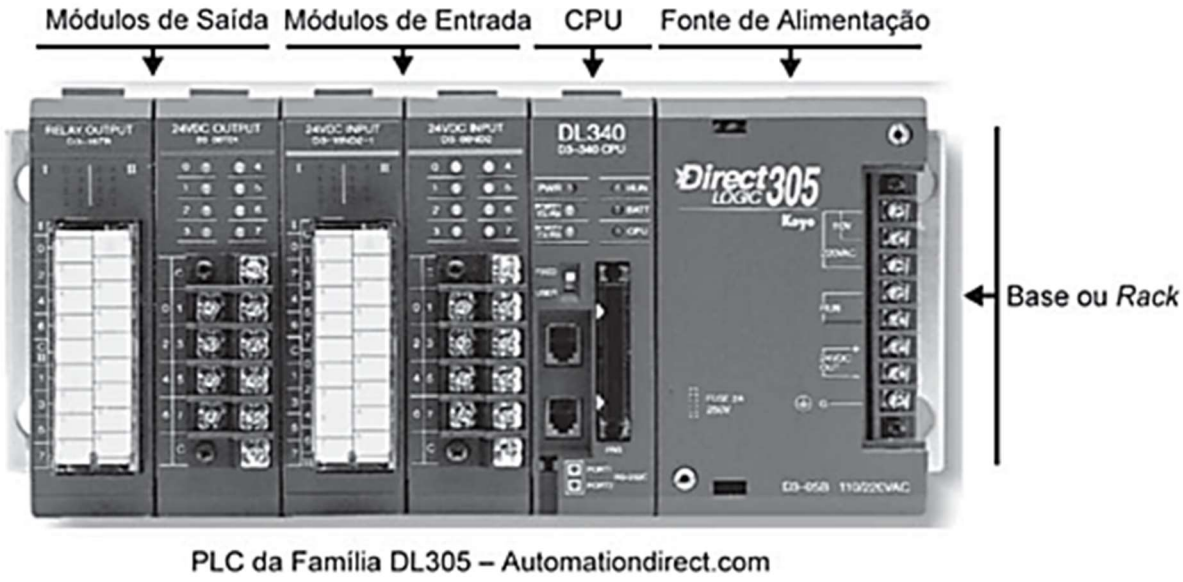
A Figura 3 mostra a estrutura básica de um CLP por meio de blocos descritos. E um CLP comercial é apresentado na Figura 4.

Figura 3 – Estrutura básica do PLC.



Fonte: Georgini (2016).

Figura 4 – Exemplo de PLC disponível no mercado.



Fonte: Georgini (2016).

2.2.1 Operação Básica

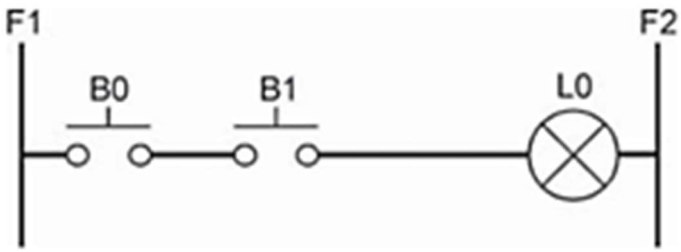
A execução de leitura da CPU é dada através dos dispositivos de entradas por meio dos circuitos/módulos de Entrada/Saída.

De acordo com Petruzella (2013, p.21), o processador (CPU) é o “cérebro” de um CLP e consiste geralmente em um microprocessador para a implementação lógica e controle das comunicações entre os módulos, necessitando de uma

memória para armazenar os resultados das operações lógicas executadas pelo microprocessador. As memórias EPROM ou EEPROM somadas à memória RAM também são necessárias para o programa.

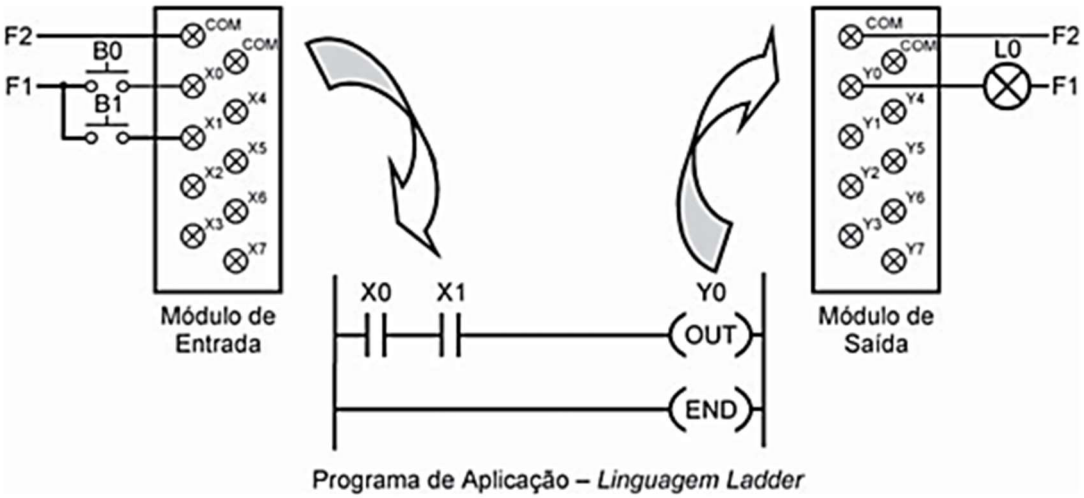
Segundo Georgini (2016, p.49), a programação do PLC é feita com uma Ferramenta de Programação que pode ser um Programador Manual (Terminal de Programação, *Handheld Programmer*) ou um PC com *Software* de Programação específico (Ambiente DOS ou *Windows*), sendo a linguagem *Ladder* (RLL - *Relay Ladder Logic*, Lógica de contatos de Relé), muito popular entre os usuários dos antigos sistemas de controles a relés. Ela é a representação lógica da sequência elétrica de operação, como ilustrado nas Figura 5 e Figura 6.

Figura 5 – Lógica convencional - contatos elétricos.



Fonte: Georgini (2016).

Figura 6 – Implementação da lógica convencional por meio de PLC.



Fonte: Georgini (2016).

2.3 LINGUAGEM DE PROGRAMAÇÃO

De acordo com Silva (2015, p. 38), a norma IEC 61131-3 especifica a sintaxe e a semântica de um conjunto unificado de linguagens de programação para o Controlador Programável. As linguagens em conformidade com a norma são: lista de instruções (*Instruction List*, IL), texto estruturado (*Structured Text*, ST), diagrama de blocos funcionais (*Function Block Diagram*, FBD), diagrama ladder (*Ladder Diagram*, LD) e sequencial gráfico de função (*Sequential Function Chart*, SFC).

2.3.1 Texto Estruturado

De acordo com Silva (2015, p. 50), a linguagem de programação denominada Texto Estruturado (ST) está de acordo com a norma IEC 61131-3. É uma linguagem de alto nível semelhante à linguagem Pascal da norma ISO 7185. Com essa linguagem é possível chamar blocos funcionais, funções e atribuições executando condicionalmente instruções e tarefas de repetição. Sua flexibilidade é de fácil entendimento e proporciona a programadores e desenvolvedores de *softwares* a interpretação do programa do processo industrial. Desenvolvida para controle industrial, a linguagem de programação ST trabalha com “expressões” constituídas de operadores e operandos que retornam um valor quando executados.

Segundo Petruzella (2013, p. 94), o texto estruturado é uma linguagem de texto de alto nível usado primariamente para implementar procedimentos complexos que não podem ser expressos em uma linguagem gráfica, utilizando declarações para definir o que executar.

Silva (2015, p. 50) diz que a linguagem oferece recurso para o desenvolvimento de: declarações; configurações (*configuration*); recurso (*resourse*); unidade de organização de programa (*Program Organization Unit*, POU); tarefa (*task*); tomada de decisões (*IF...THEN... ELSE, FOR...DO* etc.); cálculos diversos; implementação de algoritmos; e utilização de literais

2.3.2 Lista de instrução

Silva (2015, p. 113) diz que a linguagem de programação Lista de Instruções (*Instruction List*, IL) é definida como parte da norma IEC 61131. É uma linguagem

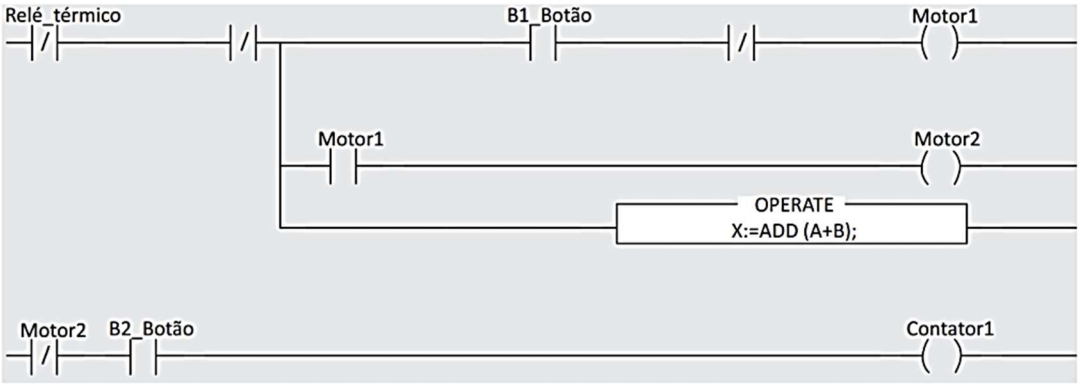
pouco estruturada e de difícil compreensão, pois usa instruções semelhantes às linguagens de programação mnemônicas desenvolvidas para CPs, ou seja, é uma linguagem próxima da forma de processamento feita pela CPU do CP e, por isso, é muito eficiente. Existe semelhança com programação em linguagem *assembly*.

2.3.3 Linguagem *Ladder*

De acordo com Silva (2015, p. 164), a linguagem de programação *ladder* (LD) é uma linguagem gráfica. Foi a primeira linguagem utilizada pelos fabricantes e é a mais aceita pelos programadores de Controlador Lógico Programável (CLP) por ser semelhante ao diagrama de comando elétrico. Essa linguagem é encontrada em quase todos os CLPs que existem no mercado de tecnologia de automação industrial.

Segundo Petruzella (2013, p. 92), a linguagem em diagrama *ladder* é a linguagem mais utilizada para CLP e é projetada para imitar a lógica a relé. O diagrama *ladder* é popular para aqueles que preferem definir as ações de controle em termos de contatos dos relés e das bobinas, além de outras funções como bloco de instruções. Na Figura 7 segue um exemplo da programação.

Figura 7 – Exemplo de programação *ladder*.



Fonte: Silva (2016).

2.3.4 Diagrama de Bloco Funcional

Segundo Petruzella (2013, p. 93), os diagramas de blocos funcionais são similares ao *layout* dos diagramas de blocos elétricos ou eletrônicos utilizados para simplificar sistemas complexos e mostrar a funcionalidade dos blocos, sendo que o

conceito primário por trás deles é o fluxo de dados. Os blocos de funções são ligados entre si para completar um circuito que satisfaz as necessidades do controle, assim os dados circulam pela malha da entrada, passam pelos blocos de funções ou de instruções e seguem para a saída.

De acordo com Silva (2015, p. 223), os blocos são de dois tipos distintos: blocos funcionais e funções. Essas duas classes se diferenciam pelo fato de os blocos funcionais possuírem persistência de dados, ou seja, devem ser instanciados e podem ter diversos ciclos de execução. Já as funções executam sua funcionalidade e depois de encerrada a execução não persiste nenhuma informação além do resultado na saída. Em geral, as funções são nativas do sistema (temporizadores, contadores etc.) mas também podem ser implementadas pelo usuário. Na unidade de organização de programa (POU), o diagrama de blocos funcionais (FBD) e de funções são referidos como blocos com características distintas um do outro.

2.3.5 Sequencial Gráfico de Função

De acordo com Silva (2015, p. 271), trata-se de um método gráfico estruturado para representar um sistema de controle sequencial ou dependentes do tempo, utilizando sequência de etapas e transições.

São características do SFC: facilidade de interpretação, modelagem do sequenciamento, modelagem de funções lógicas e modelagem da concorrência.

2.4 ARDUINO

Plataforma livre criada com o objetivo de tornar mais eficiente o desenvolvimento de protótipos, sendo eles *hardware* e *software*. Possui kits com capacidade de ler dados oriundos de botões, sensores e atuarem em saídas após processamento de dados. Para programar os kits dessa plataforma, o usuário pode utilizar a IDE nativa que possui linguagem própria. Por ser uma plataforma largamente difundida no mercado, destaca-se o grande número de material já existente para consulta que ao passar dos anos vem sofrendo mudanças para acompanhar as novas tendências tecnológicas (ARDUINO, 2021).

De acordo com Stevan Junior (2015, p. 124), essa plataforma é baseada em microcontroladores que são unidades de processamentos com diversos periféricos internos no mesmo circuito integrado, possibilitando que esse dispositivo possa interagir com o ambiente e ter controle sobre atuadores no mesmo por possuir periféricos como memórias (tanto de dados quanto de programa), comunicação serial, conversores analógico-digitais, dispositivos PWM, comparadores, entre outros, além de uma unidade central de processamento. Diferentemente de um microprocessador, um microcontrolador dispõe de praticamente tudo que precisa para operar sozinho, bastando interligar as interfaces com o meio e programá-lo com as funções desejadas.

O Arduino é uma plataforma que permite o desenvolvimento de *hardware* livre, o que permite que diversos fornecedores atendam a suas características padronizadas e desenvolvam placas periféricas chamadas de *Shields*.

O Arduino utiliza a linguagem de programação C++, que é uma linguagem de alto nível e contempla uma enorme variedade de códigos, pois além de seus códigos, pode contar com os códigos da linguagem C.

2.4.1 Funcionamento

O Arduino tem a finalidade de facilitar o controle de sistemas automatizados em ambientes comerciais e domésticos, da mesma forma que o CLP controla e automatiza sistemas industriais.

A placa funciona através de pinos que a conecta a um circuito eletrônico e, através dessa conexão, controla o circuito, enviando e recebendo informações. Para isso é necessário um *software*, baixado e instalado em um computador (KALATEC, 2022).

Os programas desenvolvidos para o Arduino determinam todos os comandos feitos pela placa e a realização das suas tarefas. Ele definirá o início, o tempo de duração, o fim dos processos e as repetições dos comandos. A linguagem para desenvolvimento do programa é criada em códigos semelhantes à linguagem C e C++.

Para serem geradas as programações é necessário a utilização do programa IDE Arduino e conectar a placa no computador através de uma conexão USB, permitindo que os programas desenvolvidos possam ser transferidos e executados.

2.4.2 Entradas e Saídas

Os pinos do Arduino podem ser configurados como entradas ou saídas. É importante observar que a grande maioria dos pinos analógicos do Arduino podem ser configurados e usados exatamente da mesma maneira que os pinos digitais.

Os pinos do Arduino padronizados para entradas não precisam ser declarados explicitamente como entradas (INPUT) com pinMode(). Pinos configurados dessa maneira são considerados em um estado de alta impedância. Os pinos de entrada fazem demandas extremamente pequenas no circuito que estão amostrando, equivalente a um resistor em série de 100 mega ohm na frente do pino. Isso significa que é necessária pouca corrente para mover o pino de entrada de um estado para outro e tornar os pinos úteis para tarefas como implementar um sensor de toque capacitivo, ler um LED como um fotodiodo ou ler um sensor analógico.

Pinos configurados como saídas (OUTPUT) com pinMode() são considerados em estado de baixa impedância. Isso significa que eles podem fornecer uma quantidade substancial de corrente para outros circuitos. Os pinos podem fornecer (corrente positiva) ou dissipar (corrente negativa) até 40mA (miliamperes) de corrente para outros dispositivos ou circuitos. Isso é corrente suficiente para acender um LED ou executar muitos sensores, por exemplo, acionar relés, solenoides ou motores.

2.4.3 Programação

A segmentação de código em funções permite que um programador crie partes modulares de código que executam uma tarefa definida e, em seguida, retornem à área de código da qual a função foi "chamada". O caso típico de criação de uma função é quando é necessário executar a mesma ação várias vezes em um programa (ARDUINO, 2021).

Para programadores acostumados a usar BASIC, funções no Arduino fornecem e estendem a utilidade de usar sub-rotinas (GOSUB em BASIC).

A padronização de fragmentos de código em funções tem várias vantagens:

- As funções ajudam o programador a se manter organizado. Muitas vezes, isso ajuda a conceituar o programa.

- As funções codificam uma ação em um lugar para que a função só precise ser pensada e depurada uma vez.
- Isso também reduz as chances de erros na modificação, se o código precisar ser alterado.

As funções tornam todo o esboço menor e mais compacto porque as seções de código são reutilizadas muitas vezes. Eles facilitam a reutilização de código em outros programas tornando-o mais modular. O uso de funções também torna o código mais legível. Existem duas funções necessárias em um esboço do Arduino, setup() e loop(). Outras funções devem ser criadas fora dos colchetes dessas duas funções.

2.5 SISTEMAS SUPERVISÓRIOS

De acordo com Moraes (2007, p. 117), sistemas supervisórios são sistemas digitais de monitoração e operação da planta, que gerenciam variáveis de processo. Estas são atualizadas continuamente e podem ser guardadas em bancos de dados locais ou remotos para fins de registro histórico.

Em todos os processos industriais existentes há dois tipos básicos de variáveis, digitais e analógicos:

- Digitais: Quando as variáveis podem ser interpretadas por apenas dois estados discretos. Exemplo: motor ligado ou motor desligado; lâmpada acesa ou apagada; motor em falha ou normal etc.
- Analógicas: Quando as variáveis assumem valores que percorrem uma determinada faixa estabelecida. Exemplo: velocidade de um carro; temperatura de um forno; corrente de um motor.

Por outro lado, os sistemas supervisórios podem ser classificados basicamente quanto a complexidade, robustez e número de entradas e saídas monitoradas. Os dois grandes grupos atualmente conhecidos são:

- HMI / IHM (*Human Machine Interface* / Interface Homem Máquina).
- SCADA (*Supervisory Control and Data Acquisition* - Aquisição de Dados e Controle do Supervisório).

2.5.1 Interface Homem Máquina (IHM)

De acordo com Moraes (2007, p. 119), uma IHM é um *hardware* industrial composto normalmente por uma tela de cristal líquido e um conjunto de teclas para navegação ou inserção de dados que utiliza um *software* proprietário para sua programação.

Há várias aplicações e utilizações para uma IHM:

- Visualização de alarmes gerados por alguma condição anormal do sistema;
- Visualização de dados dos motores e/ou equipamentos de uma linha de produção;
- Visualização de dados de processo da máquina;
- Alteração de parâmetros do processo;
- Operação em modo manual de componentes da máquina;
- Alteração de configurações de equipamentos.

Temos exemplos de máquinas operatrizes que contém IHMs e são chamadas de CNC (Comando Numérico Computadorizado).

Em máquinas automatizadas com o emprego de CNC é imprescindível o uso de IHMs dedicadas, pois existe uma necessidade real de que o operador interaja com a máquina diretamente nas seguintes situações:

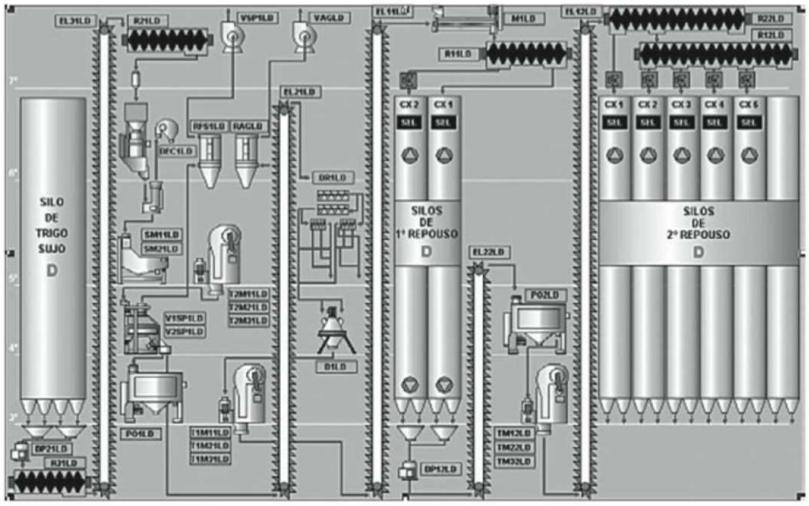
- Referenciamento dos eixos;
- Ajuste de ferramentas;
- Carga de programa de uma peça a ser usinada;
- Acompanhamento da execução do programa enquanto a máquina está usinando a peça;
- Parametrização dos acionamentos dos servomotores;
- Ajuste de velocidade de avanço das ferramentas sobre a peça;
- Visualização de alarmes;
- Tela de manutenção em que pessoas preparadas podem intervir no funcionamento da máquina;
- Realização de movimentos manuais.

2.5.2 Aquisição de Dados e Controle do Supervisório (SCADA)

Segundo Moraes (2007, p. 120), o sistema SCADA foi criado para supervisão e controle de quantidades elevadas de variáveis de entrada e saída digitais e analógicas distribuídas (Figura 8).

A sua aplicação tem sido implementada tanto na área civil quanto na industrial a integridade física das pessoas, equipamentos e produção, consistindo muitas vezes em sistemas redundantes de *hardware* e meio físico (canal de informação), permitindo pronta identificação de falhas. Alguns sistemas permitem a troca a quente do *hardware* danificado, facilitando o reparo sem necessitar de parada do sistema (MORAES, 2007, p. 120).

Figura 8 – Exemplo de uma tela de supervisório para controle e monitoramento.



Fonte: Moraes (2007).

2.5.2.1 Arquiteturas dos sistemas supervisórios

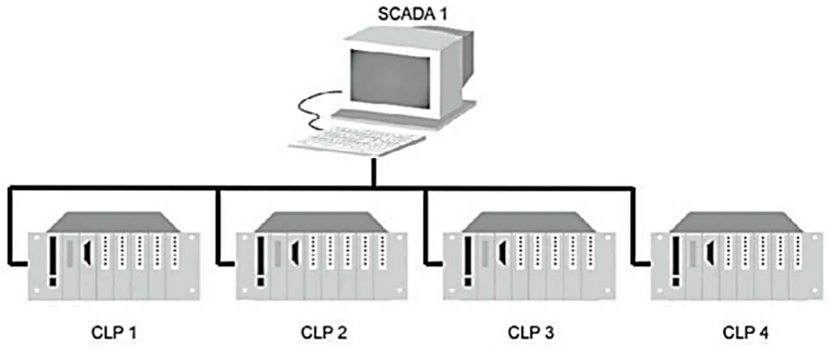
De acordo com Moraes (2007, p. 121), os sistemas de supervisão e controle comumente chamados de sistemas SCADA são sistemas configuráveis, destinados à supervisão, ao controle e à aquisição de dados de plantas industriais, apresentando custo menor que os SDCD (Sistemas Digitais de Controle Distribuído) e, por essa razão, sendo muito populares nas indústrias.

Segundo Moraes (2007, p. 121), esses sistemas possibilitam configurar os arquivos de alarmes e eventos, além de relatórios e interfaces para controle de

receitas e funções avançadas através da escrita de scripts, que são trechos de programas que permitem ampliar as funcionalidades inerentes do produto.

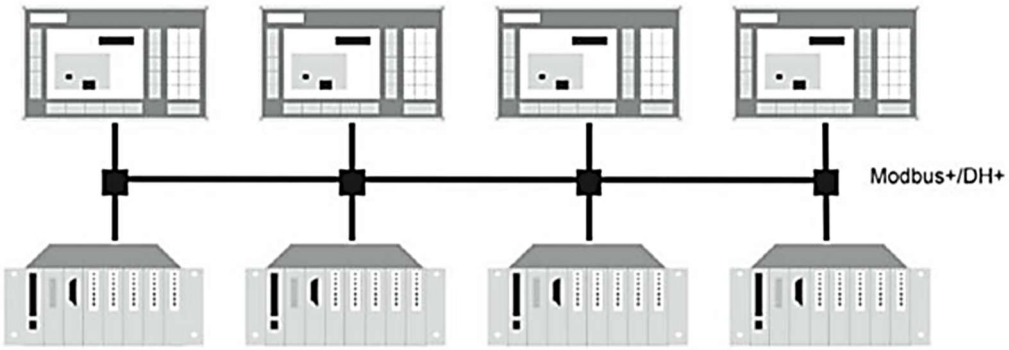
Na Figura 9 é apresentado um sistema SCADA fazendo a aquisição de dados de quatro CLPs. Na Figura 10 é apresentado um sistema com protocolo *Modbus*.

Figura 9 – Sistema SCADA fazendo a aquisição de dados de quatro CLPs.



Fonte: Moraes (2007).

Figura 10 – Exemplo de Arquitetura com protocolo *Modbus*+ / DH+.



Fonte: Moraes (2007).

De acordo com Moraes (2007, p. 121), os sistemas SCADA utilizam dois modos de comunicação: comunicação por *polling* e comunicação por interrupção, normalmente designada por *Report by Exception*.

2.5.2.1.1 Comunicação por consulta (polling)

De acordo com Moraes (2007, p. 122), neste modo de comunicação designado por mestre/escravo a estação central (*Master*) tem o controle absoluto das comunicações, efetuando sequencialmente a leitura dos dados de cada estação

remota (*Slave*), que apenas responde à estação central após a recepção de um pedido, ou seja, em *half-duplex*.

Vantagens:

- Simplicidade no processo de aquisição de dados;
- Inexistência de colisões no tráfego da rede;
- Permite, devido ao seu caráter determinístico, calcular a largura de banda utilizada pelas comunicações e garantir tempos de resposta;
- Facilidade na detecção de falhas de ligação; permite o uso de estações remotas não inteligentes.

Desvantagens:

- Incapacidade, por parte das estações remotas, de comunicar situações que requeiram trata mento imediato por parte da estação central;
- O aumento do número de estações remotas tem impactos negativos no tempo de espera;

A comunicação entre estações remotas tem obrigatoriamente que passar pela estação central.

2.5.2.1.2 Comunicação por interrupção (report by exception)

De acordo com Moraes (2007, p. 122), neste modo de comunicação, a estação remota monitora os seus valores de entrada e, quando detecta alterações significativas ou valores que ultrapassem os limites definidos, inicia a comunicação com a estação central e a consequente transferência de dados.

Vantagens:

- Evita a transferência de informação desnecessária, diminuindo o tráfego na rede;
- Permite uma rápida detecção de informação urgente;
- Permite comunicação entre estações remotas, (escravo para escravo).

Desvantagens:

- A estação central apenas consegue detectar falhas na ligação após um determinado período, ou seja, quando efetua *polling* no sistema;
- É necessária a existência de ação por parte do operador para obter os valores atualizados.

2.5.2.2 Protocolo *MODBUS*

O Protocolo *MODBUS* é um protocolo aberto, desenvolvido pela *Modicon Industrial Automation System,* atualmente Schneider Electric. Padronizado em 1979, é um dos protocolos mais utilizados atualmente, sendo de fácil operação e manutenção, tornando-o uma solução de baixo custo e vasta aplicabilidade.

De acordo com Strack (2011), o padrão *MODBUS* define um protocolo de mensagens na camada de aplicação, posicionando na camada 7 do modelo OSI que proporciona comunicação cliente/servidor entre dispositivos conectados em diferentes tipos de barramentos ou redes. Esse padrão também especifica um protocolo de comunicação serial para requisições entre um mestre e um ou mais escravos. A principal função do *Modbus* é ordenar a ação que ele deve tomar.

2.5.2.2.1 Modo de transmissão

De acordo com Souza (2008), o modo de transmissão define o conteúdo de bit da mensagem a ser transmitida na rede e como a informação da mensagem será empacotada na mensagem e descompactada.

O padrão MODBUS emprega os dois modos de transmissão, ASCII *Mode* e RTU *Mode*.

No modo ASCII (*American Standard Code for Information Interchange*), em cada byte de caractere em uma mensagem é enviado dois caracteres sem geração de erros.

No modo RTU (*Remote Terminal Unit*), cada mensagem de 8 bits contém dois caracteres hexadecimais de 4 bits.

2.6 CONCEITOS HIDRÁULICOS

Para auxiliar no entendimento do sistema de forma detalhada, será apresentada as definições de alguns conceitos de hidráulica. Este capítulo tem como finalidade, trazer esclarecimento sobre os conceitos e permitir a compreensão dos tópicos abordados.

2.6.1 Automatismo hidráulico

De acordo com Fialho (2015, p. 09), um projeto de automação que requer movimentação de elementos orgânicos de máquinas, como translação, rotação, levantamento de elevadas cargas e controle de velocidade, pode ser feito utilizando os princípios da pneumática, em que o ar comprimido é a principal fonte de energia, ou os princípios da hidráulica, em que o fluido utilizado, mais especificamente o óleo, também sob pressão, é a principal fonte de energia.

2.6.1.1 Conceitos

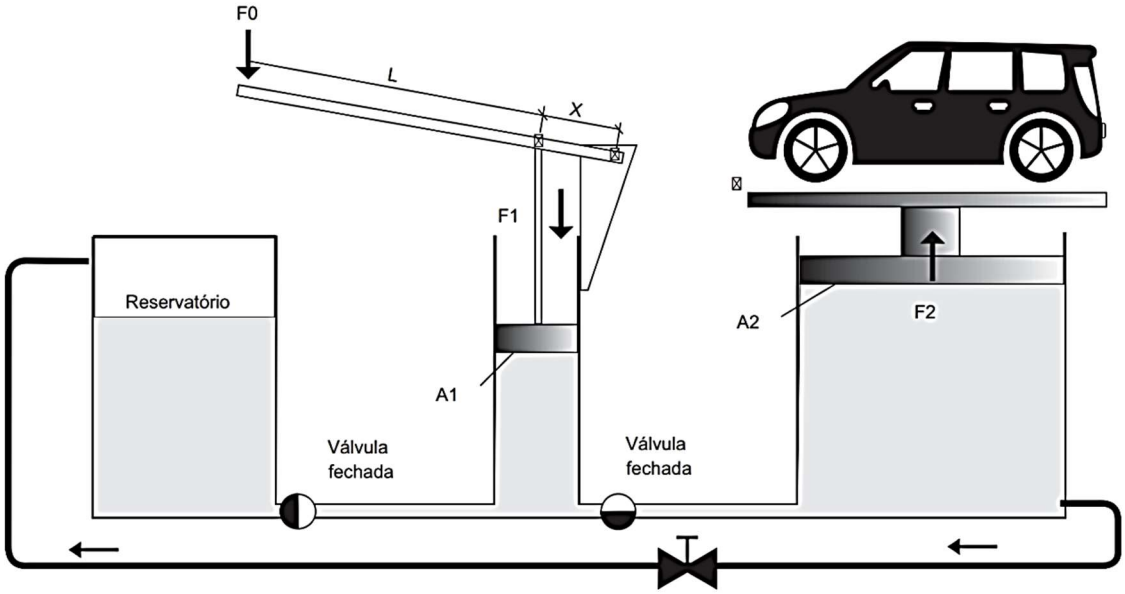
De acordo com Filho (2018, p. 15), a hidráulica é um sistema que utiliza um fluido como meio transmissor de energia para a execução de trabalho útil. Como exemplo, pode-se citar que ela utiliza óleo hidráulico sob pressão para acionamentos em máquinas e equipamentos estacionários e móveis. Sistemas hidráulicos são muito usados na indústria, na construção civil, em veículos e até na aviação.

2.6.1.2 Transmissão de energia hidráulica

De acordo com Fialho (2018, p. 17), a tecnologia dos fluidos requer o entendimento das características mecânicas desses fluidos, isto é, a sua capacidade de transmitir pressão ou energia. A denominação hidromecânica é mais empregada em óleo-hidráulica.

De acordo com Fialho (2015, p. 11), a hidráulica é responsável pelo conhecimento das leis que regem o transporte, a conversão de energia, a regulagem e o controle do fluido, agindo sobre suas variáveis (pressão, vazão, temperatura, viscosidade etc.). Nesse caso, ao observarmos a Figura 11, veremos que a energia mecânica inicial gerada pela força F0 é inicialmente amplificada para F1 pelo sistema de alavanca L e convertida em energia hidráulica, propagando-se pelo fluido até encontrar a plataforma A2 quando se converte em energia mecânica a ser entregue por meio da força F2.

Figura 11 – Princípio da alavanca hidráulica.



Fonte: Fialho (2015, p. 12).

A mensuração de F1 é dada pela relação:

$$F1 = \frac{F0.L}{X} \quad (1)$$

Lembrando o conceito de pressão hidrostática, teremos a seguinte relação válida:

$$P = \frac{F1}{A1} = \frac{F2}{A2} \quad (2)$$

2.6.2 Bombas hidráulicas

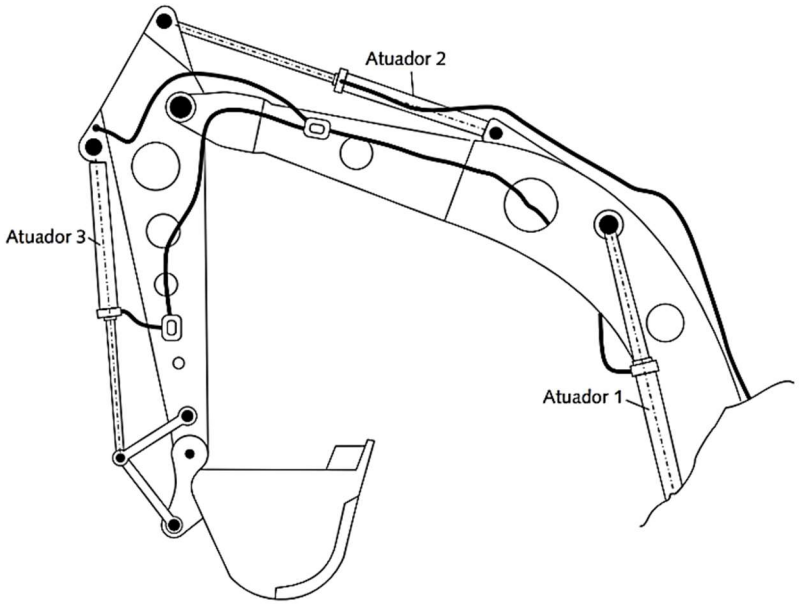
De acordo com Filho (2018, p. 41), a bomba é responsável pela geração de vazão dentro do sistema hidráulico, sendo, portanto, responsável pelo acionamento dos atuadores. Esse equipamento é utilizado para converter energia mecânica em hidráulica.

Entretanto, de acordo com Fialho (2015, p. 29), todas as bombas são classificadas em duas categorias básicas: hidrodinâmica e hidrostática. Porém, elas podem ainda ser de deslocamento fixo, isto é, quando o deslocamento se dá por meio da rotação da própria bomba e não pode ser ajustado, e de deslocamento variável, resultando em uma construção mais complexa que permite o ajuste do deslocamento.

2.6.3 Atuadores hidráulicos

De acordo com Fialho (2015, p. 40), atuadores hidráulicos são automatismos destinados a guiar partes de um mecanismo – articulado ou não – em trajetórias específicas, produzindo deslocamentos lineares ou angulares com acurada precisão de velocidade e posição, sendo, ainda, agente de transferência de força ou torque por meio da transmissão da energia hidráulica. A Figura *12* ilustra o braço articulado de uma escavadeira, exibindo três atuadores lineares responsáveis pelos diversos posicionamentos angulares que o mecanismo é capaz de executar.

Figura 12 – Braço articulado.



Fonte: Fialho (2015, p. 40).

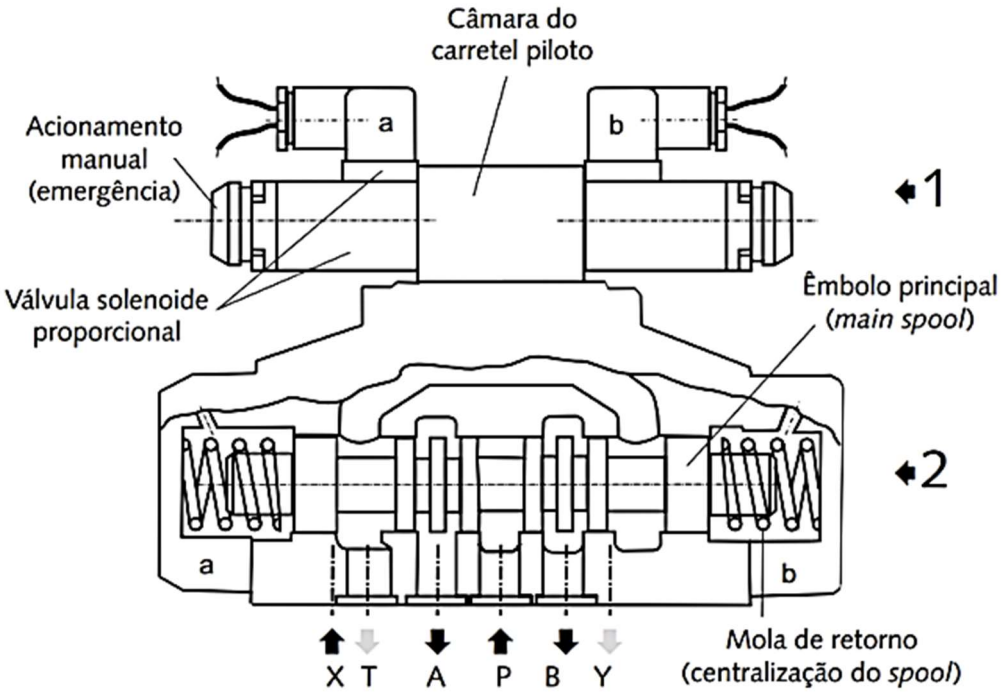
2.6.4 Válvulas proporcionais

De acordo com Silva (2015), a Válvula Proporcional é um componente de controle que geralmente é acionada por solenoides proporcionais. O sinal elétrico de entrada é convertido em sinal proporcional de saída hidráulico de vazão e ou pressão proporcional. (LINSINGEN, 2001).

De acordo com Fialho (2019, p. 137), nessas válvulas, o êmbolo pode ocupar infinitas posições intermediárias, controlando também pressão e vazão, pois o orifício de passagem de óleo aumenta ou diminui conforme a posição do êmbolo. Os solenoides proporcionais são, normalmente, de corrente contínua, e trabalham

variando a atuação conforme a variação da corrente de entrada. A Figura *13* exibe um modelo simplificado elaborado por este autor com base na geometria de uma válvula comercial de um conhecido fabricante.

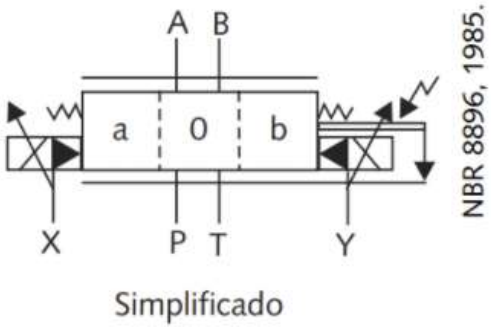
Figura 13 – Exemplo de modelo simplificado e modificado pelo autor de uma válvula distribuidora proporcional contendo uma composição de (1) válvula piloto e de (2) válvula principal, a partir de catálogo de um grande fabricante.



Fonte: Fialho (2019, p. 137).

A Figura 14 apresenta a simbologia simplificada de uma válvula direcional do tipo proporcional integrada com controle de pressão.

Figura 14 – Simbologia simplificada de uma válvula proporcional atuada por solenoides.



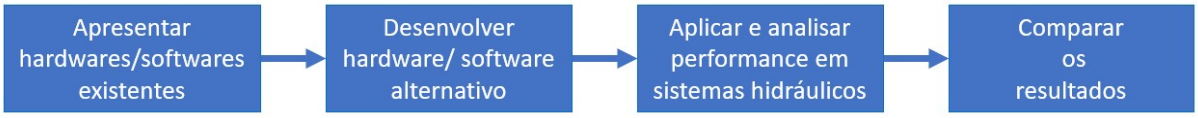
Fonte: Fialho (2019, p. 137).

3. METODOLOGIA

Neste capítulo será detalhada a metodologia aplicada para a execução deste trabalho, cujos objetivos são: desenvolver um *hardware* e *software* para controle de sistemas hidráulicos, possibilitar uma alternativa de baixo custo e de alto desempenho e comparar a proposta com produtos já desenvolvidos para esta finalidade.

É apresentado na Figura 15 um fluxograma que ilustra as etapas do planejamento realizado e executado.

Figura 15 – Fluxograma das atividades.



Fonte: Elaborado pelo autor.

A proposta de automatização dos sistemas hidráulicos será desenvolvida com um Arduino Uno, que possibilita a programação de controle proporcional, booleano e, permite a leitura de pressão, velocidade e vazão.

Em comparação com CLP's disponíveis para o controle de sistemas hidráulicos, o Arduino proporcionará um diferencial considerável de custo e possibilitará algumas propostas de controle dos sistemas propostos, também tem uma linguagem de programação mais limitada e de fácil compreensão. A principal desvantagem em comparação com os CLP's específicos, são o número de pinos de entradas e saídas.

Os pinos limitam a quantidade de componentes que podem ser inseridos no sistema controlado com o Arduino.

3.1 PROPOSTA DE APLICAÇÃO PARA SISTEMAS HIDRÁULICOS

Serão desenvolvidos diagramas hidráulicos com vasta aplicabilidade e apresentadas aplicações onde poderão ser inseridos o produto a ser desenvolvido.

A proposta inicial a ser apresentada, possibilitará o controle da velocidade e da posição de atuadores hidráulicos através do controle de vazão. Isso se dará pela utilização da válvula direcional com acionamento proporcional e do sensor de

deslocamento do atuador. Esta aplicação poderá ser utilizada em sistemas que necessitem de controle preciso de atuadores, como, por exemplo, uma garra florestal que necessita de alto controle para pegar e empilhar as toras no caminhão.

A segunda proposta a ser desenvolvida contemplará um sistema com controle de pressão através de uma válvula de alívio com acionamento proporcional. Para checagem e controle desta pressão será utilizado um transdutor de pressão. Esta aplicação poderá ser implementada em sistemas que utilizem bombas de vazão fixa com a pressão de trabalho contínua. Este sistema terá um compensador de pressão que, ao ser acionado, atingirá a pressão de trabalho, e, ao ser mantido em repouso, regulará a válvula a uma pressão próxima a zero. Portanto, a proposta possibilitará que haja redução do consumo de energia do sistema hidráulico, e da geração de calor.

A terceira proposta que será desenvolvida terá como objetivo apresentar um sistema de controle de pressão e vazão, que possibilitará o ajuste da força e da velocidade. Esta proposta poderá ser implementada em diversos projetos, como por exemplo, um braço de robô de soldagem que necessita de controle de posição, de velocidade e de força elevados.

3.2 INFORMAÇÕES DE *HARDWARE* PARA COMPARAÇÃO

A proposta do trabalho consiste em desenvolver uma solução alternativa que possibilitará a automatização de sistemas hidráulicos em maquinário de pequeno porte a menores custos, visto que, os já disponíveis no mercado possuem elevado custo, por exemplo, o VT-HACD e o VT-HMC.

O VT-HACD, foi o escolhido para apresentação de parâmetros de funcionamento, visto que, atende as necessidades de controle dos sistemas hidráulicos e possui um custo consideravelmente menor quando comparado ao VT-HMC. A bancada hidráulica da UNISINOS contempla a utilização do *hardware* VT-HACD.

O VT-HACD e o VT-HMC são *hardwares* desenvolvidos para automação de sistemas hidráulicos de alto desempenho, onde é possível controlar pressão e vazão do sistema, deslocamento dos atuadores e acionamento das bobinas.

O VT-HMC é um controlador digital com controle de eixo integrado e programações funcionais, tais como, controle de posição, de força, de pressão, de

velocidade e de torque. É um *hardware* de controle de sistemas hidráulicos de alto desempenho que contempla seus programas configurados em sua memória.

O *hardware* VT-HACD possibilita controle de loops para controlador PIDT1 (controle proporcional, integral, derivativo) e de *feedback* de estado (resposta de ações executadas no programa), além de controle de malha fechada (por exemplo, controle de posição e pressão) e padronização de sinais. Contém entradas para sistemas de medição de posição digitais: 6 entradas analógicas (tensão 0 a 10 V, e corrente 4 a 20 mA) comutáveis via *software*, 3 saídas analógicas, 8 entradas digitais e 7 saídas digitais. O *hardware* também apresenta 32 blocos com valores de comandos, de velocidades, de parâmetros do controlador, de correção de curvas, de lógica de velocidade residual e de ponto zero de correção.

O VT-HACD pode ser aplicado em diversas soluções hidráulicas, como por exemplo, injetoras de plástico, prensas, máquinas agrícolas e florestais, entre outros sistemas em que se objetiva ter controle e precisão automatizados. O *hardware* também apresenta 32 blocos com valores de comandos, velocidades e parâmetros do controlador, correção de curvas, lógica de velocidade residual e ponto zero de correção.

O grande diferencial de automatizar o sistema hidráulico é a possibilidade de controlar força, pressão, vazão, velocidade com elevada precisão e de apresentar sistemas alternativos que possam elevar a vida útil dos componentes.

A Figura 16 mostra o VT-HACD, que, foi o *hardware* disponível para comparação e testes de funcionamento do sistema desenvolvido com o Arduino.

Figura 16 – *Hardware* VT-HACD.

Fonte: https://www.boschrexroth.com/.

3.3 DETERMINAÇÃO DE *HARDWARE* PARA O PROJETO

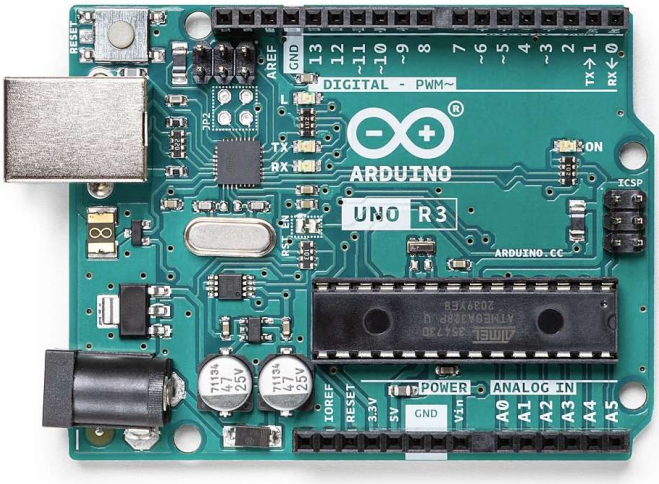
Atualmente os *hardwares* de controle de sistemas hidráulicos apresentam elevado custo, tornando-o um investimento de difícil retorno dependendo da aplicação que o sistema hidráulico apresenta.

A proposta de desenvolver um *hardware* de baixo custo para possibilitar o controle de desempenho de um sistema hidráulico parte inicialmente da escolha do tipo de *hardware*, tendo em vista que o custeio de um CLP para esta aplicação se torna, muitas vezes, inviável.

Para o desenvolvimento de *softwares* de controle hidráulico está sendo proposta a utilização do *hardware* Arduino. Este é utilizado como plataforma de prototipagem para programação de diversos tipos de componentes, contemplando entradas e saídas analógicas/digitais e aceitando a linguagem de programação C++. O *hardware* Arduino também pode receber sinais de transdutores de pressão, de sensores de posição e de velocidade, e pode controlar bobinas liga/desliga, bobinas proporcionais, entre outros. Isso dependerá somente do programa desenvolvido no *software*. O modelo de Arduino escolhido irá desenvolver propostas alternativas ao VT-HACD. Com base nas funções hidráulicas propostas, a listagem de funções determinará a quantidade de pinos que deverão ser nomeados para o projeto.

O *hardware* escolhido para o desenvolvimento do trabalho foi o Arduino Uno (Figura 17).

Figura 17 – Arduino Uno.



Fonte: Arduino Uno Rev3 — Arduino Online Shop

O Arduino Uno será o microcontrolador escolhido para o desenvolvimento do *hardware* por ser uma ferramenta de baixo custo, versátil e de vasta aplicabilidade. Ele contempla 6 entradas e 6 saídas analógicas e, também, dispõe de 14 pinos digitais, que o tornam apto para o desenvolvimento das aplicações a serem apresentadas.

3.3.1 Escolha do Arduino

Para o desenvolvimento do projeto, será utilizado o Arduino Uno. A escolha do *hardware* está vinculada principalmente no baixo custo que ele apresenta em relação aos CLP's específicos para controle de sistemas hidráulicos propostos.

O Arduino possibilita o desenvolvimento de programas com controles proporcionais e booleanos, possibilitando a utilização de sensores para leitura de dados, e controlar diversos componentes, assim como nos CLP's.

As vantagens do Arduino em relação ao CLP no desenvolvimento do projeto, são principalmente a diferença de custo entre eles, e é uma ferramenta de fácil manuseio com vasta biblioteca de programas desenvolvidos por usuários da plataforma, além de possibilitar suporte técnico.

Para o desenvolvimento das propostas de sistemas hidráulicos, o Arduino Uno possibilitará o desenvolvimento da automação dos sistemas.

3.4 DESENVOLVIMENTO INICIAL DO *SOFTWARE*

O *software* visa controlar e elevar o desempenho de sistemas hidráulicos diversos, através do controle de pressão e vazão com válvulas de acionamento proporcional e sensores de pressão e movimento.

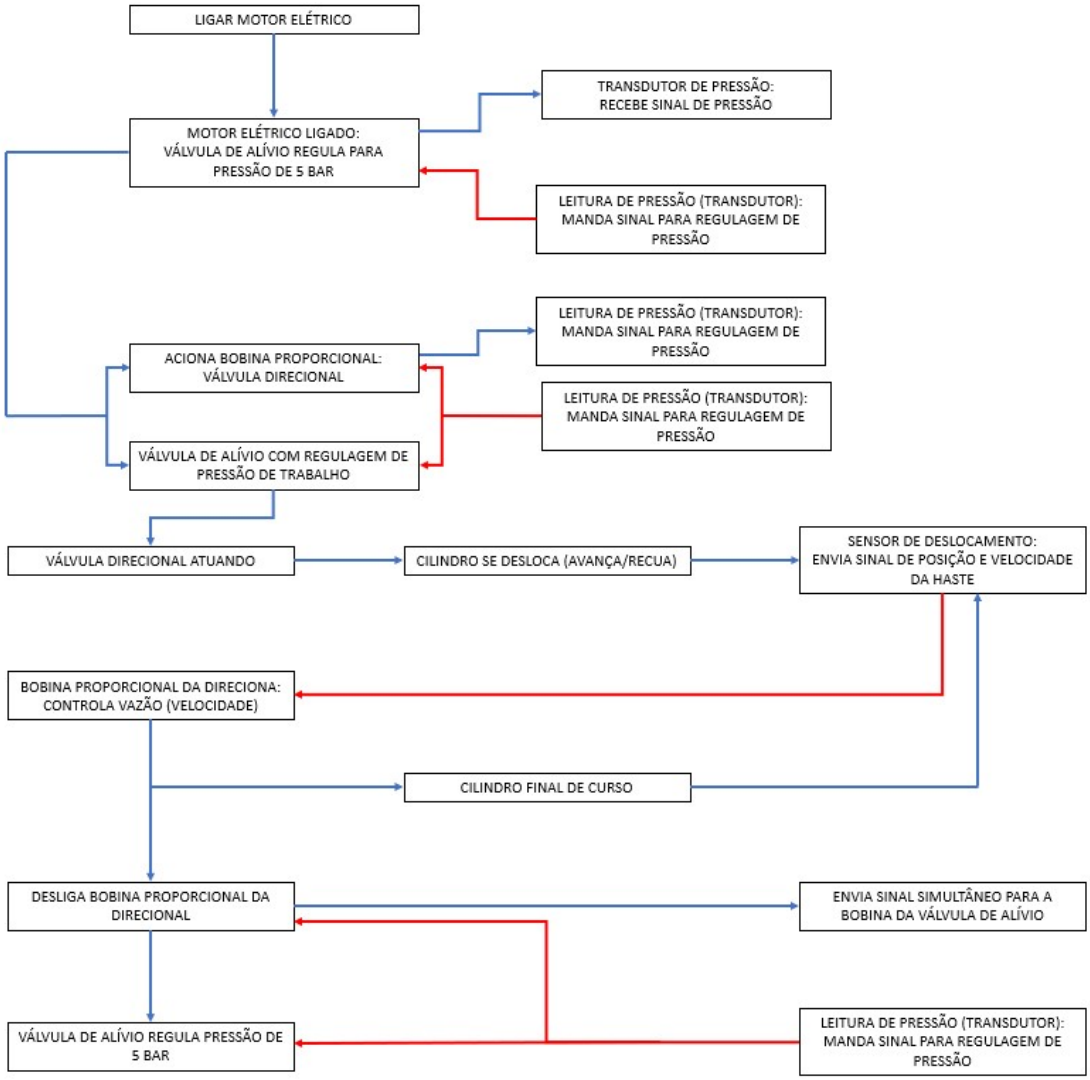
Inicialmente será necessário listar os componentes disponíveis na estrutura da universidade para desenvolvimento das propostas. O *software* completo utilizará válvula de alívio com acionamento proporcional para controle de pressão, válvula direcional com acionamento proporcional para controle de vazão, transdutor de pressão e sensor de deslocamento para o cilindro hidráulico.

A Figura 18 apresenta a lógica de funcionamento do sistema com controle de vazão e pressão, através de um diagrama de blocos que servirá de base para o desenvolvimento da programação para o projeto.

Após o desenvolvimento da lógica de funcionamento completa utilizando todos os componentes será possível montar propostas que são usuais em sistemas hidráulicos sofisticados.

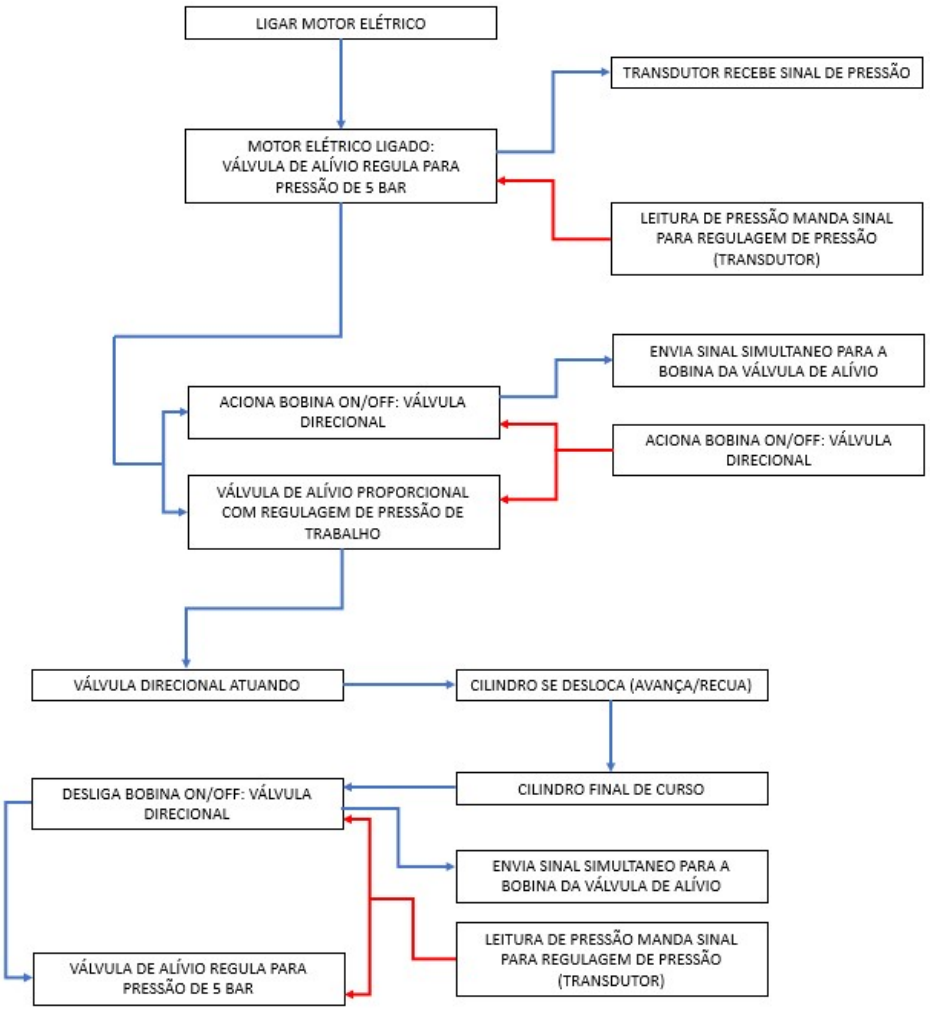
A Figura 19 apresenta a segunda proposta de *software*, que utiliza controle de pressão. O sistema usará válvula direcional liga/desliga, transdutor de pressão e válvula de alívio proporcional.

Figura 18 – Diagrama lógico completo.



Fonte: Elaborado pelo autor.

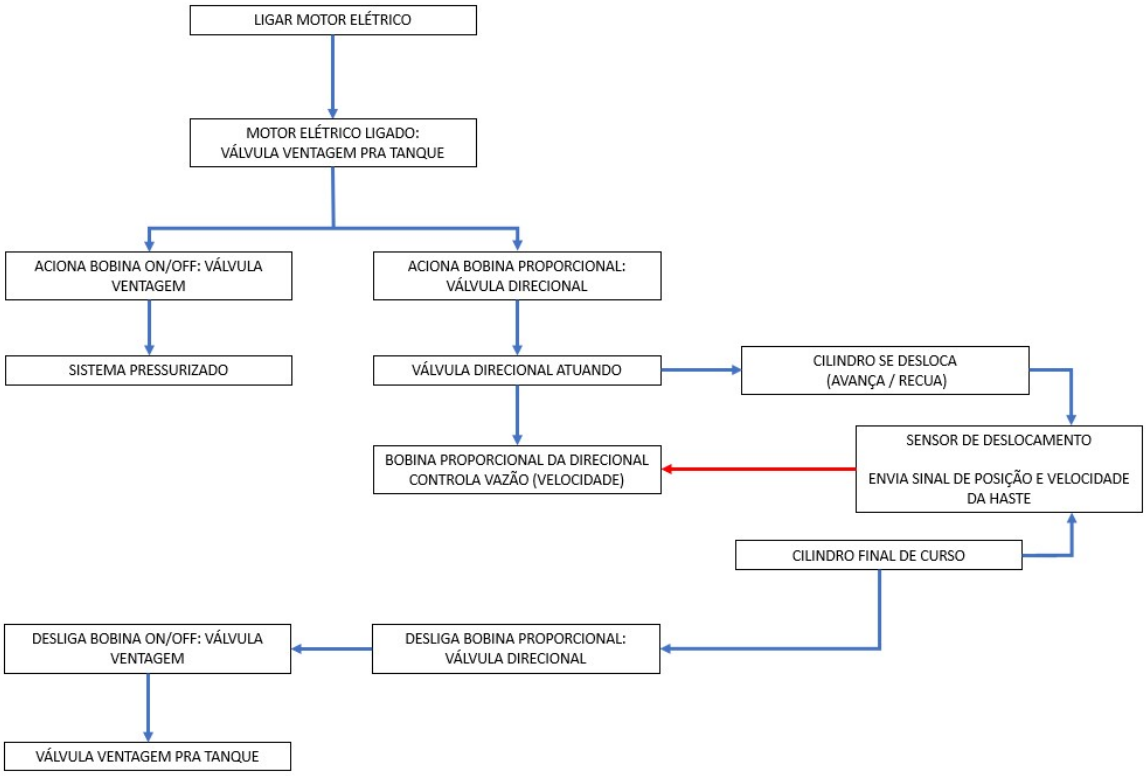
Figura 19 – Diagrama lógico de controle de pressão.



Fonte: Elaborado pelo autor.

A Figura 20 apresenta a terceira proposta de *software*, que controla a vazão no sistema, na qual impacta diretamente na velocidade de deslocamento do atuador. Neste modelo deverá ser utilizado uma válvula direcional proporcional e um sensor de deslocamento do cilindro hidráulico. Esta proposta de programa pode também trabalhar com acionamento de motores hidráulicos, necessitando apenas da troca do sensor de deslocamento por um sensor de rotação.

Figura 20 – Diagrama lógico de controle de vazão.



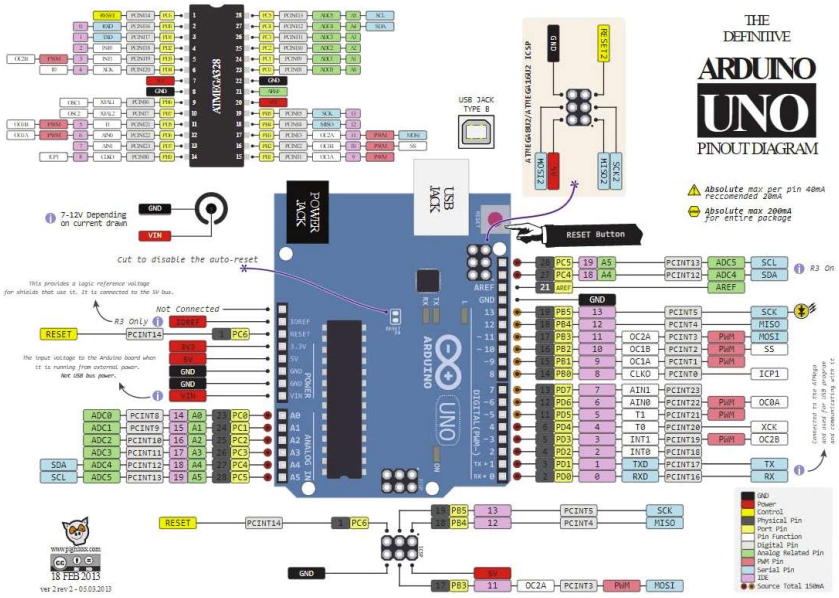
Fonte: Elaborado pelo autor.

3.4.1 Determinação de pinagem do *hardware*

Para a sequência do desenvolvimento do *software* será necessário determinar e padronizar os pinos analógicos e digitais do Arduino. Os pinos de entrada recebem sinais dos sensores (transdutor, sensor de deslocamento, sinais de liga e desliga), já a pinagem de saída envia sinais para os componentes que se deseja controlar (bobinas proporcionais, bobinas liga/desliga, liga e desliga do sistema).

Para determinar os pinos, será utilizado o datasheet do Arduino Uno como suporte na determinação de cada ligação, possibilitando que cada pino fique fixo a sua determinada função e tornando o *software* padronizado (Figura 21).

Figura 21 – Datasheet Aduino Uno.



Fonte: 85.jpg (961×680) (arduinoportugal.pt).

Após consulta do datasheet todos os pinos necessários para as funções determinadas para o desenvolvimento do controle do sistema hidráulico estarão organizados, conforme a Figura 22.

Figura 22 – Configuração de conectores.

CONFIGURAÇÃO DE PINAGEM SOFTWARE HIDRÁULICA			
ENTRADAS	**PINOS**	**FUNÇÃO**	**TIPO**
2	INT0	ENTR_01	DIGITAL
4	T0	ENTR_02	DIGITAL
A0	ADC0	ENTR_ANALOG_01	ANALÓGICA
A1	ADC1	ENTR_ANALOG_02	ANALÓGICA
A2	ADC2	ENTR_ANALOG_03	ANALÓGICA
A3	ADC3	ENTR_ANALOG_04	ANALÓGICA
SAÍDAS	**PINOS**	**FUNÇÃO**	**TIPO**
10	OC1B	SAIDA_01	DIGITAL
11	OC2A	SAIDA_02	DIGITAL
3	INT1	SAIDA_ANALOG_01	ANALÓGICA
5	T1	SAIDA_ANALOG_02	ANALÓGICA
6	AIN0	SAIDA_ANALOG_03	ANALÓGICA
DISPONÍVEIS	**PINOS**	**FUNÇÃO**	**TIPO**
0	RXD		DIGITAL
1	TXD		DIGITAL
3	INT1		DIGITAL
6	AIN0		DIGITAL
7	AIN1		DIGITAL
8	CLK0		DIGITAL
9	OC1A		DIGITAL
A5	ADC5		ANALÓGICA
A4	ADC4		ANALÓGICA

Fonte: Elaborado pelo autor.

3.4.2 Linguagem de programação

Para o desenvolvimento do *software*, os programas serão elaborados através da linguagem C++, utilizada para programação do Arduino.

Para compilar o programa será utilizado o ambiente de desenvolvimento IDE Arduino. Este é um aplicativo de computador que possui um compilador integrado, onde se pode escrever e executar o programa.

3.4.3 Interface de funcionamento

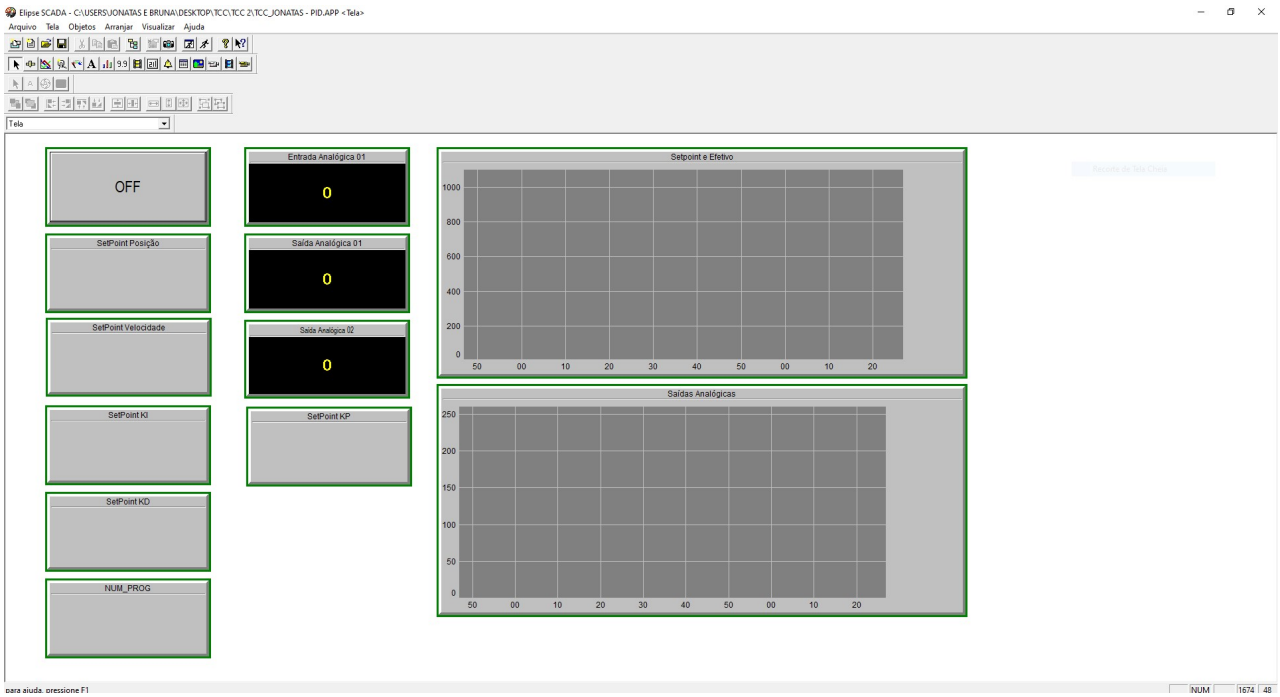
A interface de controle e acionamento dos programas a serem desenvolvidos serão montadas após a estruturação dos programas. Para desenvolvimento dos supervisórios de controle e monitoramento será utilizado o programa Elipse SCADA, visto ser um programa supervisório de fácil linguagem e totalmente intercambiável com o programa do Arduino.

A comunicação entre os programas Elipse SCADA e o Arduino funciona através da rede *Modbus* em comunicação serial, possibilitando resposta entre hardware e interface simultaneamente.

A finalidade das interfaces a serem desenvolvidas é monitorar os dados de pressão, de velocidade, de desempenho dos solenoides e de liga/desliga, podendo, também, informar falha caso seja desejável.

A Figura 23 apresenta a interface de trabalho do programa Elipse SCADA, com algumas opções inseridas como demonstração da ferramenta.

Figura 23 – Interface do Elipse SCADA.



Fonte: Elaborado pelo autor.

4. DESENVOLVIMENTO DAS APLICAÇÕES

A seguir serão apresentados os resultados desenvolvidos para o funcionamento dos programas para controle dos sistemas hidráulicos propostos.

Serão apresentadas as configurações para o desenvolvimento das interfaces de controle e monitoramento no programa Elipse SCADA, as interfaces, as programações para o Arduino, o *hardware* e os diagramas hidráulicos.

4.1 INTERFACE DE CONTROLE

Para o desenvolvimento da interface de controle, foi desenvolvido uma interface de controle através do programa Elipse. O programa Elipse SCADA foi configurado para possibilitar a comunicação com o Arduino.

O supervisório possibilita que os programas desenvolvidos possam ser configurados através da interface de controle e monitorados através de gráficos e exibição de valores como velocidade e pressão.

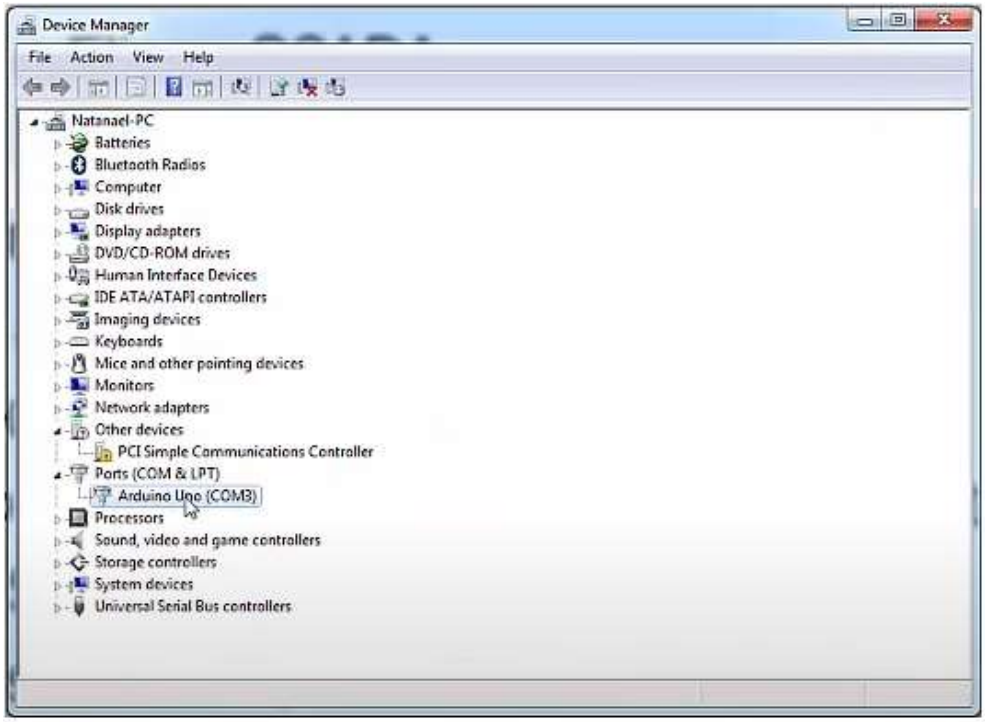
Para que o Elipse consiga trabalhar em conjunto com o Arduino será necessário que o programa seja configurado. Esta configuração será detalhada em no “passo a passo” a seguir:

Inicialmente é necessário que o Arduino esteja conectado ao computador e isso poderá ser identificado ao abrir o gerenciador de dispositivos. Como parâmetro para o desenvolvimento da proposta a escolha das portas (COM e LPT) foi selecionado a COM3, conforme Figura 24.

No Arduino precisa que seja instalado uma biblioteca *Modbus* para que os programas Elipse SCADA e IDE Arduino possibilitem a comunicação servo/escravo entre os programas. Já os arquivos da biblioteca *Modbusino* podem ser encontrados em fóruns de suporte do Elipse SCADA e baixados.

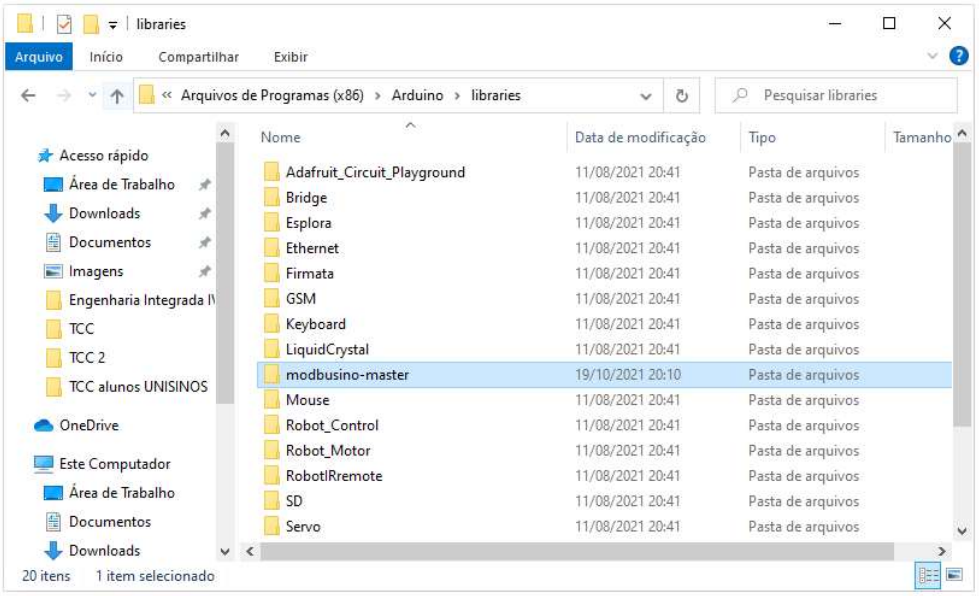
A “biblioteca *Modbusino*” é uma biblioteca *Modbus* que foi desenvolvida por programadores de Arduinos, onde, encontram-se todos os arquivos necessários para comunicação entre os programas Elipse SCADA e IDE Arduino. Os arquivos serão inseridos na pasta “*libraries*”, conforme Figura 25.

Figura 24 – Porta de trabalho para o Arduino.



Fonte: Elaborado pelo autor.

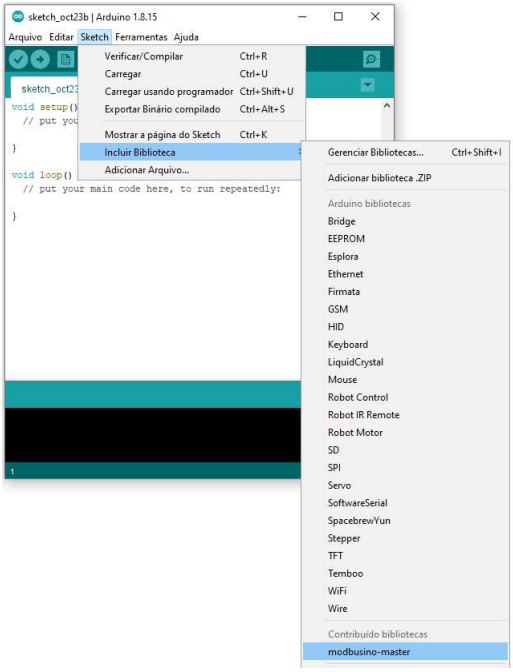
Figura 25 – Arquivos *Modbus.*



Fonte: Elaborado pelo autor.

Após a inserção dos arquivos, é necessário inicializar o IDE Arduino e fazer o carregamento da biblioteca instalada que estará habilitada para escolha no programa, conforme a Figura 26.

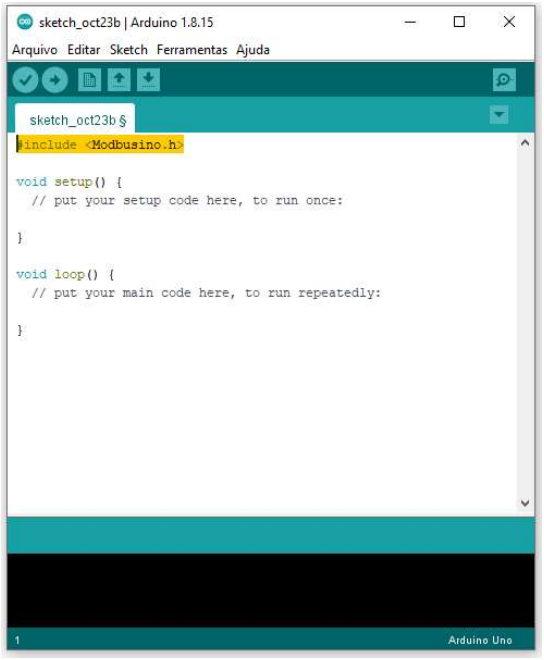
Figura 26 – Inclusão de biblioteca no IDE Arduino.



Fonte: Elaborado pelo autor.

Depois de ser inserida a biblioteca, o IDE Arduino recebe o código que vincula o sistema *Modbus* ao Arduino, conforme, Figura 27. Com a inserção da biblioteca, ficou habilitada a programação que possibilita a comunicação da rede *Modbus* com o Arduino.

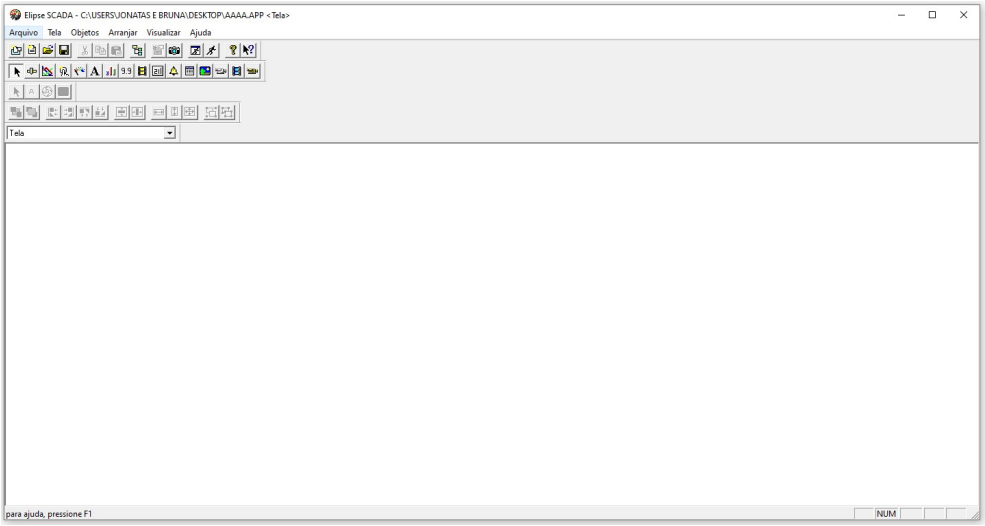
Figura 27 – Inclusão de código *Modbus* ao *software*.



Fonte: Elaborado pelo autor.

Após a inclusão da biblioteca *Modbus* no IDE Arduino, foi necessário configurar o programa Elipse SCADA, abrindo o programa e iniciando novo projeto, conforme Figura 28.

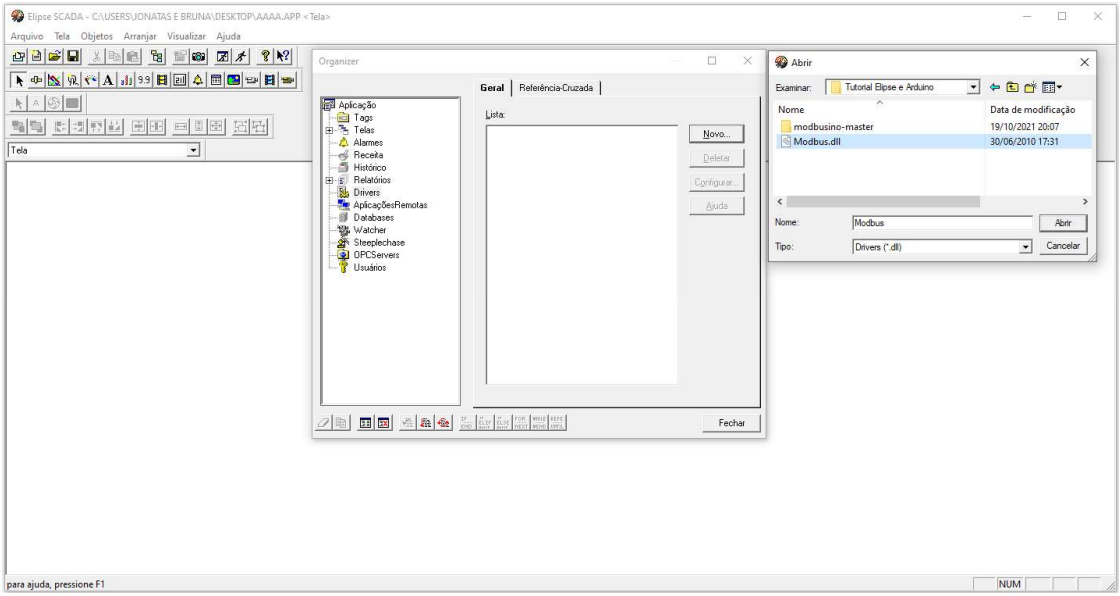
Figura 28 – Tela inicial Elipse SCADA.



Fonte: Elaborado pelo autor.

Primeiramente foi necessário inserir o arquivo *Modbus* que foram baixados junto com os arquivos para o Arduino. É necessário abrir a guia “organizer”, “drivers” e buscar o arquivo dentro da pasta onde foram armazenados os arquivos baixados, conforme Figura 29.

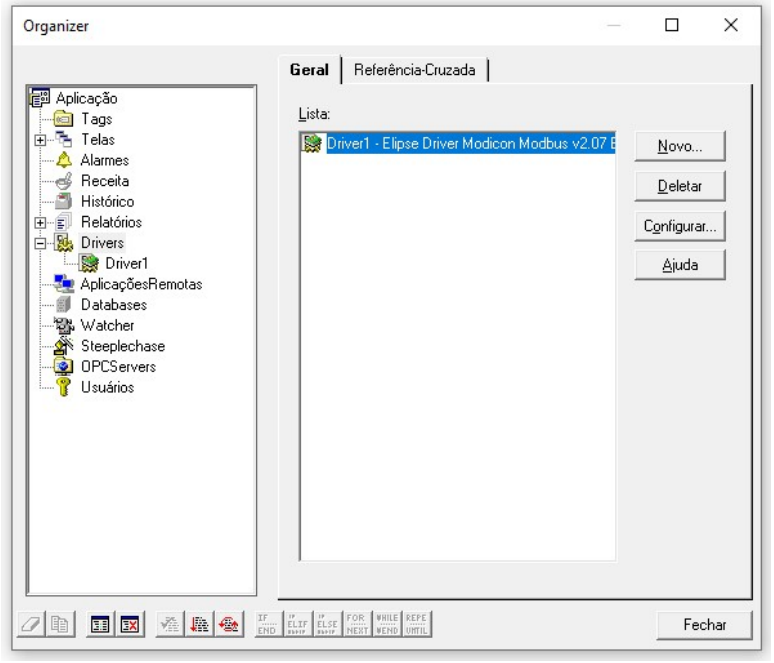
Figura 29 – Inserção de driver *Modbus*.



Fonte: Elaborado pelo autor.

Após a inserção do arquivo, o driver estará habilitado dentro do programa Elipse SCADA, sendo assim, possibilita que o programa também esteja compatível com a rede *Modbus*, conforme Figura 30.

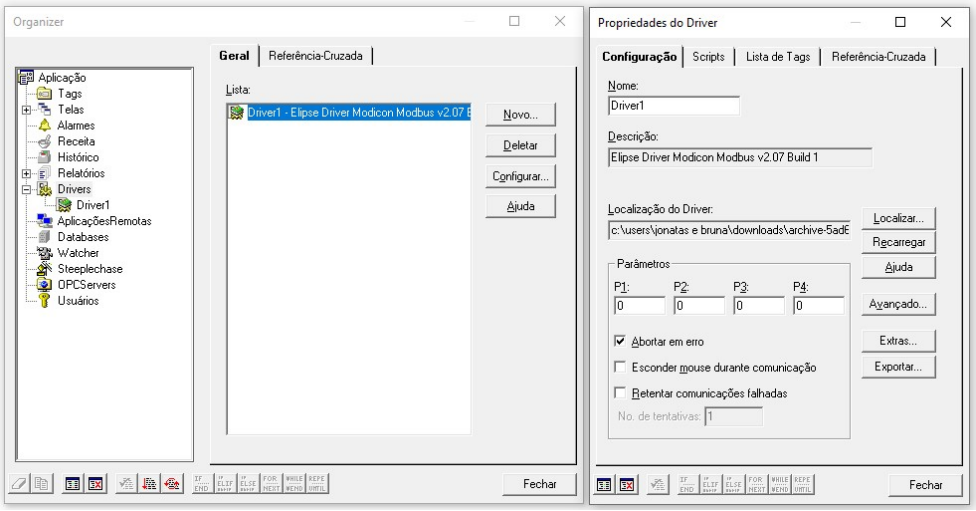
Figura 30 – Driver habilitado.



Fonte: Elaborado pelo autor.

Próxima etapa, foi configurar o driver para possibilitar a comunicação com o IDE Arduino, conforme Figura 31.

Figura 31 – Configuração inicial do driver.

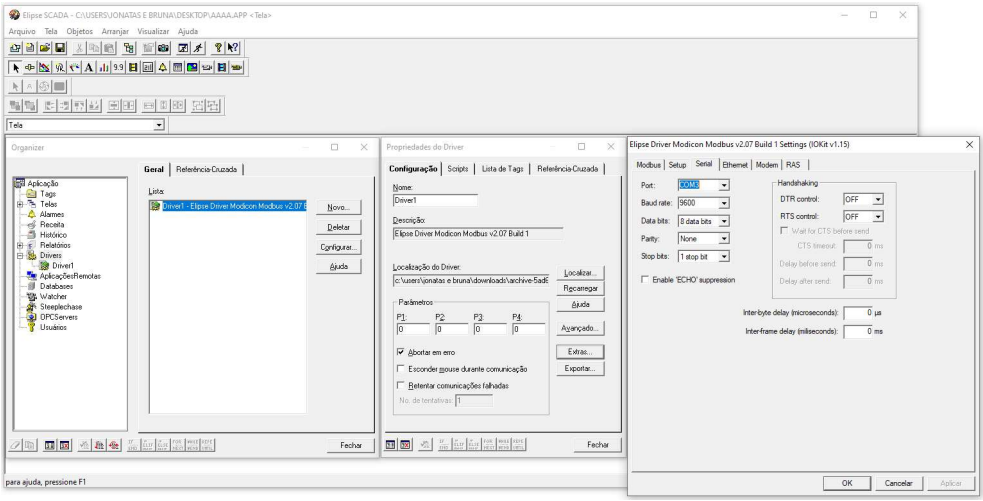


Fonte: Elaborado pelo autor.

Os parâmetros (P1, P2, P3 e P4), não foram configurados, pois são parâmetros para utilização do programa Elipse SCADA com CLP's.

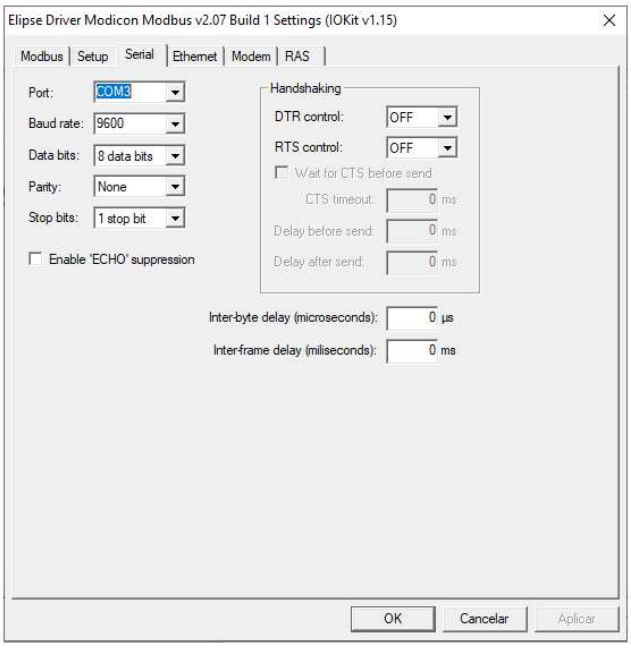
Para a configuração inicial, foi necessário habilitar a porta do Arduino, que no projeto desenvolvido foi habilitada a porta (COM3), conforme Figura 32 e Figura 33. Os demais valores não foram alterados para o desenvolvimento dos projetos.

Figura 32 – Configuração da porta de comunicação.



Fonte: Elaborado pelo autor.

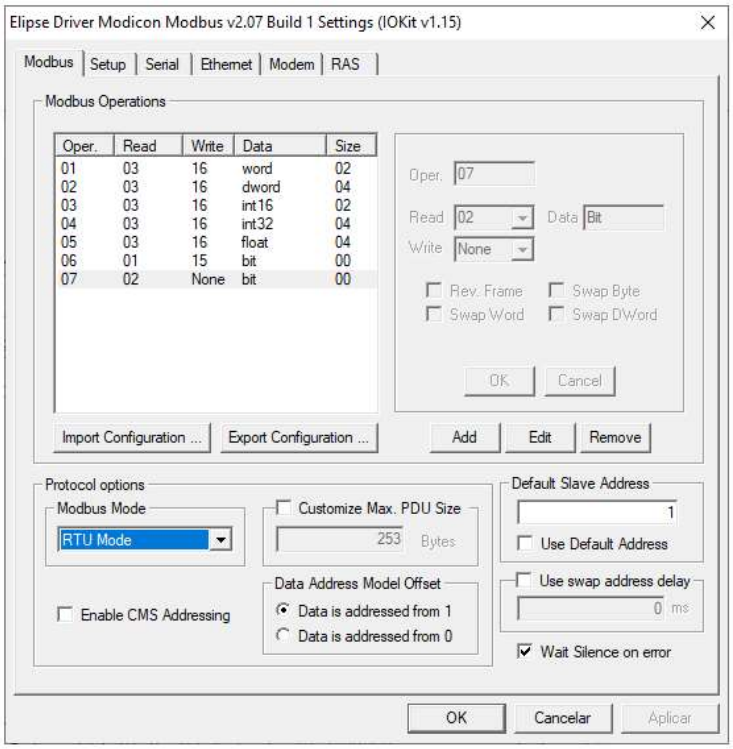
Figura 33 – Ajustes da porta de comunicação.



Fonte: Elaborado pelo autor.

Outro parâmetro que deve ser mantido é o modo de controle do *Modbus* em RTU *Mode*, conforme Figura 34.

Figura 34 – Configuração do modo de controle.

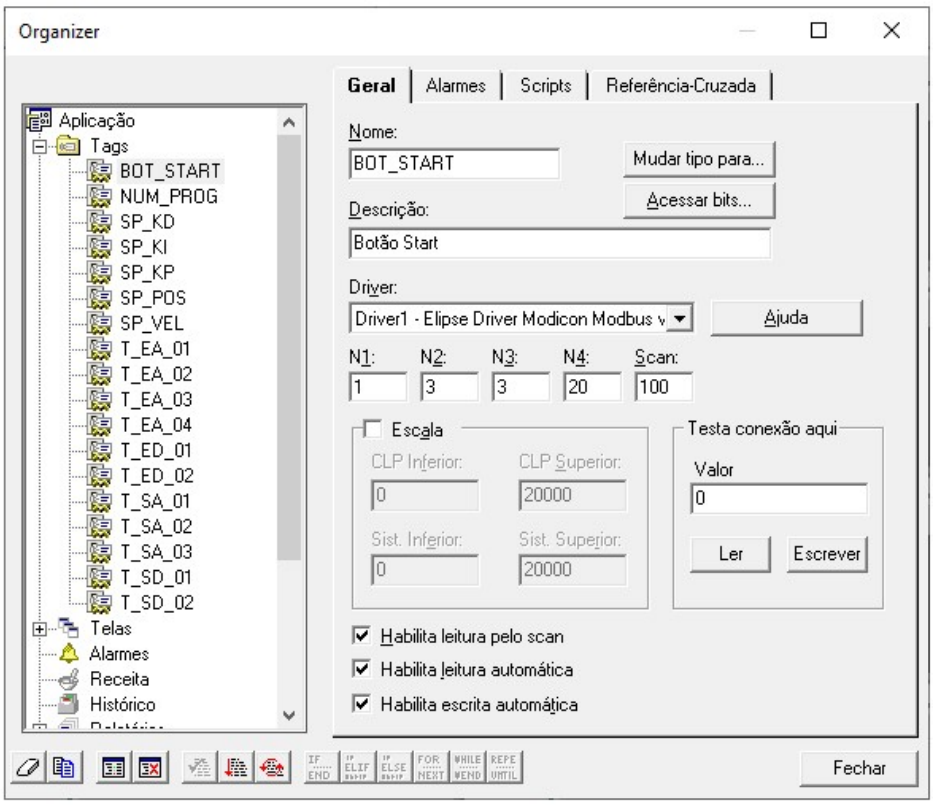


Fonte: Elaborado pelo autor.

Essas são as configurações para o driver *Modbus* funcionar e ter comunicação com o Arduino.

A próxima etapa necessária, foi criar e configurar os Tags, onde foi necessário configurar os parâmetros (N1, N2, N3 e N4) conforme Figura 35. São estes parâmetros que permitem a comunicação com o driver associado ao Elipse SCADA, que no caso do desenvolvimento deste projeto é do Arduino.

Figura 35 – Inserção de tags.



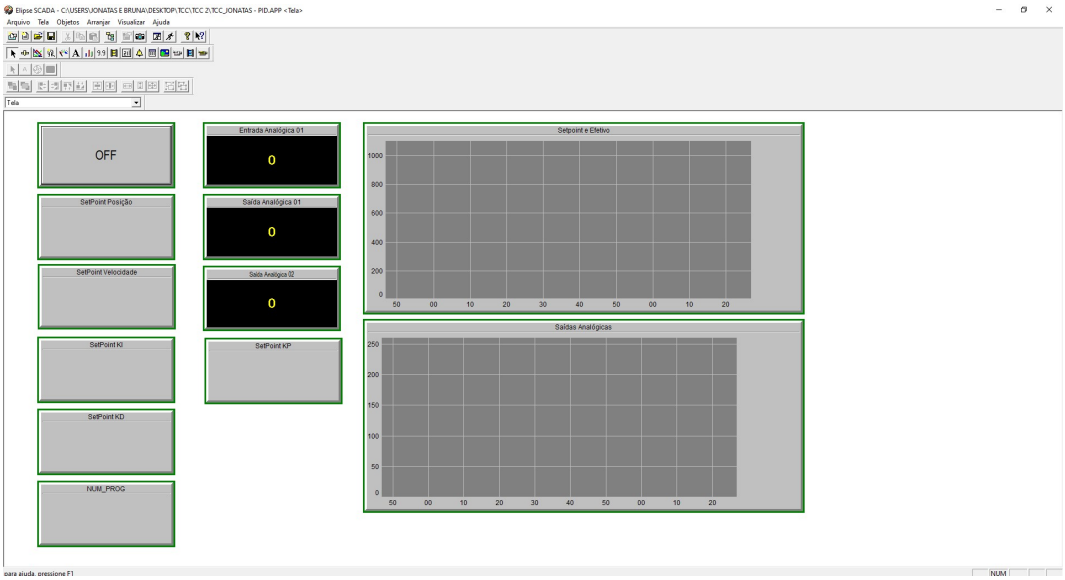
Fonte: Elaborado pelo autor.

A supervisão de um processo com o Elipse SCADA ocorre através da leitura de variáveis de processos no campo. Os valores dessas variáveis são associados a objetos do sistema chamados Tags.

Os Tags desenvolvidos, foram nomeados e configurados para as funções desenvolvidas para o projeto. Cada Tag criado, será atribuído a uma função de controle do supervisório e tem comunicação simultânea com o *software* desenvolvido para o Arduino. O parâmetro N4 determina a posição dentro do vetor de dados no Arduino.

Após as configurações, foram desenvolvidas as telas de controle de cada sistema hidráulico desenvolvido e será controlado e monitorado pela interface de controle. A Figura 36 exemplifica a interface de controle com algumas funções desenvolvidas para teste.

Figura 36 – Interface IHM.



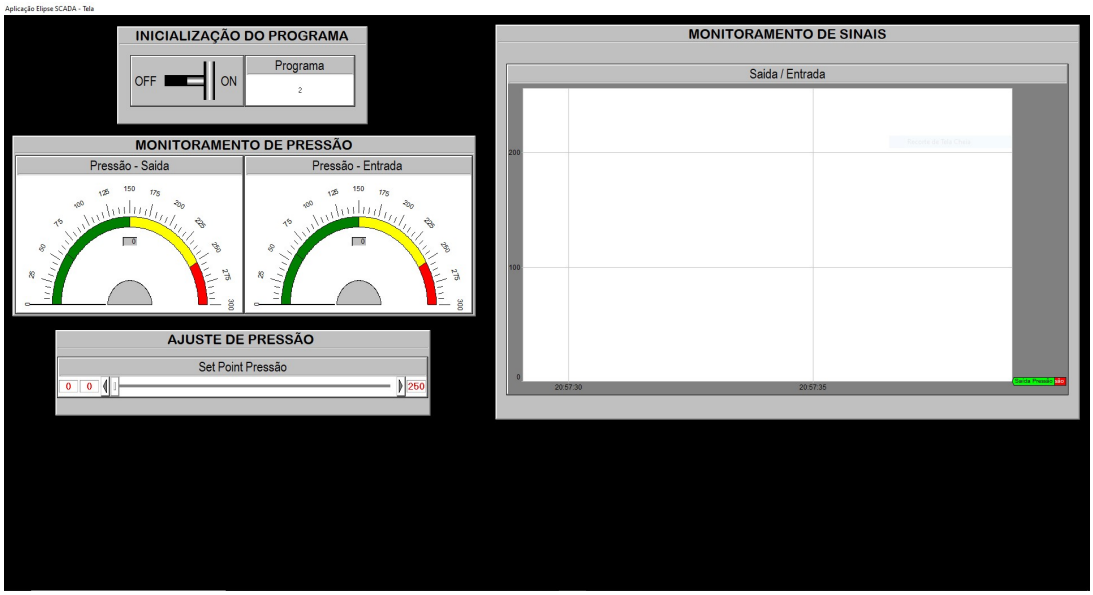
Fonte: Elaborado pelo autor.

4.1.1 Interface de controle de pressão

A interface desenvolvida para o controle de pressão de sistemas hidráulicos, tem a finalidade de controlar de válvulas de alívio proporcionais e monitorar o desempenho dos dados de entrada e saída do Arduino.

A Figura 37 mostra o supervisório desenvolvido para a proposta de controle de sistemas hidráulicos que precisam de controle de pressão.

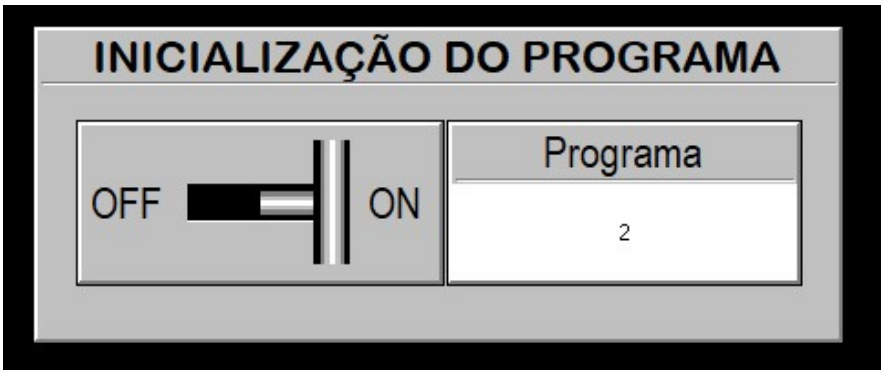
Figura 37 – Supervisório de controle de pressão para sistemas hidráulicos.



Fonte: Elaborado pelo autor.

A primeira função que o supervisório possibilita é de inicialização do programa, conforme Figura 38, com o botão de liga e desliga e a opção de escolha de programa, esta que pode abranger diversas propostas com a mesma interface, tornando o dispositivo versátil.

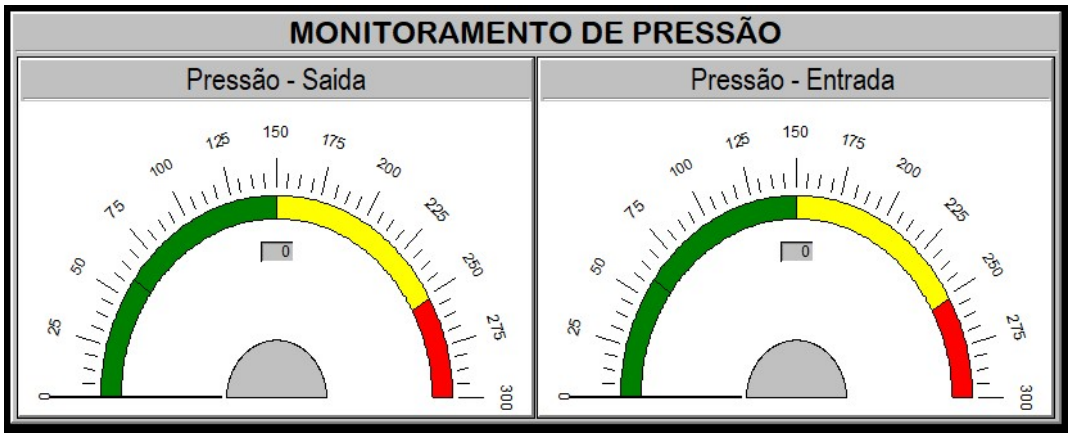
Figura 38 – Botão e seleção de programa.



Fonte: Elaborado pelo autor.

A segunda tela do sistema supervisório, Figura 39, tem a finalidade de monitorar as pressões de entrada (transdutor de pressão) e saída (controle da válvula de alívio). Em um comparativo prático, a pressão de saída é parametrizada conforme a programação desenvolvida que relaciona sempre a pressão lida no transdutor de pressão.

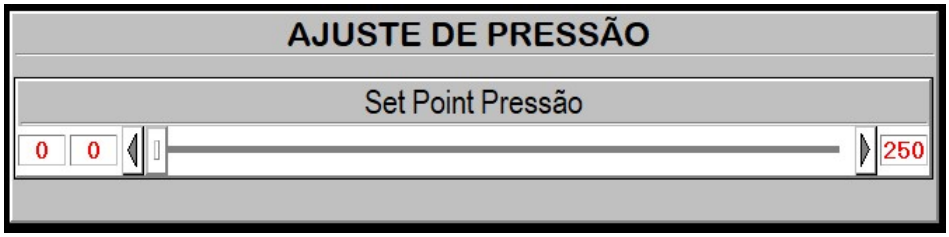
Figura 39 – Monitoramento de pressões.



Fonte: Elaborado pelo autor.

O ajuste de pressão ilustrado na Figura 40, mostra a possibilidade de o sistema ter uma regulagem “manual” da válvula de alívio proporcional caso haja necessidade de ajuste de pressão.

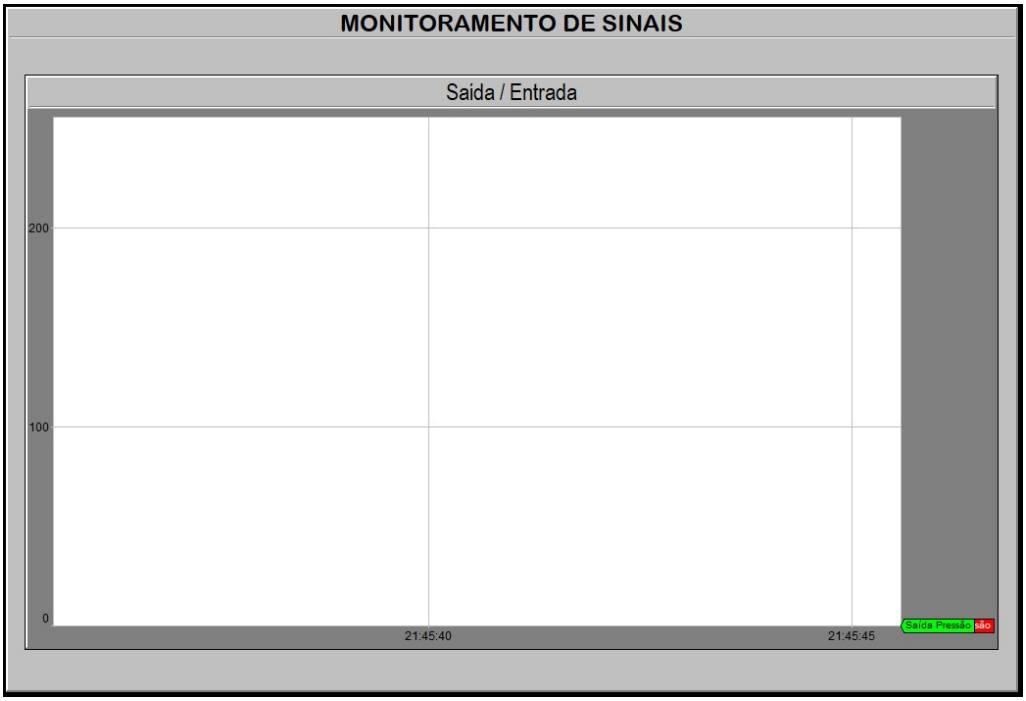
Figura 40 – Ajuste de pressão do sistema hidráulico.



Fonte: Fonte: Elaborado pelo autor.

A tela de monitoramento de sinais (Figura 41) tem controle em tempo real do desempenho dos sinais de entrada e saída do Arduino. Eles monitoram as oscilações dos sinais enviados e recebidos e apontam se há defasagem nas respostas do programa com os *hardwares*.

Figura 41 – Monitoramento de sinais do *software*.



Fonte: Elaborado pelo autor.

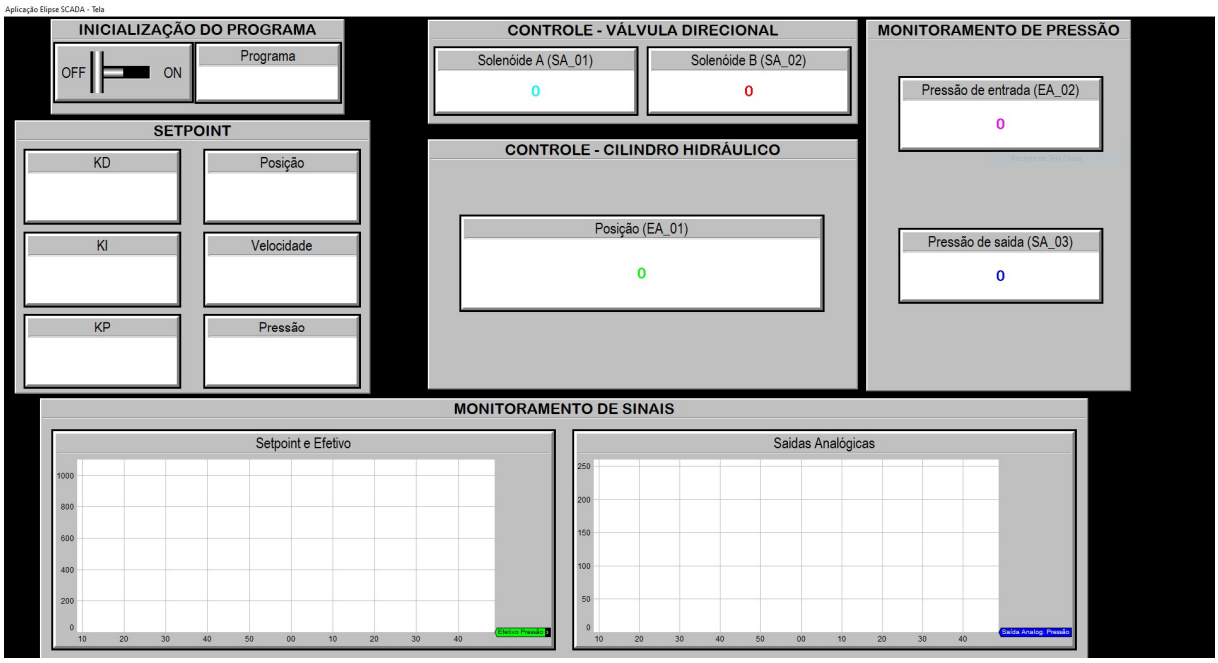
4.1.2 Interface de controle de pressão e vazão

A interface desenvolvida para o controle de pressão e vazão de sistemas hidráulicos, possibilita um sistema mais versátil e completo. Além de controlar e monitorar a válvula de alívio proporcional, o sistema controla também a velocidade de deslocamento do atuador. A válvula direcional por ter controle proporcional,

possibilita que a vazão seja controlada e impactando diretamente na variação de velocidade do atuador e para o controle de posição do atuador, foi utilizado o sensor de deslocamento.

A Figura 42 apresenta a interface desenvolvida para o controle de um sistema hidráulico completo, mas também possibilita que a interface possa ser utilizada em um sistema que seja requerido somente o controle do atuador como também somente do controle da pressão do sistema.

Figura 42 – Supervisório de controle de pressão e vazão para sistemas hidráulicos.



Fonte: Elaborado pelo autor.

Para iniciar o funcionamento do sistema foi necessária a escolha do programa para a aplicação desejada e acionar através do botão *on/off*. O *setpoint* tem a finalidade de configurar o desempenho do sistema, onde, KP é o controlador proporcional com a finalidade de reduzir o tempo de resposta, KI é um controlador integral com finalidade de eliminar o erro no estado estacionário e KD é o controlador derivativo que estabiliza o controle do sistema e aumenta a agilidade de uma resposta. Estas funções estão diretamente relacionadas com a eficiência do deslocamento do atuador que é configurado através do configurador de posição e velocidade. O ajuste de pressão é do sistema e se dá no configurador *setpoint* de pressão.

A Figura 43 apresenta os configuradores do *setpoint*, botão de inicialização do sistema e escolha do programa.

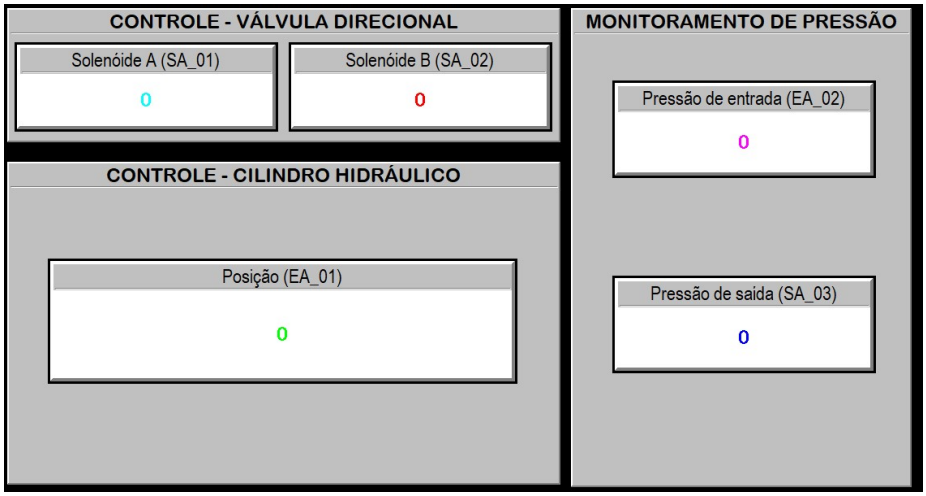
Figura 43 – Inicialização do programa.

Fonte: Elaborado pelo autor.

Para o monitoramento dos componentes do sistema hidráulico, o supervisório contempla displays pontuais que para a válvula direcional pode-se monitorar os valores e em qual lado está sendo enviado o sinal para a válvula. Em simultâneo pode-se verificar a posição do atuador em movimento e o ponto onde parou.

Os displays para o monitoramento das pressões do transdutor de pressão (entrada) e regulagem da válvula de alívio (saída) foram alocados a direita do monitor, para facilitar o monitoramento simultâneo com as demais informações, conforme a Figura 44.

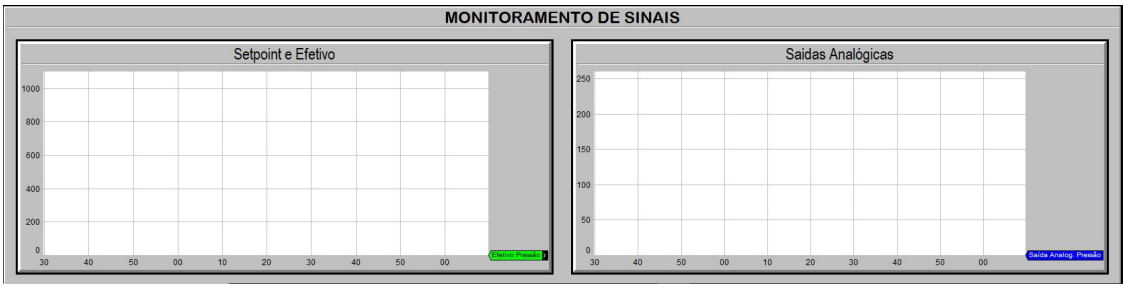
Figura 44 – Displays de monitoramento do sistema.



Fonte: Elaborado pelo autor.

O monitoramento de sinais apresenta a mesma finalidade do sistema de controle de pressão, porém com o incremento de todas as informações de saídas analógicas do Arduino além de monitorar o *Setpoint* e efetivo do sistema, conforme apresentado na Figura 45.

Figura 45 – Monitoramento de sinais.



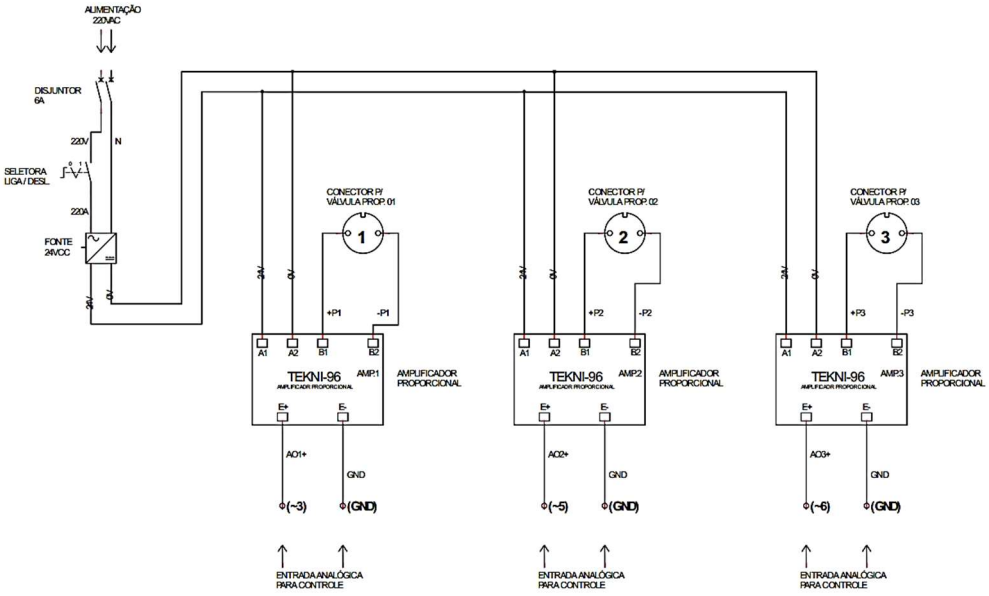
Fonte: Elaborado pelo autor.

4.2 *HARDWARE*

Para possibilitar a automação dos sistemas hidráulicos propostos, foi desenvolvido o *hardware* com a finalidade de facilitar as aplicações, pois contempla plugs de engate rápidos.

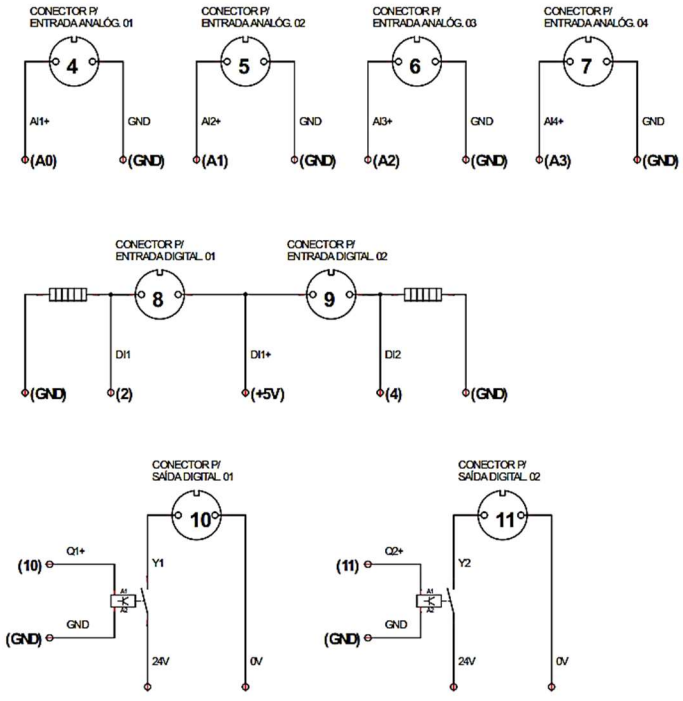
O hardware utiliza amplificadores proporcionais que estão interligados ao Arduino. Os amplificadores são alimentados por uma fonte que está sendo alimentada pela rede de 220V. Para a segurança da fonte, foi utilizado uma seletora junto a um disjuntor. Conforme o diagrama nas Figura 46.

Figura 46 – Diagrama elétrico (amplificadores proporcionais).



Fonte: Elaborado pelo autor.

Figura 47 – Diagrama elétrico (conectores).



Fonte: Elaborado pelo autor.

Além dos componentes elétricos, o *hardware* utiliza um painel elétrico que foi desenvolvido para ajudar a proteger os componentes e organiza os cabos e conectores que foram utilizados. A Figura 48, detalha os conectores do *hardware*.

Figura 48 – Conectores do painel elétrico.

Fonte: Elaborado pelo autor.

Para padronizar o *hardware*, nas conexões foram inseridos nomes para facilitar programações e conectadas às portas do Arduino, conforme mostra a Tabela 1. A Figura 49, apresenta o painel com seus conectores nomeados para possibilitar uma melhor organização para montagem nos componentes eletro-hidráulicos.

Figura 49 – Denominação dos conectores.

Fonte: Elaborado pelo autor.

Tabela 1 - Portas de entrada e saída.

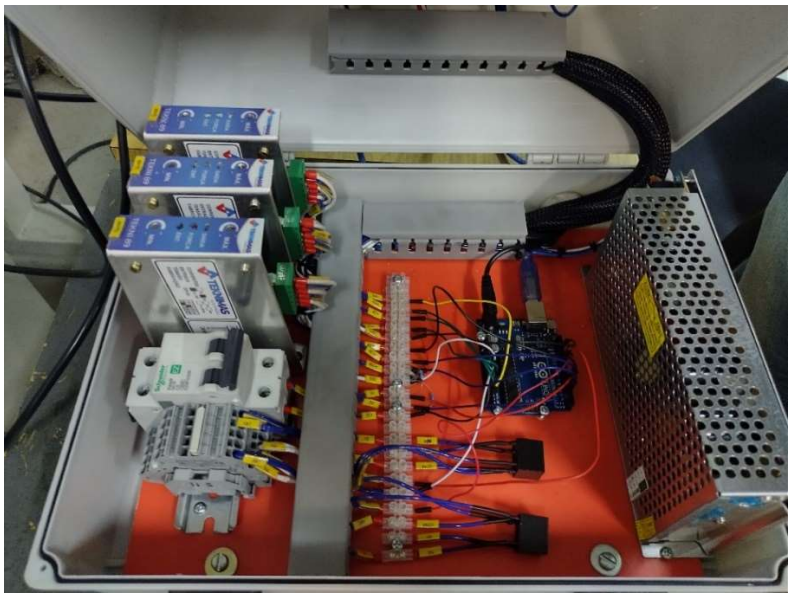
PORTAS CONECTADAS AO ARDUINO

ARDUINO	CONEXÕES
2	ENTR_01
4	ENTR_02
A0	ENTR_ANALOG_01
A1	ENTR_ANALOG_02
A2	ENTR_ANALOG_03
A3	ENTR_ANALOG_04
10	SAIDA_01
11	SAIDA_02
3	SAIDA_ANALOG_01
5	SAIDA_ANALOG_02
6	SAIDA_ANALOG_03

Fonte: Elaborado pelo autor.

O Arduino foi centralizado dentro do painel para facilitar a organização dos fios utilizados, conforme mostra a Figura 50.

Figura 50 – Montagem do Arduino.



Fonte: Elaborado pelo autor.

O painel também foi montado de forma organizada, buscando a melhor disposição dos cabos conforme a Figura 51.

Figura 51 – Montagem elétrica dos conectores.

Fonte: Elaborado pelo autor.

4.3 PROGRAMA DO ARDUINO

Os programas foram desenvolvidos para o acionamento, controle e monitoramento através do método de programação por linguagem C++. Cada proposta de sistema hidráulico desenvolvido contempla seu *software* independente. O *software* se completa com a utilização dos supervisórios desenvolvidos.

Foram desenvolvidos três programas, onde o primeiro possibilita somente o controle de velocidade e posição de atuadores. Posteriormente foi desenvolvido o *software* para controle de pressão. Com o desenvolvimento dos programas de controle de pressão e velocidade, foi possível desenvolver a terceira proposta, que contempla o sistema completo e possibilita alto desempenho e será apresentado detalhadamente.

As programações foram feitas no programa IDE Arduino e posteriormente inseridas no *hardware*.

O programa para sistema com controle de velocidade e pressão foi detalhado para mostrar as funções desenvolvidas para a lógica de funcionamento e foi inserido no Anexo A com todo o programa desenvolvido. As propostas de controle de pressão e controle de velocidade desenvolvidas separadamente foram inseridas no

Anexo B e Anexo C, pois suas funções foram inseridas dentro do programa completo.

4.3.1 Sistema de controle de velocidade e pressão

A programação desenvolvida para controle de velocidade e pressão, contempla os programas com os controles individuais propostos, ou seja, pode-se habilitar somente o controle de pressão ou de velocidade.

As primeiras linhas do programa desenvolvido são as inserções da biblioteca *modbus* para comunicação do IDE Arduino com o Elipse SCADA e a lógica do PID.

O Controlador proporcional integral derivativo (PID), é uma técnica de controle de processos que une as ações derivativa, integral e proporcional, fazendo assim com que o sinal de erro seja minimizado pela ação proporcional, zerado pela ação integral e obtido com uma velocidade antecipada pela ação derivativa.

```
#include <Modbusino.h>
#include <PID_v2.h>
```

O código abaixo inicializa o ID do dispositivo para comunicação com o a rede *modbus*.

```
ModbusinoSlave modbusino_slave(1);
```

Com a inclusão do código PID, é necessário que a estrutura para o PID seja declarada, conforme mostra a linha abaixo.

```
PID_v2 myPID(5.0,0.0,0.0, PID::Direct);
```

Para que seja alocada o número de registradores para comunicação com o Elipse, foi inserida a linha abaixo:

Os registradores servem a um propósito específico e contribuem para o funcionamento do programa no processamento das instruções, o que significa que são usados nas etapas principais de busca, interpretação e execução das instruções.

```
uint16_t tab_reg[20];
```

A variável tempo recebe o número de milissegundos decorridos desde o início do programa. O tipo de dado retornado é *unsigned long*.

```
unsigned long millisTarefa1 = millis();
```

Para o auxílio no desenvolvimento das posições para tabela *modbus* com o Elipse SCADA, foram enumerados e adicionados valores como comentários.

```
/*
00 - SAIDA_01
01 - SAIDA_02
02 - ENTR_01
03 - ENTR_01
04 - SAIDA_ANALOG_01
05 - SAIDA_ANALOG_02
06 - SAIDA_ANALOG_03
07 - ENTR_ANALOG_01
08 - ENTR_ANALOG_02
09 - ENTR_ANALOG_03
10 - ENTR_ANALOG_04
11 - NUMERO DO PROGRAMA
12 - SET POINT POSICAO PROGRAMA
13 - SET POINT VELOCIDADE PROGRAMA
14 - KP
15 - KI
16 - KD
17
18
19 - ATUALIZA PROGRAMA / BOT START
*/
```

Com o auxílio das posições inseridas como comentários, foi possível inserir as variáveis da tabela *modbus*. Estas constantes podem ser chamadas de *Tags*. Estas *Tags* foram utilizadas dentro do programa Elipse SCADA.

```
const int T_SD_01 = 0;
const int T_SD_02 = 1;
const int T_ED_01 = 2;
const int T_ED_02 = 3;
const int T_SA_01 = 4;
const int T_SA_02 = 5;
const int T_SA_03 = 6;
const int T_EA_01 = 7;
const int T_EA_02 = 8;
const int T_EA_03 = 9;
const int T_EA_04 = 10;
const int NUM_PROG = 11;
const int SP_POS = 12;
const int SP_VEL = 13;
const int SP_KP = 14;
const int SP_KI = 15;
const int SP_KD = 16;
```

```
const int SP_PR = 17;
const int BOT_START = 19;
```

Com a determinação dos valores das posições das variáveis, elas foram inseridas para serem utilizadas e vinculadas aos pinos do Arduino.

```
const int SAIDA_01 = 10;
const int SAIDA_02 = 11;
const int ENTR_01 = 2;
const int ENTR_02 = 4;
const int SAIDA_ANALOG_01 = 3;
const int SAIDA_ANALOG_02 = 5;
const int SAIDA_ANALOG_03 = 6;
const int ENTR_ANALOG_01 = 0;
const int ENTR_ANALOG_02 = 1;
const int ENTR_ANALOG_03 = 2;
const int ENTR_ANALOG_04 = 3;
```

A função *setup*, serve para a inicialização dos parâmetros do Arduino. Esta função é executada somente uma vez na inicialização.

```
void setup()
```

Dentro da função setup, temos a definição da taxa de transferência para o modbus, além de definir o comportamento dos pinos do Arduino e a inicialização dos parâmetros genéricos do PID.

```
{

   modbusino_slave.setup(115200);

   pinMode(SAIDA_01,OUTPUT);
   pinMode(SAIDA_02,OUTPUT);
   pinMode(SAIDA_ANALOG_01,OUTPUT);
   pinMode(SAIDA_ANALOG_02,OUTPUT);
   pinMode(SAIDA_ANALOG_03,OUTPUT);
   pinMode(ENTR_01,INPUT);
   pinMode(ENTR_02,INPUT);


   myPID.Start(100,0,100);
}
```

A função loop, repete consecutivamente as leituras de entradas e saídas enquanto a placa estiver ligada, permitindo o seu programa responder a essas mudanças.

```
void loop()
```

```
{
    double saida_auxiliar; // declara variável de saída


    tab_reg[T_EA_01] = analogRead(ENTR_ANALOG_01);
    tab_reg[T_EA_02] = analogRead(ENTR_ANALOG_02);
    tab_reg[T_EA_03] = analogRead(ENTR_ANALOG_03);
    tab_reg[T_EA_04] = analogRead(ENTR_ANALOG_04);

    tab_reg[T_ED_01] = digitalRead(ENTR_01);
    tab_reg[T_ED_01] = digitalRead(ENTR_01);


    analogWrite(SAIDA_ANALOG_01,tab_reg[T_SA_01]);
    analogWrite(SAIDA_ANALOG_02,tab_reg[T_SA_02]);
    analogWrite(SAIDA_ANALOG_03,tab_reg[T_SA_03]);

    digitalWrite(SAIDA_01,logic(tab_reg[T_SD_01]));
    digitalWrite(SAIDA_02,logic(tab_reg[T_SD_02]));
```

A próxima linha da programação permite a escolha do programa para ser executado, neste caso está sendo escolhido o programa de número 3. Os programas desenvolvidos que podem ser habilitados são:

- Programa 1: Configurado para somente controle de velocidade;
- Programa 2: Configurado para somente o controle de pressão;
- Programa 3: Configurado para controle completo do sistema;

```
    if (tab_reg[NUM_PROG]==3)
    {
```

Ainda dentro da função onde foi determinado o número do programa, têm a atualização da saída de pressão e este possibilita o controle da válvula proporcional de pressão utilizada no sistema.

```
        tab_reg[T_SA_03]=tab_reg[SP_PR];
```

Os parâmetros de proporcionalidade configurados através do Elipse SCADA, foram alterados dentro da lógica desenvolvida abaixo. Além de inicializar o sistema, é executada a lógica PID, possibilitando a configuração dos parâmetros KP, KD, KI. Após serem configurados os parâmetros, o programa começa a executar o deslocamento do atuador recebendo dados do sensor de posição.

```
        if (logic(tab_reg[BOT_START]))
        {

          myPID.Start( tab_reg[T_EA_01],0, tab_reg[SP_POS]);
```

```
  double aux1 = float(tab_reg[SP_KP]) /10;
  double aux2 = float(tab_reg[SP_KI]) /10;
  double aux3 = float(tab_reg[SP_KD]) /10;
  myPID.SetTunings( aux1, aux2, aux3);
  myPID.SetOutputLimits(-255, 255);
  myPID.SetSampleTime(20);

}
```

A constante desenvolvida abaixo, tem a funcionalidade de rodar o PID e colocar o valor de saída na variável output_PID.

```
const double output_PID = myPID.Run(tab_reg[T_EA_01]);
```

A função de condição desenvolvida abaixo tem como finalidade atualizar as saídas para as válvulas direcionais conforme o valor em output_PID, limitando de acordo com o valor de velocidade máxima selecionada para o sistema. Valores menores que 20 são desconsiderados para a saída, evitando assim as oscilações. Foi inserido um valor de 25 unidades com a finalidade de ultrapassar a corrente mínima para movimentar a válvula, pois sem esse valor a válvula oscila e a movimentação do atuador fica instável.

```
if (output_PID<20 && output_PID>-20 )
{
  tab_reg[T_SA_01]=0;
  tab_reg[T_SA_02]=0;
}

if (output_PID>20)
{

   saida_auxiliar = output_PID;
   if (saida_auxiliar>230)
   {
      saida_auxiliar=230;
   }


   if (saida_auxiliar > tab_reg[SP_VEL]-25)
   {
      saida_auxiliar = tab_reg[SP_VEL]-25;
   }

   tab_reg[T_SA_01]=saida_auxiliar+25;

   tab_reg[T_SA_02]=0;
}

if (output_PID<-20)
{
```

```
            tab_reg[T_SA_01]=0;


             saida_auxiliar = output_PID;
             if (saida_auxiliar<-230)
             {
               saida_auxiliar=-230;
             }

            if (saida_auxiliar < -tab_reg[SP_VEL]+25)
            {
              saida_auxiliar = - tab_reg[SP_VEL]+25;
            }
            tab_reg[T_SA_02]=-saida_auxiliar+25;
          }


    }
```

A condição *else*, serve para zerar as saídas, caso o número do programa não seja o selecionado, no caso desta programação, se não for o programa número 3.

```
    else
    {
          tab_reg[T_SA_01]=0;
          tab_reg[T_SA_02]=0;
          tab_reg[T_SA_03]=0;
          tab_reg[T_SD_01]=0;
          tab_reg[T_SD_02]=0;
    }
```

A função abaixo, determina a frequência de atualização do *Modbus*.

```
    //if((millis() - millisTarefa1) > 50)
    //{
      // millisTarefa1 = millis();
      // modbusino_slave.loop(tab_reg, 20);
    // }
     modbusino_slave.loop(tab_reg, 20);
}
```

A função auxiliar desenvolvida, serve para converter a lógica booleana do valor de entrada, ou seja, se o valor for maior ou igual a 1, o sinal valida a função.

```
bool logic(int valor)
{
  if (valor>=1) return HIGH;
  else return LOW;
}
```

A última função do programa, serve para escalar os valores de saídas.

```
int escala(int valor,int val_in, int val_final)
{
  double aux1 = float(valor);
  double aux2 = float(val_in);
  double aux3 = float(val_final);

  double result = ((aux1/255) * (aux3-aux2)) + aux2;

  return int(result);
}
```

4.4 DIAGRAMAS HIDRÁULICOS

Foram desenvolvidas três propostas de sistemas hidráulicos que foram automatizados.

A primeira proposta contempla um sistema com compensação de pressão e a finalidade de obter redução do consumo de potência quando o sistema estiver sem mover o atuador. A bomba de vazão fixa é muito aplicada em unidades hidráulicas utilizadas em prensas e pequenas aplicações que necessitam de movimentos de cargas, como esteiras transportadoras, braços de articulação de pulverizadores, entre outros.

A segunda proposta consiste em um sistema que possibilita o controle de velocidade e posição do atuador, proporcionando melhor controle para sistemas que exigem exatidão em movimento de cargas, como guindastes, prensas hidráulicas, injetoras, entre outros.

A terceira proposta apresenta um sistema mais completo, voltado para o alto desempenho, unindo proporcionalidade tanto na regulagem de vazão quanto pressão. Para o monitoramento do sistema foi inserido transdutor de pressão e sensor de posição para enviar dados para o Arduino. Esta proposta pode ser aplicada em diversos diagramas, portanto abrange todas as aplicações de alta performance atuais no mercado.

As propostas foram detalhadas separadamente para evidenciar suas particularidades e possíveis aplicações.

4.4.1 Sistema de compensador de pressão

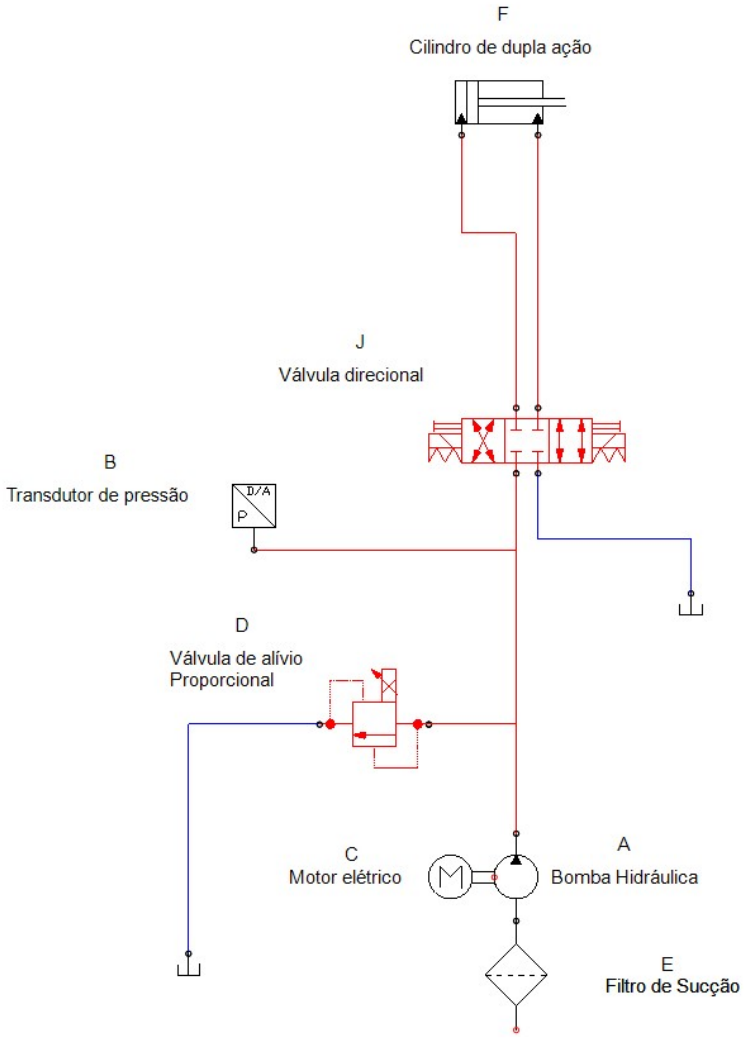
O sistema proposto com compensação de pressão consiste em controlar a pressão de trabalho e pressão em repouso do atuador. A proposta necessita da utilização de um transdutor de pressão e válvula de alívio proporcional.

Por meio do controle da pressão que foi programada pelo Arduino, o sistema pode ser monitorado e automatizado. O transdutor de pressão envia para o Arduino os dados necessários para monitoramento e controle da pressão.

Com o controle da pressão o sistema passa a ser ventado (pressão em repouso próximo de 0 Bar) e segue somente regulando a pressão de trabalho quando acionado a solenoide da válvula direcional.

Para sistemas que utilizam bombas de vazão fixas com uma demanda de trabalho intermitente, a proposta apresentada pode apresentar ganho econômico, elevada vida útil dos componentes, redução na geração de calor e principalmente na redução do consumo de potência, pois a relação de potência está diretamente associada à pressão regulada para o trabalho do sistema. A Figura 52 apresenta o diagrama proposto para a função de compensador de pressão.

Figura 52 – Sistema com compensador de pressão.



Fonte: Elaborado pelo autor.

4.4.2 Sistema com controle de velocidade

O sistema proposto com controle de velocidade, tem como principal função, o controle da vazão no sistema.

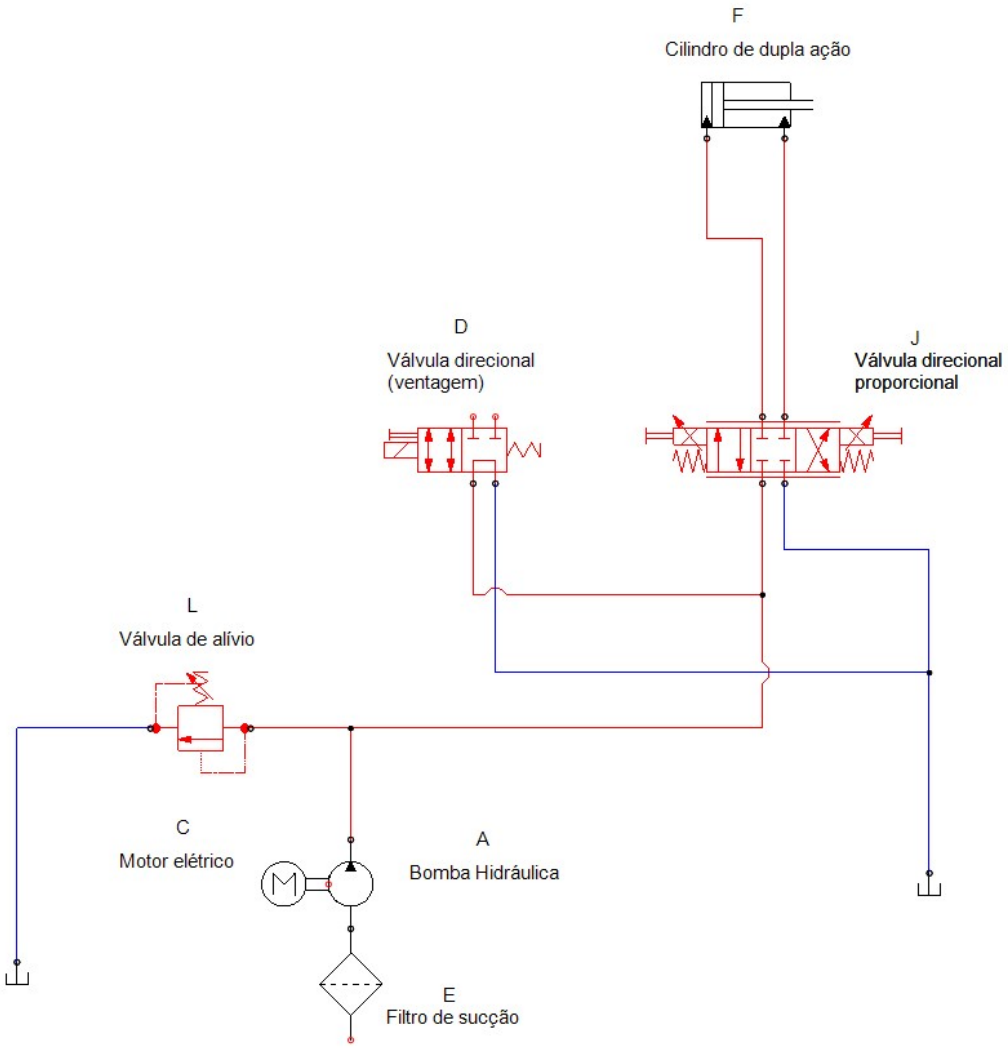
A válvula direcional proporcional utilizada, permite que o sistema tenha controle de vazão e consequentemente, controle de velocidade.

O sensor linear, permite que o cilindro hidráulico seja monitorado, ou seja, a posição de avanço e recuo do atuador é monitorada. O monitoramento possibilita que os dados enviados para o Arduino, sejam convertidos em controle de vazão, velocidade e posição.

A maior vantagem desta proposta está na exatidão do controle do atuador, proporcionando maior desempenho em aplicações de grande complexibilidade.

A Figura 53 apresenta o diagrama proposto para a função de controle de velocidade e posição.

Figura 53 – Sistema com controle de velocidade.



Fonte: Elaborado pelo autor.

4.4.3 Sistema com controle de velocidade e pressão

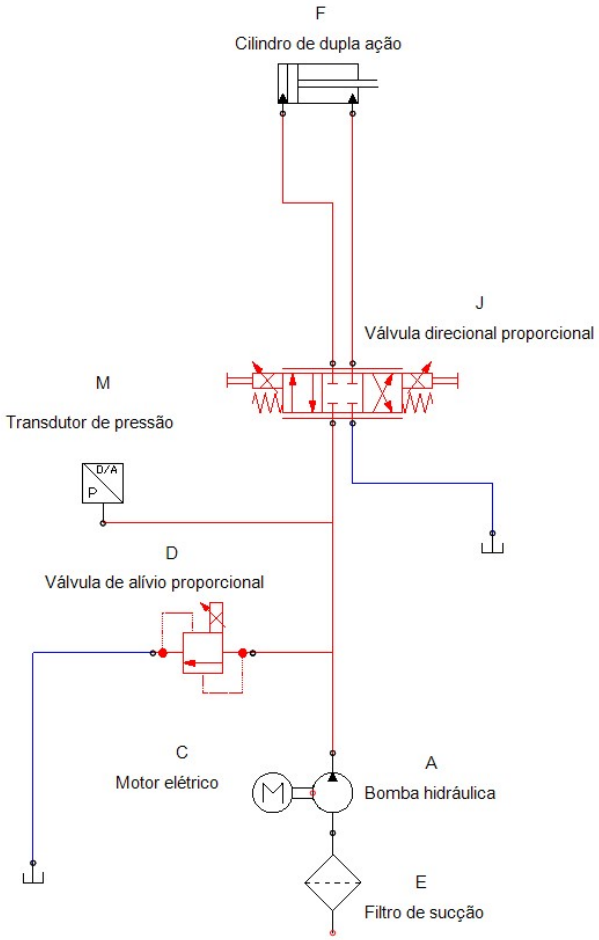
O sistema proposto com controle de pressão e vazão possibilita diversas aplicações de alta performance, pois possibilita que as pressões sejam configuradas conforme a exigência de força que o sistema solicita.

O controle de pressão que o transdutor de pressão informa para o Arduino pode ser corrigido conforme a solicitação de carga que o sistema exige, além de possibilitar que a velocidade de avanço do atuador seja proporcional ao acionamento.

O sistema proposto pode ser aplicado em sistemas navais que exigem tempo de resposta elevado para movimento da embarcação e para equipamentos de corte de toras que necessitam de proporcionalidade de elevadas pressões.

A proposta é atrativa, pois contempla todas as funções desenvolvidas nos demais sistemas. A Figura 54 apresenta o diagrama completo com as funções propostas em um único sistema.

Figura 54 – Sistema com controle de vazão e pressão.



Fonte: Elaborado pelo autor.

5. VALIDAÇÃO DOS RESULTADOS

Os resultados obtidos foram divididos em duas etapas, onde, a primeira etapa foram os resultados preliminares obtidos através dos testes feitos na unidade hidráulica de pequeno porte utilizada para aplicações educativas. A segunda etapa foi realizada com a unidade hidráulica da Rexroth do laboratório da universidade e que é uma unidade hidráulica de porte industrial, com a finalidade de apresentar o funcionamento do trabalho desenvolvido e validar o funcionamento dos programas para o Arduino e garantir controle e monitoramento pelo supervisório.

5.1 *RESULTADOS OBTIDOS COM UNIDADE HIDRÁULICA DIDÁTICA*

O desenvolvimento dos programas para os diagramas hidráulicos propostos foi validado utilizando a bancada hidráulica da Unisinos onde foi montado o sistema de controle de velocidade.

Os resultados validados mostram a influência de alteração dos parâmetros de velocidade, KP, KI e KD. Os valores foram plotados nos gráficos de desempenho que foram desenvolvidos para o supervisório.

A Figura 55, apresenta o sistema desenvolvido para validação dos resultados, onde, foram utilizados, válvula direcional proporcional e sensor de deslocamento do atuador. Não foram testadas as funções com controle de pressão, pois não havia transdutor de pressão disponível.

Figura 55 - Montagem do Sistema Hidráulico.

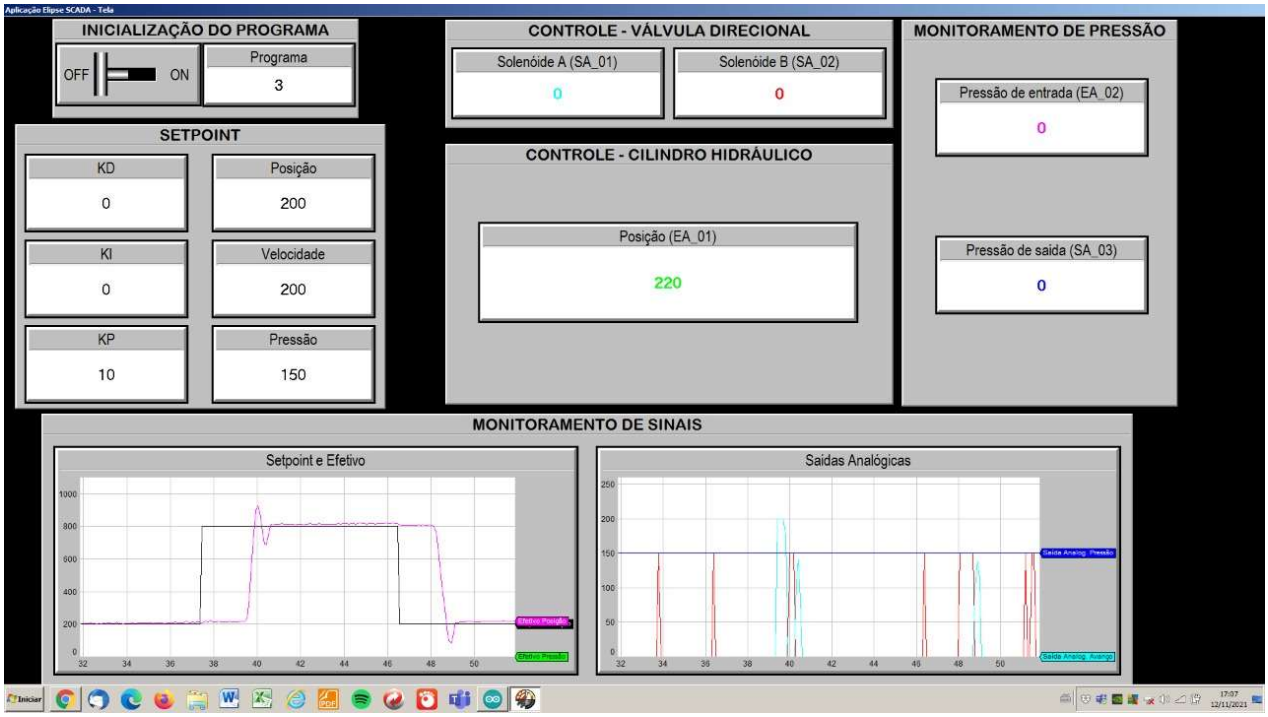
Fonte: Elaborado pelo autor.

Para o monitoramento do desempenho da válvula direcional, as posições selecionadas para parametrização dos dados de comparação foram posição inicial 200 e posição final 800, que são valores configurados no programa e determinam a posição inicial (haste recolhida) e final (haste completamente avançada). Os valores de posição inicial e final são adimensionais, ou seja, são os valores configurados para parametrizar o deslocamento do cilindro em seu curso. O programa utilizado para os testes foi o programa número 3. Além das posições, os parâmetros KD, KI e KP serão alterados para serem obtidos dados comparativos da influência de cada parâmetro na resposta do programa.

O programa número 3 possibilita controle de pressão, porém, para os testes esta função não foi utilizada.

A Figura 56, mostra o resultado obtido com a configuração de KP=10, KI=0, KD=0 e Velocidade=200. As penas de monitoramento de sinais efetivo e *setpoint*, mostram o tempo de resposta entre o valor efetivo (cor preta) e a resposta do solenoide sob influência do ajuste do PID (cor rosa). As penas das saídas analógicas mostram a resposta de acionamento entre as bobinas do solenoide A e solenoide B.

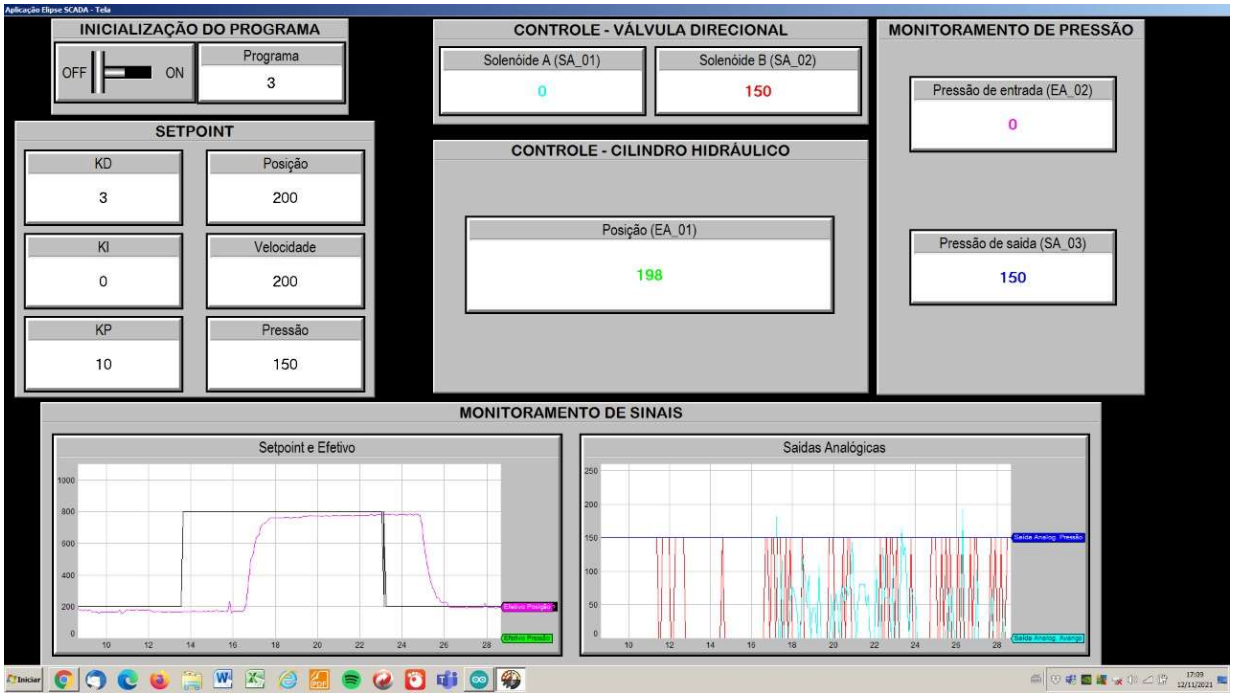
Figura 56 - Parametrização e resultados.



Fonte: Elaborado pelo autor.

Em comparação da Figura 56 com a Figura 57, é possível verificar que alterando o parâmetro KD, foi obtido um valor de resposta mais estável na relação de posicionamento do atuador, porém a resposta entre a posição 200 e posição 800 é mais lenta. No diagrama de penas é possível observar a defasagem apresentada entre o efetivo e o *setpoint*, que é decorrente do tempo que usuário executa a função no supervisório, ou seja, o tempo necessário para o acionamento da função.

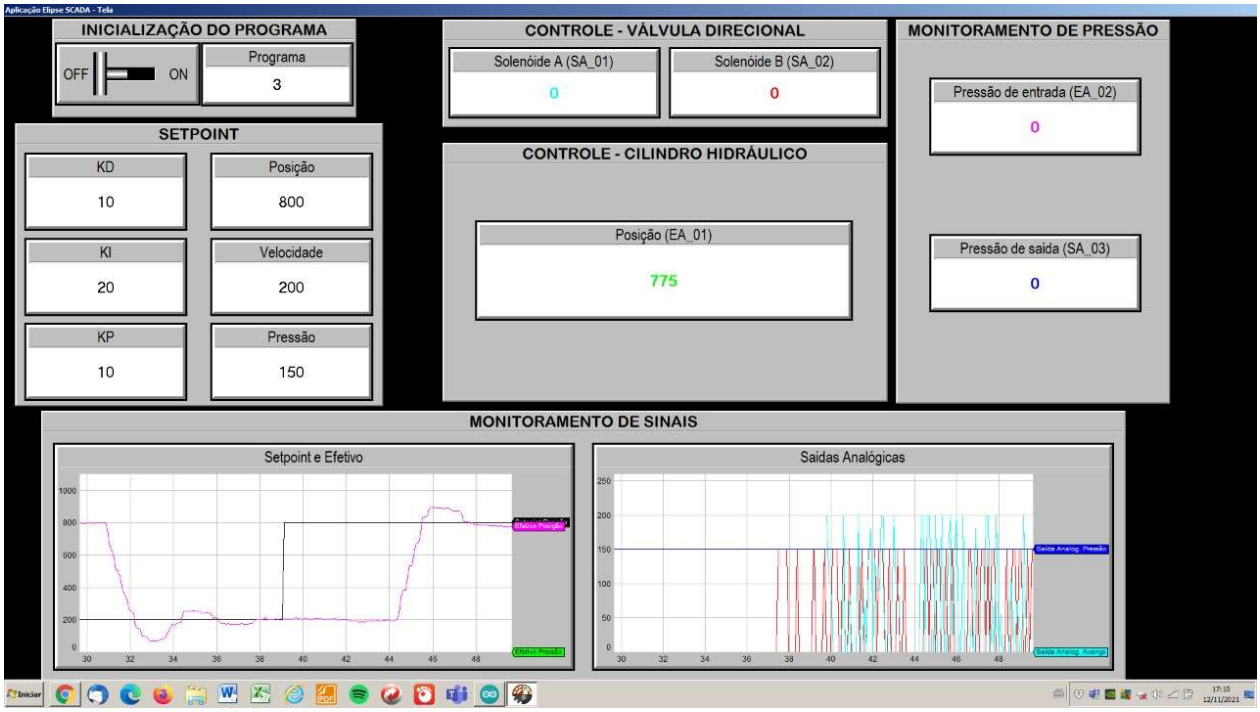
Figura 57 - Desempenho com inserção do fator KD.



Fonte: Elaborado pelo autor.

Com a inserção do fator KI, o tempo de resposta ficou muito lento e com elevada oscilação de resposta, influenciando na instabilidade do atuador. Conforme apresenta a Figura 58.

Figura 58 - Influência do fator KI.



Fonte: Elaborado pelo autor.

O controle de velocidade também foi outro parâmetro que foi alterado, e sua influência acarretam oscilação da resposta de estabilidade do atuador. Comparando os resultados obtidos na Figura 56 e Figura 59, é possível verificar que o tem uma amplitude de oscilação maior, mesmo utilizando os mesmos parâmetros de controle do PID.

Figura 59 - Influência da Velocidade.

Fonte: Elaborado pelo autor.

Os resultados obtidos através dos valores apresentados, mostram a comunicação do supervisório com a programação do Arduino, além de demonstrar o funcionamento no controle de sistemas hidráulicos, possibilitando que sejam implementados em aplicações comerciais.

5.2 *RESULTADOS COM UNIDADE HIDRÁULICA INDUSTRIAL*

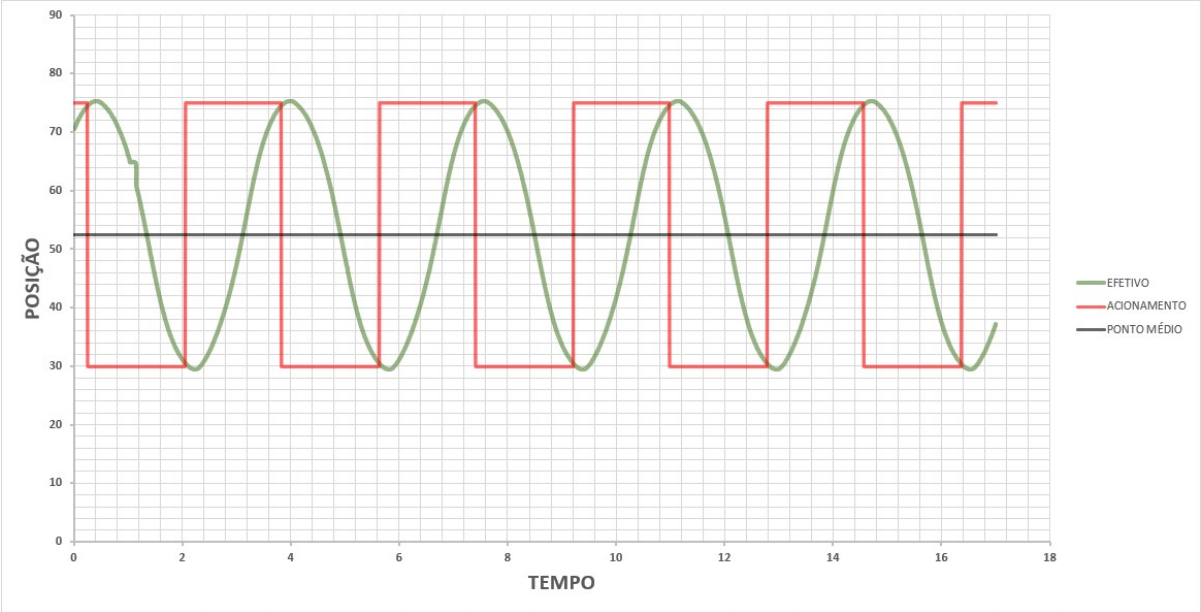
A partir dos resultados obtidos através da utilização da bancada hidráulica educacional, foram desenvolvidos testes de funcionalidade do programa de controle desenvolvido na unidade hidráulica da Rexroth que é uma bancada hidráulica de grande porte e que utiliza para o seu controle o *hardware* VT-HACD. Esta unidade hidráulica contempla válvula direcional com controle proporcional, sensor de posição

fixado no cilindro hidráulico, válvula reguladora de pressão manual e transdutor de pressão.

Antes de repetir os testes que foram aplicados na unidade hidráulica educacional, foram realizados testes com a utilização do CLP VT-HACD. Foi inserido os valores de posição e velocidade, porém com parâmetros de posição aleatórios, não se repetiu os valores de posição inseridos no Arduino, pois o intuito do teste é mostrar que o monitoramento desenvolvido contempla as mesmas informações de resposta do acionamento da válvula direcional.

O ciclo efetuado gerou valores plotados no supervisório de controle do CLP e mostram a resposta do acionamento proporcional. A Figura 60 apresenta o resultado das posições mínima do atuador no valor de 30 e a máxima no valor de 75. Estes valores apresentados são adimensionais e são parâmetros utilizados na programação.

Figura 60 - Monitoramento de sinais gerados pelo CLP.



Fonte: Elaborado pelo autor.

Para o teste realizado foi inserido o programa de controle proporcional de velocidade e posição sendo assim, o controle da válvula direcional e a leitura de posição do cilindro hidráulico. Não foi utilizado o transdutor de pressão para esta

validação, pois o objetivo é apresentar o funcionamento validado nos resultados preliminares em uma unidade hidráulica de aplicações de grande porte.

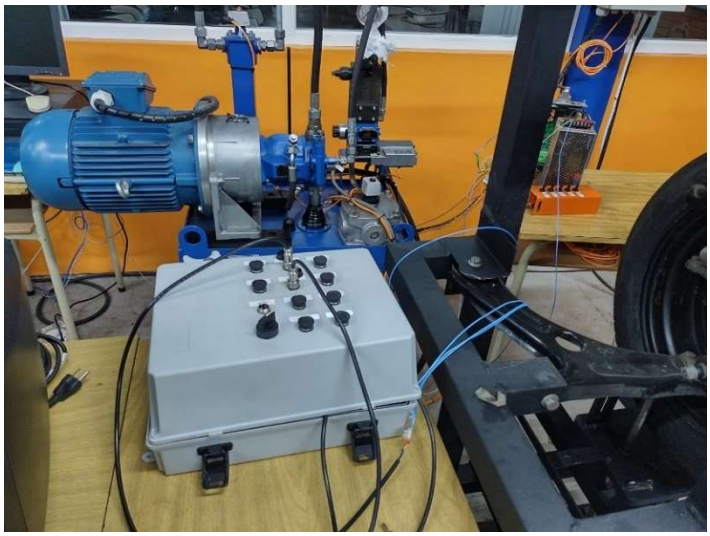
Figura 61 - Unidade Hidráulica Rexroth.

Fonte: Elaborado pelo autor.

A Figura 61 apresenta a unidade hidráulica de grande porte utilizada para validação das aplicações desenvolvidas.

Foi necessário realocar os cabos dos componentes da unidade hidráulica que estão interligados ao CLP e vinculá-los nas entradas e saídas do Arduino. Conforme a Figura 62, Figura 63 e Figura 64.

Figura 62 - *Hardware* de controle.



Fonte: Elaborado pelo autor.

Figura 63 - Entradas e saídas do CLP VT-HACD.

Fonte: Elaborado pelo autor.

Figura 64 - Cilindro hidráulico com sensor de posição.

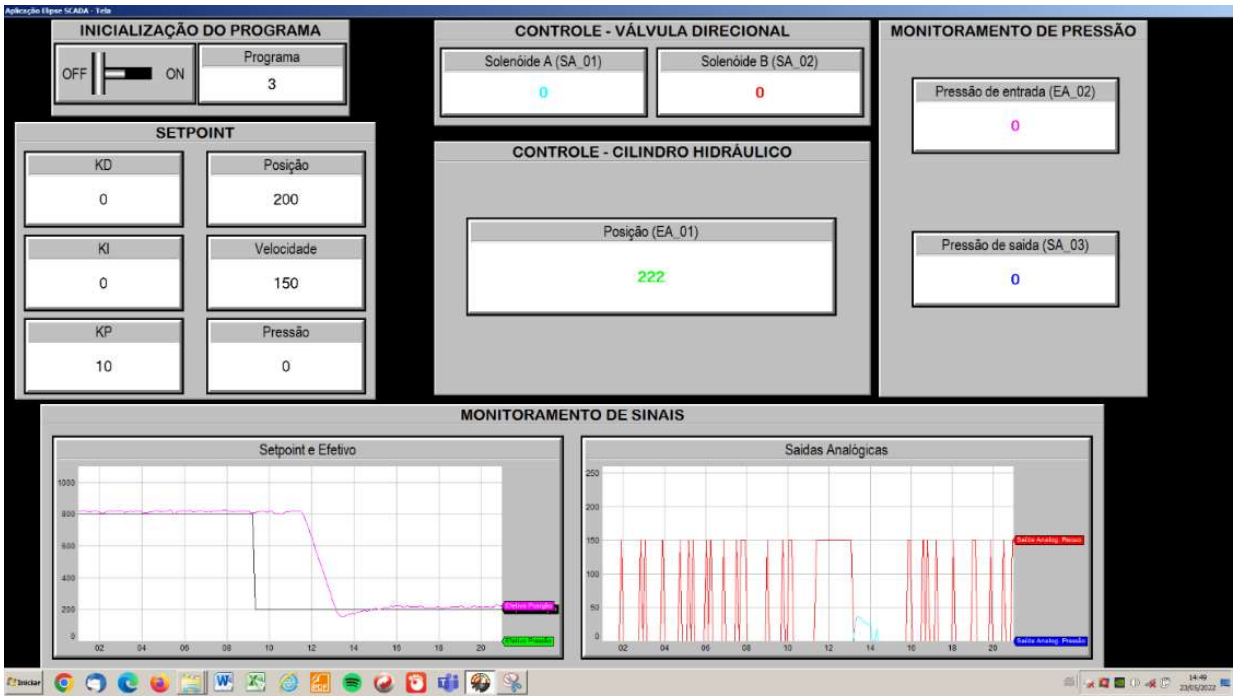
Fonte: Elaborado pelo autor.

Assim como nos resultados obtidos na unidade hidráulica didática, o monitoramento das posições selecionadas para parametrização dos dados de comparação foram, a posição inicial igual a 200, e posição final igual a 800, estes valores são parâmetros do programa e são adimensionais. O programa utilizado para os testes foi o programa número 3.

O programa número 3 possibilita controle de pressão, porém, para os testes esta função não foi utilizada, pois, a válvula de regulagem de pressão é de regulagem manual.

A Figura 65 mostra o resultado obtido com a configuração de KP=10, KI=0, KD=0 e Velocidade=150. As penas de monitoramento de sinais efetivo e *setpoint*, mostram o tempo de resposta entre o valor efetivo (cor preta) e a resposta do solenoide sob influência do ajuste do PID (cor rosa). As penas das saídas analógicas mostram a resposta de acionamento entre as bobinas do solenoide A e solenoide B, assim como apresentado nos resultados preliminares.

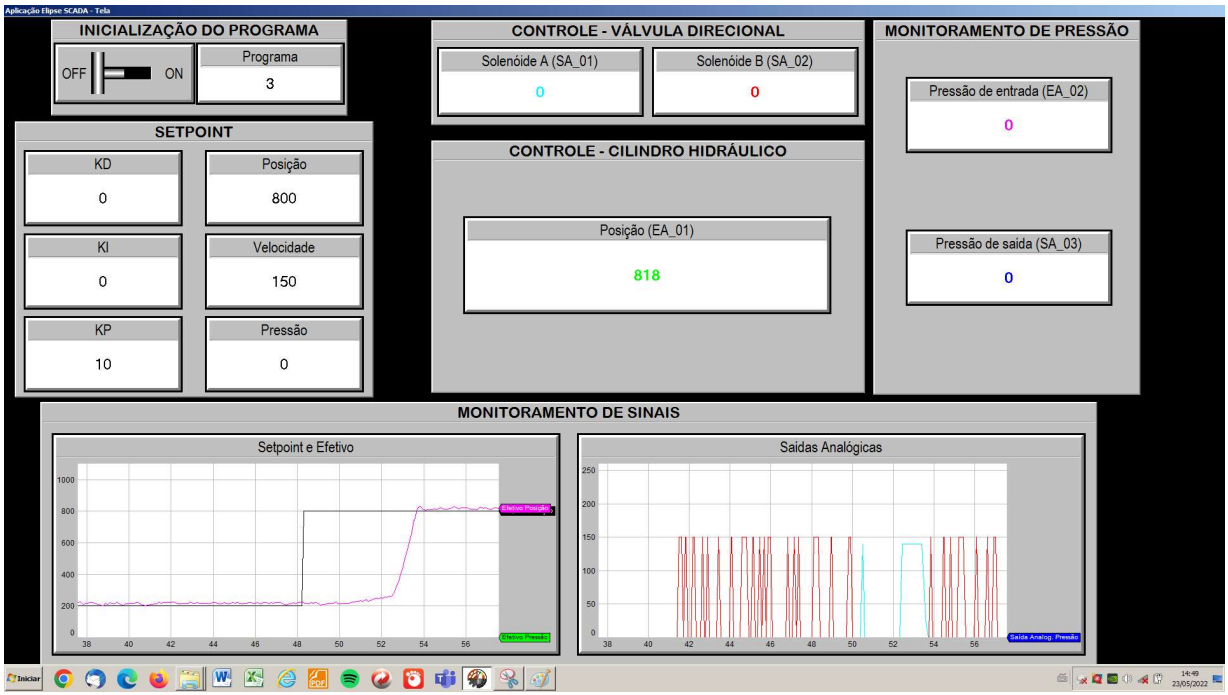
Figura 65 – Acionamento da válvula direcional na posição de recuo.



Fonte: Elaborado pelo autor.

Após o atuador recuar e se manter na posição inicial configurada, foi alterada a configuração de posição para o valor 800 que desloca a haste do atuador para sua posição final com o controle da velocidade, onde, é possível perceber o desempenho da saída analógica (resposta do acionamento das bobinas da válvula direcional) conforme a Figura 66.

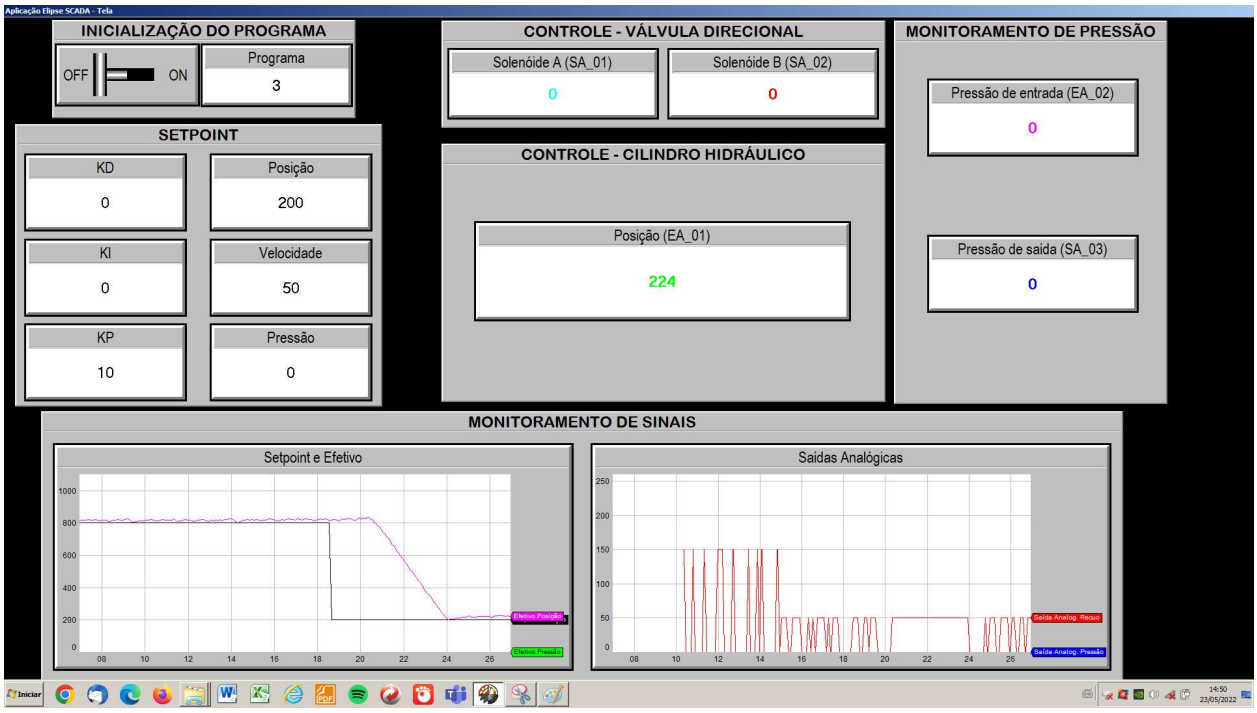
Figura 66 - Acionamento da válvula direcional na posição de avanço.



Fonte: Elaborado pelo autor.

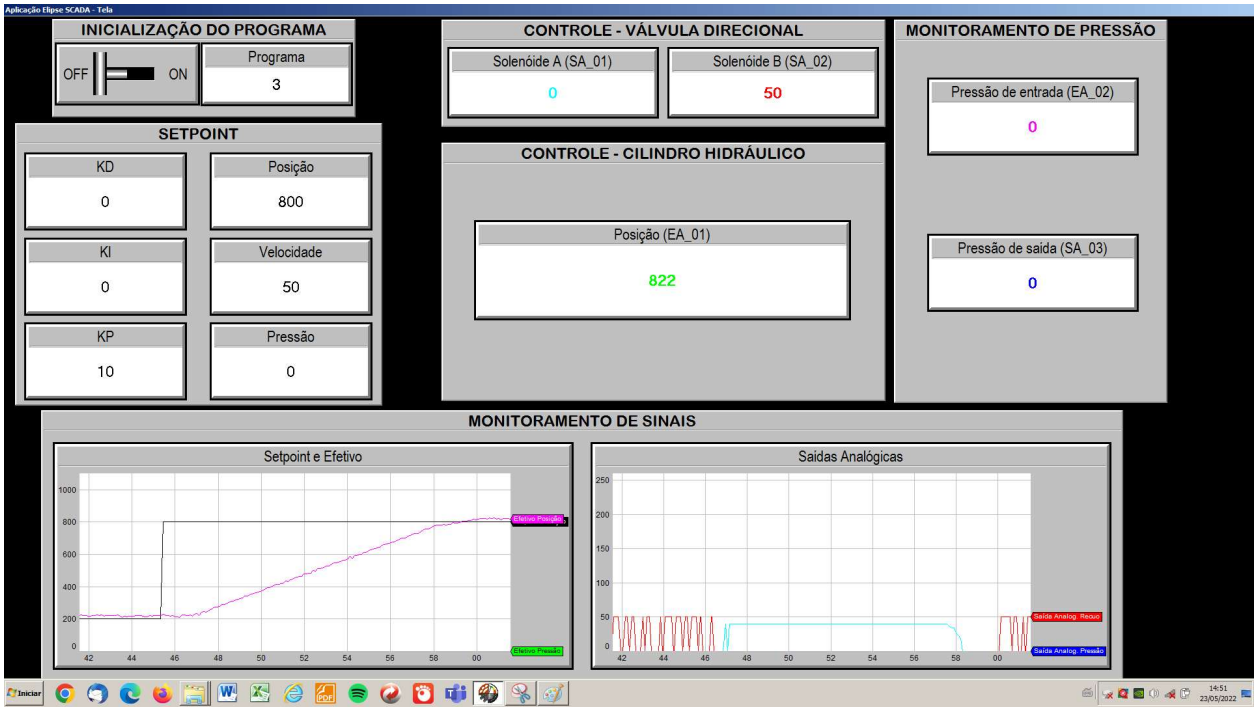
Alterando o parâmetro de velocidade para o valor de 50, foi possível observar a resposta de acionamento entre as bobinas do solenoide A e solenoide B com redução na amplitude dos sinais. Conforme a Figura 67 e Figura 68.

Figura 67 - Posição inicial com alteração da velocidade.



Fonte: Elaborado pelo autor.

Figura 68 - Posição final com alteração da velocidade.



Fonte: Elaborado pelo autor.

Foi possível perceber que tanto na unidade hidráulica didática quanto na unidade industrial, o controle proporcional de posição e velocidade testados foram efetivos, possibilitando o controle e monitoramento. Nas duas condições apresentadas a resposta foi satisfatória.

O sistema pode ser utilizado para substituir CLP com programação para sistemas hidráulicos e os programas desenvolvidos podem atender a diversas aplicações por possibilitar controle de pressão, velocidade e posição.

6. CONCLUSÃO

A elaboração deste trabalho foi fundamental para o aprimoramento dos conhecimentos em automação, linguagens de programação, desenvolvimento de supervisórios além do conhecimento em sistemas hidráulicos. Desenvolver programas automatizados para aplicações hidráulicas agrega desempenho e baixo custo em relação a aplicações com VT-HACD.

O desenvolvimento das 3 propostas de automatização mostra a possibilidade teórica de utilização de sensores para controle preciso de sistemas hidráulicos, reduzindo a geração de energia e calor, tornando-os, assim, mais rentáveis. Porém, devido à limitação estrutural relacionada à falta de componentes, conseguiu-se validar apenas uma das propostas.

O teste de controle proporcional de velocidade e posição comprovou a hipótese de que é possível a automatização de sistemas hidráulicos com custos menores através do uso do *hardware* Arduino.

As maiores dificuldades da proposta desenvolvida foram os programas desenvolvidos em linguagem C++, onde houve necessidade do suporte da orientação para o desenvolvimento das lógicas.

As aplicações desenvolvidas podem introduzir o desenvolvimento de mais funções e aplicações inserindo novas técnicas de programação e desenvolvimento de supervisório. Ao analisar o desempenho do controle de posição e velocidade através dos gráficos do programa validado, é possível visualizar que os parâmetros KP, KI, KD e velocidade podem estar sendo influenciados pelo tempo de resposta do controle proporcional. Porém são necessários mais estudos para comprovar tal achado.

Os objetivos iniciais propostos foram atendidos com o desenvolvimento do *hardware* utilizando Arduino, as programações voltadas para o controle de sistemas hidráulicos com acionamento proporcional, e foram desenvolvidas as interfaces de controle e monitoramento dos programas. Os resultados foram validados através de testes de funcionamento com a utilização de duas opções de bancadas hidráulicas, uma unidade hidráulica de porte industrial e a unidade hidráulica de porte educacional.

6.1 *SUGESTÕES DE TRABALHOS FUTUROS*

A fim de expandir o estudo a respeito de controle proporcional e parâmetros PID e somar com o estudo realizado no presente trabalho, sugere-se os seguintes estudos:

a) Instrumentação dos componentes para obter melhor desempenho e resultados;

b) Influência dos parâmetros PID na resposta de acionamento;

c) Obter o melhor ajuste e controle através do comparativo dos parâmetros PID e velocidade;

REFERÊNCIAS

ARDUINO. **What is Arduino.** Disponível em: <https://docs.arduino.cc/foundations/basics/whats-arduino>. Acesso em: 10 nov. 2021.

FIALHO, Arivelto Bustamante. **Automação hidráulica – Projetos, Dimensionamento e Análise de Circuitos.** 7. ed. São Paulo: Editora Érica. 2019.

FIALHO, Arivelto Bustamante. **Automatismos hidráulicos – Princípios básicos, dimensionamentos de componentes e aplicações práticas.** 1. ed. São Paulo: Editora Érica. 2015.

FILHO, Elmo Souza Dutra da S., SANTOS, Bruna Karine D. **Sistemas hidráulicos e pneumáticos.** 1 ed. Porto Alegre: Editora Sagah. 2018.

GEORGINI, Marcelo. **Automação aplicada: descrição e implementação de sistemas sequenciais com PLCs.** 9. ed. São Paulo: Editora Érica. 2016.

MORAES, Cícero Couto de., CASTRUCCI, Plínio. **Engenharia de automação industrial.** 2. ed. Rio de Janeiro: Editora LTC. 2007.

PETRUZELLA, Frank D. **Controladores Lógicos Programáveis.** 4 ed. Porto Alegre: Editora AMGH. 2013.

SCHEFFER, Eduardo., NERY, Eduardo G., SEIXAS, Jordana L. **Teoria de controle e servomecanismo.** 1 ed. Porto Alegre: Editora Sagah. 2018.

SILVA, Edilson Alfredo da. **Introdução às linguagens de programação para CLP.** 1. ed. São Paulo: Editora Blucher. 2016.

SILVA, Edival Alves da. **Desenvolvimento de sistema hidráulico didático de controle proporcional para o laboratório de hidráulica DAMEC-CT**. 2015. Trabalho de Conclusão de Curso. Universidade Tecnológica Federal do Paraná.

SILVEIRA, Paulo Rogério da., SANTOS, Winderson E. dos. **Automação e controle discreto.** 9. ed. São Paulo: Editora Érica. 1998.

SOUZA, Vitor Amadeu**. O protocolo modbus**. Available in:< http://www. cerne-tec. com. br/Modbus. pdf> Acesso, v. 11, 2008.

STEVAN JUNIOR, Sergio Luiz., SILVA, Rodrigo Adamshuk. **Automação e instrumentação industrial com Arduino**. 1. ed. São Paulo: Editora Érica. 2015.

STRACK, Guilherme. **Módulo de I/O remoto MODBUS**. 2011.

KALATEC. **Arduino: o que é, para que serve, como funciona e tipos**. Disponível em: <https://blog.kalatec.com.br/arduino-o-que-e/>. Acesso em: 18 abr. 2022.

ANEXO A – PROGRAMA PARA CONTROLE DE PRESSÃO E VELOCIDADE

```
#include <Modbusino.h>
#include <PID_v2.h>

ModbusinoSlave modbusino_slave(1);

PID_v2 myPID(5.0,0.0,0.0, PID::Direct);

uint16_t tab_reg[20];

unsigned long millisTarefa1 = millis();


const int T_SD_01 = 0;
const int T_SD_02 = 1;
const int T_ED_01 = 2;
const int T_ED_02 = 3;
const int T_SA_01 = 4;
const int T_SA_02 = 5;
const int T_SA_03 = 6;
const int T_EA_01 = 7;
const int T_EA_02 = 8;
const int T_EA_03 = 9;
const int T_EA_04 = 10;
const int NUM_PROG = 11;
const int SP_POS = 12;
const int SP_VEL = 13;
const int SP_KP = 14;
const int SP_KI = 15;
const int SP_KD = 16;
const int SP_PR = 17;
const int BOT_START = 19;


const int SAIDA_01 = 10;
const int SAIDA_02 = 11;
const int ENTR_01 = 2;
const int ENTR_02 = 4;
const int SAIDA_ANALOG_01 = 3;
const int SAIDA_ANALOG_02 = 5;
const int SAIDA_ANALOG_03 = 6;
const int ENTR_ANALOG_01 = 0;
const int ENTR_ANALOG_02 = 1;
const int ENTR_ANALOG_03 = 2;
const int ENTR_ANALOG_04 = 3;


void setup()
{

   modbusino_slave.setup(115200);

   pinMode(SAIDA_01,OUTPUT);
   pinMode(SAIDA_02,OUTPUT);
   pinMode(SAIDA_ANALOG_01,OUTPUT);
   pinMode(SAIDA_ANALOG_02,OUTPUT);
```

```
    pinMode(SAIDA_ANALOG_03,OUTPUT);
    pinMode(ENTR_01,INPUT);
    pinMode(ENTR_02,INPUT);

    myPID.Start(100,0,100);
}

void loop()
{
     double saida_auxiliar;

     tab_reg[T_EA_01] = analogRead(ENTR_ANALOG_01);
     tab_reg[T_EA_02] = analogRead(ENTR_ANALOG_02);
     tab_reg[T_EA_03] = analogRead(ENTR_ANALOG_03);
     tab_reg[T_EA_04] = analogRead(ENTR_ANALOG_04);

     tab_reg[T_ED_01] = digitalRead(ENTR_01);
     tab_reg[T_ED_01] = digitalRead(ENTR_01);

     analogWrite(SAIDA_ANALOG_01,tab_reg[T_SA_01]);
     analogWrite(SAIDA_ANALOG_02,tab_reg[T_SA_02]);
     analogWrite(SAIDA_ANALOG_03,tab_reg[T_SA_03]);

     digitalWrite(SAIDA_01,logic(tab_reg[T_SD_01]));
     digitalWrite(SAIDA_02,logic(tab_reg[T_SD_02]));

     if (tab_reg[NUM_PROG]==3)
     {

          tab_reg[T_SA_03]=tab_reg[SP_PR];

          if (logic(tab_reg[BOT_START]))
          {

            myPID.Start( tab_reg[T_EA_01],0, tab_reg[SP_POS]);
            double aux1 = float(tab_reg[SP_KP]) /10;
            double aux2 = float(tab_reg[SP_KI]) /10;
            double aux3 = float(tab_reg[SP_KD]) /10;
            myPID.SetTunings( aux1, aux2, aux3);
            myPID.SetOutputLimits(-255, 255);
            myPID.SetSampleTime(20);

          }

          const double output_PID = myPID.Run(tab_reg[T_EA_01]);


           if (output_PID<20 && output_PID>-20 )
           {
             tab_reg[T_SA_01]=0;
             tab_reg[T_SA_02]=0;
           }

           if (output_PID>20)
           {

              saida_auxiliar = output_PID;
              if (saida_auxiliar>230)
              {
                 saida_auxiliar=230;
```

```
            }


            if (saida_auxiliar > tab_reg[SP_VEL]-25)
            {
              saida_auxiliar = tab_reg[SP_VEL]-25;
            }

            tab_reg[T_SA_01]=saida_auxiliar+25;

            tab_reg[T_SA_02]=0;
          }

          if (output_PID<-20)
          {

            tab_reg[T_SA_01]=0;


            saida_auxiliar = output_PID;
            if (saida_auxiliar<-230)
            {
              saida_auxiliar=-230;
            }

            if (saida_auxiliar < -tab_reg[SP_VEL]+25)
            {
              saida_auxiliar = - tab_reg[SP_VEL]+25;
            }
            tab_reg[T_SA_02]=-saida_auxiliar+25;
          }


    }

    else
    {
        tab_reg[T_SA_01]=0;
        tab_reg[T_SA_02]=0;
        tab_reg[T_SA_03]=0;
        tab_reg[T_SD_01]=0;
        tab_reg[T_SD_02]=0;
    }



    //if((millis() - millisTarefa1) > 50)
    //{
      // millisTarefa1 = millis();
      // modbusino_slave.loop(tab_reg, 20);
    // }
     modbusino_slave.loop(tab_reg, 20);
}

bool logic(int valor)
{
  if (valor>=1) return HIGH;
  else return LOW;
}
```

```
int escala(int valor,int val_in, int val_final)
{
  double aux1 = float(valor);
  double aux2 = float(val_in);
  double aux3 = float(val_final);

  double result = ((aux1/255) * (aux3-aux2)) + aux2;
```

ANEXO B – PROGRAMA PARA CONTROLE DE PRESSÃO

```
#include <Modbusino.h>

ModbusinoSlave modbusino_slave(1);

uint16_t tab_reg[20];

unsigned long millisTarefa1 = millis();


const int SAIDA_01 = 10;
const int SAIDA_02 = 11;
const int ENTR_01 = 2;
const int ENTR_02 = 4;
const int SAIDA_ANALOG_01 = 3;
const int SAIDA_ANALOG_02 = 5;
const int SAIDA_ANALOG_03 = 6;
const int ENTR_ANALOG_01 = 0;
const int ENTR_ANALOG_02 = 1;
const int ENTR_ANALOG_03 = 2;
const int ENTR_ANALOG_04 = 3;

const int T_SD_01 = 0;
const int T_SD_02 = 1;
const int T_ED_01 = 2;
const int T_ED_02 = 3;
const int T_SA_01 = 4;
const int T_SA_02 = 5;
const int T_SA_03 = 6;
const int T_EA_01 = 7;
const int T_EA_02 = 8;
const int T_EA_03 = 9;
const int T_EA_04 = 10;
const int NUM_PROG = 11;
const int SP_POS = 12;
const int SP_VEL = 13;
const int SP_KP = 14;
const int SP_KI = 15;
const int SP_KD = 16;
const int SP_PR = 17;
const int BOT_START = 19;

void setup()
{
   modbusino_slave.setup(115200);

   pinMode(SAIDA_01,OUTPUT);
   pinMode(SAIDA_02,OUTPUT);
   pinMode(SAIDA_ANALOG_01,OUTPUT);
   pinMode(SAIDA_ANALOG_02,OUTPUT);
   pinMode(SAIDA_ANALOG_03,OUTPUT);
   pinMode(ENTR_01,INPUT);
   pinMode(ENTR_02,INPUT);

}
```

```
void loop()
{

    tab_reg[T_EA_01] = analogRead(ENTR_ANALOG_01);
    tab_reg[T_EA_02] = analogRead(ENTR_ANALOG_02);
    tab_reg[T_EA_03] = analogRead(ENTR_ANALOG_03);
    tab_reg[T_EA_04] = analogRead(ENTR_ANALOG_04);

    tab_reg[T_ED_01] = digitalRead(ENTR_01);
    tab_reg[T_ED_01] = digitalRead(ENTR_01);

    analogWrite(SAIDA_ANALOG_01,tab_reg[T_SA_01]);
    analogWrite(SAIDA_ANALOG_02,tab_reg[T_SA_02]);
    analogWrite(SAIDA_ANALOG_03,tab_reg[T_SA_03]);

    digitalWrite(SAIDA_01,logic(tab_reg[T_SD_01]));
    digitalWrite(SAIDA_02,logic(tab_reg[T_SD_02]));


    if (tab_reg[NUM_PROG]==2)
    {

        if (logic(tab_reg[BOT_START]))
        {

          tab_reg[T_SA_01]=tab_reg[SP_PR];
        }
        else
        {
           tab_reg[T_SA_01]=0;
        }


    }

     //if((millis() - millisTarefa1) > 200)
     //{
     //   millisTarefa1 = millis();
     //   modbusino_slave.loop(tab_reg, 20);
     //}
     modbusino_slave.loop(tab_reg, 20);
}

bool logic(int valor)
{
  if (valor>=1) return HIGH;
  else return LOW;
}


int escala(int valor,int val_in, int val_final)
{
  double aux1 = float(valor);
  double aux2 = float(val_in);
  double aux3 = float(val_final);

  double result = ((aux1/255) * (aux3-aux2)) + aux2;

  return int(result);
}
```

ANEXO C – PROGRAMA PARA CONTROLE DE VELOCIDADE

```
#include <Modbusino.h>
#include <PID_v2.h>

ModbusinoSlave modbusino_slave(1);

PID_v2 myPID(5.0,0.0,0.0, PID::Direct);

uint16_t tab_reg[20];

unsigned long millisTarefa1 = millis();

const int SAIDA_01 = 10;
const int SAIDA_02 = 11;
const int ENTR_01 = 2;
const int ENTR_02 = 4;
const int SAIDA_ANALOG_01 = 3;
const int SAIDA_ANALOG_02 = 5;
const int SAIDA_ANALOG_03 = 6;
const int ENTR_ANALOG_01 = 0;
const int ENTR_ANALOG_02 = 1;
const int ENTR_ANALOG_03 = 2;
const int ENTR_ANALOG_04 = 3;

const int T_SD_01 = 0;
const int T_SD_02 = 1;
const int T_ED_01 = 2;
const int T_ED_02 = 3;
const int T_SA_01 = 4;
const int T_SA_02 = 5;
const int T_SA_03 = 6;
const int T_EA_01 = 7;
const int T_EA_02 = 8;
const int T_EA_03 = 9;
const int T_EA_04 = 10;
const int NUM_PROG = 11;
const int SP_POS = 12;
const int SP_VEL = 13;
const int SP_KP = 14;
const int SP_KI = 15;
const int SP_KD = 16;

const int BOT_START = 19;


void setup()
{
      modbusino_slave.setup(115200);

   pinMode(SAIDA_01,OUTPUT);
   pinMode(SAIDA_02,OUTPUT);
   pinMode(SAIDA_ANALOG_01,OUTPUT);
   pinMode(SAIDA_ANALOG_02,OUTPUT);
   pinMode(SAIDA_ANALOG_03,OUTPUT);
   pinMode(ENTR_01,INPUT);
   pinMode(ENTR_02,INPUT);
```

```
   myPID.Start(100,0,100);


}

void loop()
{

    double saida_auxiliar;

    tab_reg[T_EA_01] = analogRead(ENTR_ANALOG_01);
    tab_reg[T_EA_02] = analogRead(ENTR_ANALOG_02);
    tab_reg[T_EA_03] = analogRead(ENTR_ANALOG_03);
    tab_reg[T_EA_04] = analogRead(ENTR_ANALOG_04);

    tab_reg[T_ED_01] = digitalRead(ENTR_01);
    tab_reg[T_ED_01] = digitalRead(ENTR_01);

    analogWrite(SAIDA_ANALOG_01,tab_reg[T_SA_01]);
    analogWrite(SAIDA_ANALOG_02,tab_reg[T_SA_02]);
    analogWrite(SAIDA_ANALOG_03,tab_reg[T_SA_03]);

    digitalWrite(SAIDA_01,logic(tab_reg[T_SD_01]));
    digitalWrite(SAIDA_02,logic(tab_reg[T_SD_02]));


    if (tab_reg[NUM_PROG]==1)
    {

        if (logic(tab_reg[BOT_START]))
        {

          myPID.Start( tab_reg[T_EA_01],0, tab_reg[SP_POS]); // EFETIVO, 0, SETPOINT
          double aux1 = float(tab_reg[SP_KP]) /10;
          double aux2 = float(tab_reg[SP_KI]) /10;
          double aux3 = float(tab_reg[SP_KD]) /10;
          myPID.SetTunings( aux1, aux2, aux3);
          myPID.SetOutputLimits(-255, 255);

          myPID.SetSampleTime(20);


          //tab_reg[T_SD_01]=1;
          //digitalWrite(SAIDA_01,logic(tab_reg[T_SD_01]));
          //delay(200);
          //tab_reg[T_SD_01]=0;
          //digitalWrite(SAIDA_01,logic(tab_reg[T_SD_01]));

        }


        const double output_PID = myPID.Run(tab_reg[T_EA_01]);

        if (output_PID==0)
         {
           tab_reg[T_SA_01]=0;
           tab_reg[T_SA_02]=0;
         }

         /*
```

```
            if (saida_auxiliar > tab_reg[SP_VEL])
            {
              saida_auxiliar = tab_reg[SP_VEL];
            }

            if (saida_auxiliar < -tab_reg[SP_VEL])
            {
              saida_auxiliar = -tab_reg[SP_VEL];
            }
            */

            if (output_PID>0)
            {

              saida_auxiliar = output_PID;
              if (saida_auxiliar > tab_reg[SP_VEL])
              {
                saida_auxiliar = tab_reg[SP_VEL];
              }

              tab_reg[T_SA_01]=saida_auxiliar;

              tab_reg[T_SA_02]=0;
            }

            if (output_PID<0)
            {

              tab_reg[T_SA_01]=0;

              saida_auxiliar = output_PID;
              if (saida_auxiliar < -tab_reg[SP_VEL])
              {
                saida_auxiliar = - tab_reg[SP_VEL];
              }
              tab_reg[T_SA_02]=-saida_auxiliar;
            }

        }


        /*if (tab_reg[NUM_PROG]==0)
        {
              tab_reg[T_SA_01]=0;
              tab_reg[T_SA_02]=0;
              tab_reg[T_SA_03]=0;
              tab_reg[T_SD_01]=0;
              tab_reg[T_SD_02]=0;
        }*/


         /* if((millis() - millisTarefa1) > 100)
         {
           millisTarefa1 = millis();
           modbusino_slave.loop(tab_reg, 20);
         }*/
         modbusino_slave.loop(tab_reg, 20);
}

bool logic(int valor)
```

```
{
  if (valor>=1) return HIGH;
  else return LOW;
}


int escala(int valor,int val_in, int val_final)
{
  double aux1 = float(valor);
  double aux2 = float(val_in);
  double aux3 = float(val_final);

  double result = ((aux1/255) * (aux3-aux2)) + aux2;

  return int(result);
}
```