Universidade Federal de Campina Grande
Centro de Engenharia Elétrica e Informática
Departamento de Engenharia Elétrica

# Trabalho de Conclusão de Curso

## Desenvolvimento da Programação do Controle de uma Planta Industrial por CLP

Douglas de Souza Almeida

Campina Grande - PB

Maio de 2024

Douglas de Souza Almeida

# Desenvolvimento da Programação do Controle de uma Planta Industrial por CLP

Trabalho de Conclusão de Curso submetido à Coordenaria de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do Grau de Engenheiro Eletricista.

Universidade Federal de Campina Grande - UFCG

Centro de Engenharia Elétrica e Informática - CEEI

Departamento de Engenharia Elétrica - DEE

Coordenação de Graduação em Engenharia Elétrica - CGEE

Péricles Rezende Barros, Dr.

(Orientador)

Campina Grande - PB

Maio de 2024

Douglas de Souza Almeida

# Desenvolvimento da Programação do Controle de uma Planta Industrial por CLP

*Trabalho de Conclusão de Curso submetido à Coordenaria de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do Grau de Engenheiro Eletricista.*

Aprovado em _____ /_____ /_____

---

**Professor Avaliador**
Universidade Federal de Campina Grande
Avaliador

---

**Péricles Rezende Barros**
Universidade Federal de Campina Grande
Orientador

Campina Grande - PB
Maio de 2024

*"Dedico este trabalho aos meus pais, Arquiles da Silva Almeida e Rosicleia de Souza Machado Almeida*

# Agradecimentos

Gostaria de expressar minha sincera gratidão às pessoas que estiveram ao meu lado nesta jornada:

Aos meus amados pais, Arquiles e Rose, e aos meus irmãos, Kauã e Maria Eduarda, cujo apoio e presença constante foram fundamentais para eu alcançar este marco em minha vida.

À minha incrível namorada, cujo apoio e compreensão iluminou cada passo do meu caminho.

À minha família, que sempre esteve ao meu lado, apoiando-me incondicionalmente e dando-me forças para perseguir meus sonhos, independentemente dos desafios que enfrentei.

À equipe do LIEC, por fornecer o ambiente de pesquisa e os recursos necessários para a execução das atividades, especialmente à Anna, cuja colaboração foi vital para o desenvolvimento deste trabalho. Seu apoio e orientação foram inestimáveis.

Agradeço ao meu professor orientador, Péricles, pela oportunidade oferecida, por compartilhar seu conhecimento e pela orientação que guiou este trabalho para o sucesso.

A todos os meus colegas, professores e funcionários que ao longo desses anos contribuíram de maneiras diversas para o meu crescimento pessoal e profissional. Suas influências foram cruciais para minha formação.

*"Quanto mais as pessoas acreditam em alguma coisa,*
*quanto mais se dedicam à ela,*
*mais podem influenciar o seu acontecimento.",*
*(Bernardinho)*

# Resumo

Este trabalho apresenta o desenvolvimento de uma programação para uma planta industrial em escala laboratorial. Inicialmente programada em linguagem *ladder*, a planta foi posteriormente convertida para diagrama de blocos de função e texto estruturado. A planta é composta por quatro tanques interligados, abastecidos por um reservatório central de água e controlados por um CLP Allen Bradley. Os ganhos do controlador PI foram ajustados utilizando o software *BR-Tuning*, e diferentes técnicas de sintonia foram comparadas em termos de desempenho do sistema.

**Palavras-chave:** Programação de CLP, Controle de Processos, Sintonização de Controlador, Planta Industrial, *Ladder*, Diagrama de Blocos de Função, Texto Estruturado.

# Abstract

This work presents the development of programming for a laboratory-scale industrial plant. Initially programmed in ladder logic, the plant was later converted to function block diagram and structured text. The plant consists of four interconnected tanks, supplied by a central water reservoir, and controlled by an Allen Bradley PLC. The PI controller gains were adjusted using BR-Tuning software, and different tuning techniques were compared in terms of system performance.

**Keywords:** PLC Programming, Process Control, Controller Tuning, Industrial Plant, Ladder, Function Block Diagram, Structured Text.

# Lista de ilustrações


Figura 1 – Estrutura básica do CLP . . . . . 3
Figura 2 – Ciclo de varredura do CLP . . . . . 4
Figura 3 – Exemplo do código em FBD . . . . . 8
Figura 4 – Exemplo Gráfico de um *Set* e *Reset* em Ladder . . . . . 9
Figura 5 – Exemplo Gráfico de um contato de selo em Ladder . . . . . 10
Figura 6 – Bloco de função personalizado em lista de instruções que reinicia os valores dos temporizadores para zero . . . . . 11
Figura 7 – SFC utilizando Texto Estruturado no elemento de transição no RSLogix 5000 . . . . . 12
Figura 8 – Diagrama de blocos do sistema de controle PID em malha fechada . . . 14
Figura 9 – Planta didática com tanques de nível . . . . . 16
Figura 10 – Esquema de conexão dos componentes . . . . . 16
Figura 11 – CLP utilizado na planta . . . . . 17
Figura 12 – Modelo de tanque presente na planta didática . . . . . 18
Figura 13 – Válvulas elétricas utilizadas na planta . . . . . 19
Figura 14 – Motor-bomba usado na planta . . . . . 19
Figura 15 – Transmissor utilizado na planta . . . . . 20
Figura 16 – Inversores utilizados na planta . . . . . 20
Figura 17 – Painel completo da planta . . . . . 21
Figura 18 – Rotina Principal . . . . . 22
Figura 19 – BBA_CONTROL em *ladder* . . . . . 23
Figura 20 – BBA_CONTROL em FBD . . . . . 24
Figura 21 – BBA_CONTROL em ST . . . . . 25
Figura 22 – BBA_CONTROL em *ladder* . . . . . 26
Figura 23 – BBA_CONTROL em FBD . . . . . 27
Figura 24 – SEGURANCA em LD . . . . . 29
Figura 25 – SEGURANCA em FBD . . . . . 30
Figura 26 – VALV em LD . . . . . 32
Figura 27 – VALV em FBD . . . . . 33
Figura 28 – VALV em FBD . . . . . 34
Figura 29 – MOD_OPER em LD . . . . . 35
Figura 30 – MOD_OPER em LD . . . . . 36
Figura 31 – MOD_OPER em FBD . . . . . 37
Figura 32 – LIC06 em LD . . . . . 38
Figura 33 – LIC06 em FBD . . . . . 39
Figura 34 – LIC06 em FBD . . . . . 40

Figura 35 – LIC06 em FBD . . . . . 41
Figura 36 – LIC06 em ST . . . . . 42
Figura 37 – Variáveis do processo em cada técnica . . . . . 45
Figura 38 – Variáveis manipuladas em cada técnica . . . . . 46
Figura 39 – Visão Geral do BR-Tuning . . . . . 53

# Lista de tabelas

Tabela 1 – Operadores padrões . . . . . 6
Tabela 2 – Declarações do ST . . . . . 7
Tabela 3 – Descrição das *tags* . . . . . 23
Tabela 4 – Descrição das Tags da sub-rotina SEGURANCA . . . . . 28
Tabela 5 – Ganhos do Controlador PI . . . . . 44
Tabela 6 – Medidas de desempenho para cada técnica . . . . . 47

# Lista de abreviaturas e siglas

| | |
|---|---|
| CLP | Controlador Lógico Programável |
| LIEC | Laboratório de Instrumentação Eletrônica e Controle |
| IL | *Instruction List* |
| ST | *Structured Text* |
| SFC | *Sequential Function Chart* |
| LD | *Ladder Diagram* |
| FBD | *Function Block Diagram* |
| HP | *Horsepower* |
| OPC | *Open Platform Communications* |
| IHM | Interface Homem-Máquina |
| VVC | *Voltage Vector Control* |
| POU | *Program Organisation Unit* |
| MV | *Program Organisation Unit* |
| DDC | *Direct Digital Control* |
| SP | *Setpoint* |
| PV | *Process Variable* |
| MV | *Manipulated Variable* |
| IAE | *Integral Absolute Error* |
| ZF | Zak-Friedman |
| KO | Kano-Ogawa |

# Lista de símbolos

| | |
|---|---|
| $m$ | Metros |
| C | *Celsius* |
| V | *Volt* |
| $\pi$ | Pi |
| h | Altura |
| A | *Ampere* |
| kg | *Kilograma* |

# Sumário


**1 INTRODUÇÃO . . . . . 1**
**1.1 Motivação . . . . . 1**
**1.2 Objetivos . . . . . 2**
1.2.1 Objetivos Gerais . . . . . 2
1.2.2 Objetivos Específicos . . . . . 2
**1.3 Organização do Trabalho . . . . . 2**

**2 FUNDAMENTAÇÃO TEÓRICA . . . . . 3**
**2.1 Controlador Lógico Programável . . . . . 3**
2.1.1 Texto Estruturado . . . . . 5
2.1.2 Diagrama de Blocos de Função . . . . . 8
2.1.3 Diagrama Ladder . . . . . 9
2.1.4 Lista de Instruções . . . . . 10
2.1.5 Sequenciamento Gráfico de Funções . . . . . 12
**2.2 Controlador PID . . . . . 13**

**3 DESENVOLVIMENTO . . . . . 15**
**3.1 Descrição da Planta . . . . . 15**
**3.2 Programação . . . . . 21**
**3.3 Identificação e Controle . . . . . 43**

**4 RESULTADOS E DISCUSSÕES . . . . . 44**

**5 CONCLUSÃO . . . . . 49**
**5.1 Trabalhos futuros . . . . . 49**

**REFERÊNCIAS BIBLIOGRÁFICAS . . . . . 51**

**ANEXO A – BR-TUNING . . . . . 53**

# 1 Introdução

Controladores Lógicos Programáveis (CLPs) emergem como dispositivos primordiais na implementação e controle de sistemas automatizados. Os CLPs são instrumentos de estado sólido da família dos computadores, utilizando circuitos integrados em vez de dispositivos eletromecânicos para implementar funções de controle. Eles são capazes de armazenar instruções, como sequenciamento, temporização, contagem, aritmética, manipulação de dados e comunicação para controlar máquinas e processos industriais. São usados não apenas no controle de variáveis de processo, garantindo a precisão e a eficiência das operações industriais, mas também no intertravamento de sistemas, assegurando a segurança e a integridade das operações industriais. (BRYAN; BRYAN, 1997).

Além disso, os CLPs são essenciais na otimização de processos industriais, proporcionando eficiência e confiabilidade na execução de tarefas sequenciais e na sincronização de processos e elementos auxiliares em diferentes setores, como manufatura, química e processos. Ademais, o uso de CLPs tem impacto direto na redução de custos, especialmente em sistemas de controle complexos. Atualmente, a maioria dos elementos de controle usados para executar a lógica do sistema foi substituída pelos CLPs (NAMEKAR; YADAV, 2020).

A programação eficaz do controle de uma planta industrial por CLP envolve a aplicação de algoritmos e lógicas de controle para garantir a estabilidade do processo, a precisão na medição e a resposta adequada a perturbações externas.

Este trabalho será fundamentado pelo padrão IEC 61131 (TIEGELKAMP; JOHN, 2010), que estabelece os padrões para a programação de sistemas de automação industrial. Este padrão fornece diretrizes e recomendações para a implementação de lógica de controles em CLP. A partir dessas diretrizes, será desenvolvida a programação de uma lógica de controle para uma planta industrial em escala laboratorial através do CLP presente no Laboratório de Instrumentação Eletrônica e Controle (LIEC).

## 1.1 Motivação

A implementação bem-sucedida de sistemas automatizados requer conhecimento técnico especializado e habilidades avançadas de programação. Nesse contexto, o uso do padrão IEC 61131-3 oferece uma estrutura padronizada para o desenvolvimento de *software* de controle, facilitando a interoperabilidade entre diferentes sistemas e garantindo a qualidade e confiabilidade do código.

## 1.2 Objetivos

Esta seção apresenta os objetivos gerais e específicos do trabalho necessários para a conclusão deste projeto.

### 1.2.1 Objetivos Gerais

Este trabalho tem como objetivo geral desenvolver a programação da lógica de controle de uma planta industrial em escala laboratorial por CLP e adaptar o controlador da malha de controle, conforme os padrões estabelecidos pelo padrão IEC 61131.

### 1.2.2 Objetivos Específicos

- Estudar os princípios e conceitos fundamentais do padrão IEC 61131-3, com foco nas linguagens de programação *ladder*, diagrama de blocos funcionais e texto estruturado;
- Analisar os requisitos específicos da planta de nível e identificar as funcionalidades e estratégias de controle necessárias para sua programação;
- Desenvolver a programação de uma planta de nível utilizando as linguagens de programação mencionadas, seguindo os padrões estabelecidos pela norma;
- Sintonizar o controlador da malha de controle;
- Realizar testes para validação da programação.

## 1.3 Organização do Trabalho

Este documento está estruturado em cinco capítulos, cada um abordando aspectos específicos do projeto. No primeiro capítulo, são apresentados a definição do problema e os objetivos do trabalho a ser desenvolvido. O segundo capítulo trata da fundamentação teórica e revisão bibliográfica que embasam o trabalho. No terceiro capítulo, é detalhado o desenvolvimento do projeto, incluindo metodologias, técnicas e procedimentos adotados. Os resultados obtidos são discutidos no quarto capítulo, seguidos pela conclusão e considerações finais no último capítulo.

# 2 Fundamentação Teórica

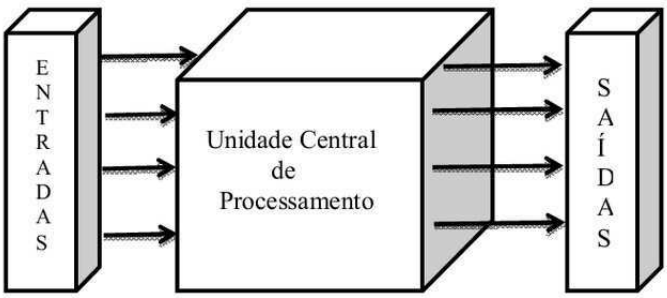
Este capítulo apresenta os conceitos teóricos principais para compreender o projeto proposto. Inicia-se com uma explanação sobre Controladores Lógicos Programáveis (CLPs), onde são abordados os tipos de linguagens de programação definidos pelo padrão IEC 61131-3, destacando suas características, aplicabilidade e exemplos práticos. Também abordaremos sobre o controlador Proporcional-Integral-Derivativo (PID), descrevendo seus princípios básicos e sua aplicação em sistemas de controle de realimentação.

## 2.1 Controlador Lógico Programável

Um CLP, como ilustrado na Figura 1, é formado basicamente de duas seções:

- uma unidade central de processamento (*Central Processing Unit - CPU*);
- interfaces de entrada e saída.

Figura 1 – Estrutura básica do CLP




Fonte: MESQUITA, Jefferson e ANDRADE (2012)

A CPU é responsável por executar as instruções de controle e supervisionar todas as atividades do sistema. Os três componentes que formam a CPU são:

- o processador;
- a memória;
- a fonte de alimentação do sistema.

A execução de um programa em um CLP segue um ciclo de varredura (Figura 2), no qual o controlador lê as entradas, executa as instruções do programa armazenado em

sua memória e atualiza as saídas. Esse ciclo é repetido continuamente para garantir o controle contínuo do processo.

Figura 2 – Ciclo de varredura do CLP


Fonte: MESQUITA, Jefferson e ANDRADE (2012)

Os CLPs podem ser classificados em dois tipos principais: compactos e modulares. O primeiro, como o próprio nome sugere, são unidades integradas que possuem todos os componentes essenciais, como CPU, entradas e saídas digitais e analógicas, alimentação e interface de programação, em um único gabinete. Eles são adequados para aplicações simples e de menor porte, onde há espaço limitado e requisitos de E/S não muito complexos.

Por outro lado, os CLPs modulares são compostos por módulos separados para diferentes funções, como CPU, módulos de E/S digitais e analógicas, fonte de alimentação e outros. Esses módulos são conectados entre si através de barramentos ou redes internas do CLP, permitindo uma maior flexibilidade na configuração do sistema. Os CLPs modulares são ideais para aplicações que exigem expansibilidade, alta densidade de E/S ou requisitos específicos de interface.

Quanto às linguagens de programação, a norma IEC 61131-3 fornece três linguagens textuais e três linguagens gráficas para escrever programas de aplicação. As linguagens textuais são:

- Lista de Instruções (*Instruction List* - IL);
- Texto Estruturado (*Structured Text* - ST);
- Sequenciamento Gráfico de Funções - versão textual (*Sequential Function Chart* - SFC).

As linguagens gráficas são:

- Diagrama *Ladder* (*Ladder Diagram* - LD);
- Diagrama de Bloco de Funções (*Function Block Diagram* - FBD);
- Sequenciamento Gráfico de Funções - versão gráfica.

As linguagens ST, IL, FBD e SFC podem ser usadas dentro de blocos de ação, os quais representam unidades de execução de instruções ou operações específicas dentro do programa do CLP, bem como em transições da programação SFC.

### 2.1.1 Texto Estruturado

O texto estruturado é uma linguagem textual de alto nível na qual muitas operações e instruções podem ser executadas em uma única linha de comando. Diferentemente das linguagens de baixo nível orientadas a máquina, o ST oferece uma ampla gama de declarações abstratas para descrever funcionalidades complexas de maneira extremamente compacta. Podemos considerar o ST como uma linguagem de alto nível devido à essa característica. O ST compartilha semelhanças com linguagens como Pascal ou C.

Essa linguagem foi especialmente projetada para programar funções aritméticas complexas, manipular tabelas e trabalhar com objetos de texto e palavras.

A norma IEC define uma faixa de operadores padrões para operações aritméticas e boolenas.

Tabela 1 – Operadores padrões

| **Operador** | **Descrição** | **Precedência** |
|---|---|---|
| (...) | Expressão com parêntesis | Maior |
| Função (...) | Lista de parâmetros de uma função | - |
| ** | Exponenciação | - |
| - | Negação | - |
| NOT | Complemento booleano | - |
| * | Multiplicação | - |
| / | Divisão | - |
| MOD | Operador de módulo | - |
| + | Soma | - |
| - | Subtração | - |
| <,>,<=,>= | Comparação | - |
| = | Igualdade | - |
| <> | Desigualdade | - |
| AND, &<br>, XOR | E booleano<br>Ou Exclusivo booleano | -<br>- |
| OR | Ou booleano | Menor |

Fonte: GUIMARÃES (2005)

Como mostrado na Tabela 1, quando os operadores têm a mesma precedência, eles são avaliados da esquerda para a direita. Expressões entre parêntesis têm a maior precedência, ou seja, devem ser avaliadas antes das demais, de dentro pra fora. As declarações de ST são resumidas na tabela 2.

Tabela 2 – Declarações do ST

| **Palavra-Chave** | **Descrição** | **Exemplo** | **Explicação** |
|---|---|---|---|
| := | Atribuição | d := 10; | Atribuição de um valor calculado à direita para o identificador à esquerda. |
| | Chamar de um FB | **FBName**(<br>Par1:=10,<br>Par2:=20); | Chamada de outra POU do tipo FB incluindo os parâmetros |
| RETURN | Return | **RETURN**; | Sai da atual POU e retorna para a POU chamada. |
| IF | Selecão | **IF** d < e **THEN** f := 1;<br>**ELSIF** d = e **THEN** f := 2;<br>**ELSE** f := 3;<br>**END IF**; | Seleção de alternativas por meio de expressões booleanas. |
| CASE | Multi-seleção | **CASE** f **OF**<br>1 : g := 11; 2 : g := 12;<br>**ELSE** g := FunNameO;<br>**END CASE**; | Seleção de um bloco de declarações, dependendo do valor da expressão "f". |
| FOR | Iteração (1) | **FOR** h := 1 **TO** 10 **BY** 2 **D0**<br>f[h/2] := h;<br>**END FOR**; | Loop múltiplo de um bloco de declarações com uma condição de início e fim e um valor de incremento. |
| WHILE | Iteração (2) | **WHILE** m > 1 **DO**<br>n := n / 2;<br>**END WHILE**; | Laço múltiplo de um bloco de instrução com condição de término no início. |
| REPEAT | Iteração (3) | **REPEAT**<br>i := i*j;<br>**UNTIL** i < 10000<br>**END REPEAT**; | Loop múltiplo de um bloco de declarações com condição de término no final. |
| EXIT | Fim de um loop | **EXIT**; | Término prematuro de uma declaração de iteração. |
| ; | Declaração fictícia | ; ; | |

Nota: Chamadas de função não são declarações por si só, mas podem ser usadas dentro de uma expressão como operandos.

Fonte: Tiegelkamp e John (2010)

Como mostrado na Tabela 2, cada declaração utiliza o valor de variáveis individuais como “d”, “e” e “f”, constantes ou o resultado de uma computação de várias variáveis (por exemplo, multiplicação: i * j). A parte de uma declaração que combina várias variáveis e/ou chamadas de função para produzir um valor é chamada de expressão. Compreender as expressões é essencial para entender como funcionam as declarações do ST (TIEGELKAMP; JOHN, 2010).

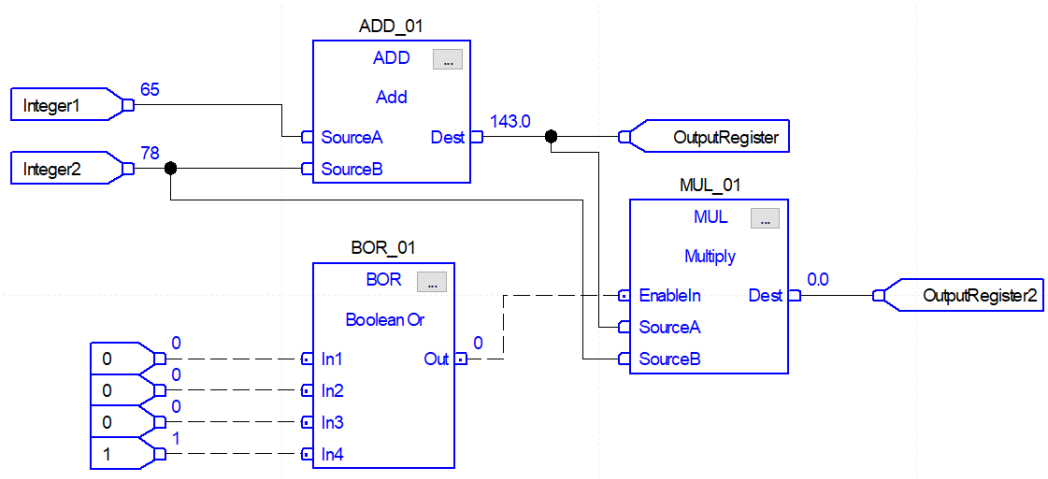
Portanto, sua sintaxe intuitiva e poderosa, aliada à sua capacidade de lidar com operações aritméticas e lógicas avançadas, tornam-no uma escolha ideal para a implementação de lógica de controle em controladores programáveis.

### 2.1.2 Diagrama de Blocos de Função

A linguagem FBD (*Function Block Diagram*) originou-se no campo do processamento de sinais, onde valores inteiros e/ou de ponto flutuante são importantes. Enquanto isso, ela se estabeleceu como uma linguagem universalmente utilizável no campo dos controladores industriais (TIEGELKAMP; JOHN, 2010).

De acordo com o estudo de Hanssen (2015), o conceito desta linguagem baseia-se em conexões entre funções e blocos de função. Ela permite a expressão e execução de funções complexas de forma simplificada e compreensível. Como isso inclui o uso de funções lógicas padrão, como *AND, OR, NOT*, em uma forma gráfica, muitas pessoas com conhecimento de eletrônica digital tem facilidade com o FBD.

Figura 3 – Exemplo do código em FBD




Fonte: Romanov (2020a)

Na Figura 3, temos os blocos ADD, BOR e MUL. A instrução *ADD* soma o valor de *Integer1*, ao de *Integer2* e armazena o resultado em *OutputRegister*. A instrução BOR representa uma operação lógica *OR* booleana, essa função é derivada de funções de portas lógicas e avalia como verdadeira sempre que qualquer uma das entradas for igual a 1 ou tiver nível alto. A instrução MUL é utilizada para realizar a multipliação entre *OutputRegister* e *Integer2* e armazenando o resultado em *OutputRegister2*, essa instrução só é ativada se o *EnableIn* estiver em nível lógico alto.

Em Diagramas de Blocos de Funções, o usuário é capaz de rotear os valores dos registradores para múltiplos destinos. Não é possível rotear dois registradores de entrada

para um único bloco de instrução de entrada. No entanto, é possível rotear uma saída única para duas entradas diferentes.

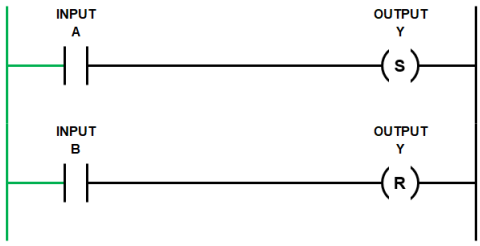
### 2.1.3 Diagrama Ladder

Antes dos CLPs, as plantas de fabricação empregavam circuitos baseados em relés para energizar diferentes cargas com base na forma como os relés estavam conectados. Os padrões de circuito usados para esses desenhos são conhecidos como diagrama *ladder*. Os relés eram caros, exigiam manutenção constante e não podiam ser facilmente reconfigurados. Com a introdução dos CLPs nesse processo, era essencial manter uma semelhança com o sistema antigo; assim, a lógica *ladder* foi criada como a primeira linguagem de programação de CLPs (ROMANOV, 2020b).

A lógica *ladder* recebe esse nome porque o *software* é organizado na forma de uma escada. No lado esquerdo (contatos), as instruções de lógica *ladder* são definidas como condições , enquanto as do lado direito (bobinas) são instruções que são acionadas se as condições forem atendidas. Cada degrau da escada se estende da esquerda para a direita e é executado de cima para baixo pelo CLP.

Conforme o CLP começa a processar as instruções, ele as lê no lado esquerdo e determina se a lógica desse lado do degrau está definida como verdadeira. A Lógica é avaliada como verdadeira quando uma corrente hipotética é capaz de passar pelas instruções. Cada instrução possui um conjunto de condições que a tornam verdadeira ou falsa.

Figura 4 – Exemplo Gráfico de um *Set* e *Reset* em Ladder




Fonte: Ladder... (2023)

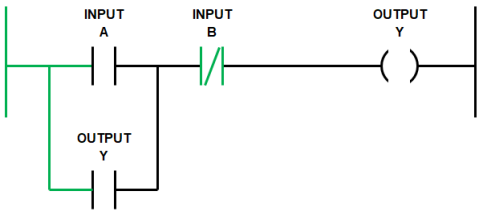
Na Figura 4, temos uma representação de um código em *ladder* onde apresenta duas linhas verticais, chamadas de trilhos, sendo que o trilho esquerdo fornece energia ao circuito, passando então por cada trilho. Cada trilho possui chaves e bobinas de saída. As chaves podem realizar operações *OR, AND e NOT*, e por meio dessas operações lógicas básicas podemos criar qualquer lógica de programação no CLP. Quando todas as condições de um trilho são atendidas, a bobina de saída conectada é energizada. Isso ativa o sistema do mundo real, como ligar ou desligar um motor. Esse processo é continuamente escaneado

e repetido pelo CLP e pelo sistema de controle para garantir a automação do sistema operacional.

Ainda na Figura 4, temos as entradas A e B e a saída Y. Quando tanto a entrada A quanto a entrada B são falsas, então o estado da saída Y não muda. Se a saída Y for falsa, então ela permanece falsa. Se a entrada A muda para verdadeira momentaneamente, então o símbolo de *Set* muda o estado da saída Y para verdadeira. Então, após varreduras subsequentes, se a entrada A mudar para o estado falso, isso não afeta o estado da saída Y. Em outras palavras, a saída Y é travada em verdadeira.

Somente quando a entrada B muda para o estado verdadeiro, o símbolo de *Reset* muda o estado da saída Y de volta para falso. Em outras palavras, a saída Y está agora destravada. Portanto, se tanto a entrada A quanto a entrada B estiverem verdadeiras ao mesmo tempo, então, no diagrama *ladder* da Figura 4, a primeira linha é avaliada e a saída Y é definida como verdadeira. Mas, em seguida, a segunda linha é avaliada e a saída Y é definida como falsa. Assim que a varredura alcança o final de todo o programa, ela executará o estado da saída Y como *False*.

Figura 5 – Exemplo Gráfico de um contato de selo em Ladder




Fonte: Ladder... (2023)

Na Figura 5, temos outra representação de um código em *ladder*, dessa vez mostrando um contao de selo. Se entrada A se tornar verdadeira e a entrada B for falsa, então a saída Y se torna verdadeira. Quando a varredura retorna novamente do topo, se a entrada A se tornar verdadeira, com a entrada B ainda falsa, então a saída Y permanece verdadeira.

Uma vez que tenhamos acionado o "selo" usando a entrada A, a saída Y permanecerá travada mesmo se a entrada A se tornar falsa. A saída Y permanecerá travada em verdadeira até que a entrada B se torne verdadeira. Como a entrada B é um símbolo de contato normalmente fechado (NF), quando ela se torna verdadeira, o fluxo lógico é bloqueado e a saída Y se torna falsa. Assim, liberando a trava.

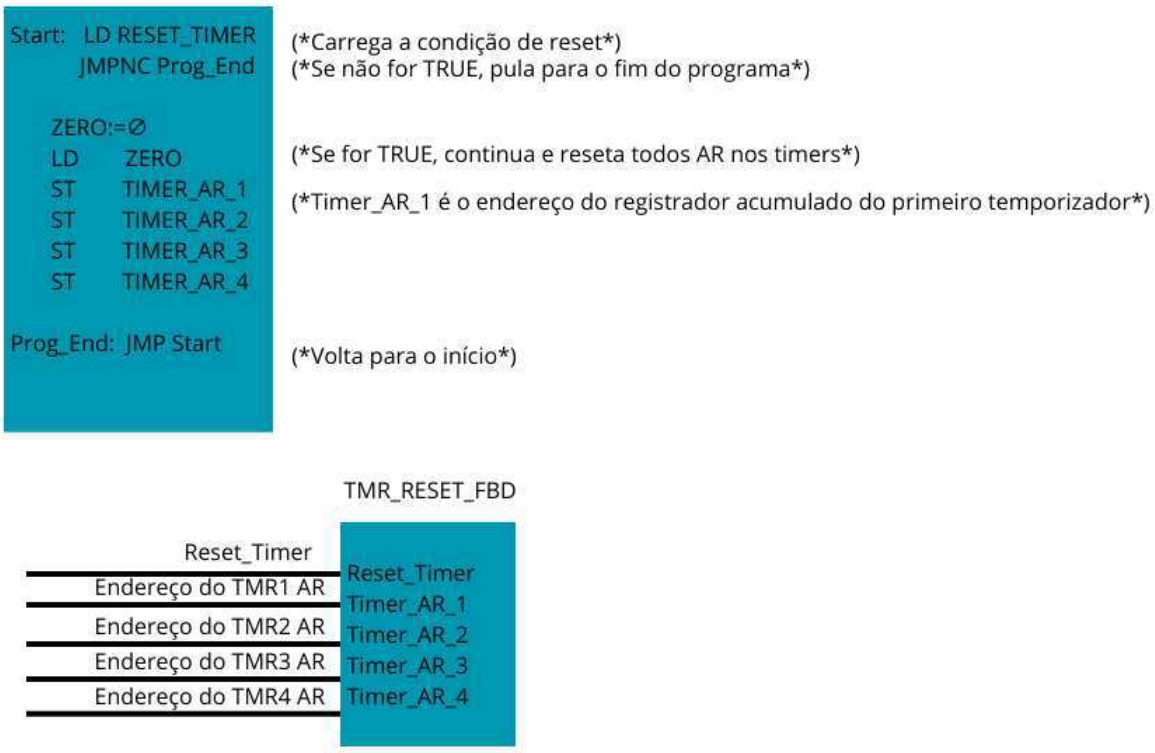
### 2.1.4 Lista de Instruções

Lista de instruções é uma linguagem de baixo nível semelhante à linguagem de máquina ou *assembly* usada com microprocessadores. Apesar de haver desvantagens

associadas ao uso de uma linguagem de baixo nível como IL, a vantagem da linguagem é que ela não requer muita potência de computação. A razão pela qual a linguagem continua a ser usada é que ela existe há mais tempo do que as outras linguagens no padrão, juntamente com o LD (HANSSEN, 2015).

Nos CLPs mais antigos, o código em IL ou LD pode ser programado e transferido para a CPU por meio de um painel especial. Não era necessário (ou possível), como é hoje, conectar um PC para programação e diagnóstico. Esse tipo de linguagem é útil para pequenas aplicações, bem como para aplicações que requerem otimização de velocidade do programa ou de uma rotina específica no programa. IL pode ser usada para criar blocos de função personalizados. Uma aplicação típica de IL pode envolver a inicialização para zero (ou seja, reset) dos registradores de valor acumulado para todos os temporizadores em um programa de controle. Como mostrado na Figura 6, um programador poderia usar IL para criar um bloco de função que carregaria o conteúdo dos registradores acumulados (AR) de todos os temporizadores com um valor zero.

Figura 6 – Bloco de função personalizado em lista de instruções que reinicia os valores dos temporizadores para zero




Fonte: Adaptado de Bryan e Bryan (1997)

O endereço do registrador acumulado do primeiro temporizador é TMR1_AR. Em FBD, este endereço é conhecido como Timer_AR_1 para que o programa IL possa interpretá-lo. O resultado do programa IL será que os valores nos registradores acumulados especificados serão redefinidos para 0. A variável Reset_Timer acionará o bloco e iniciará a

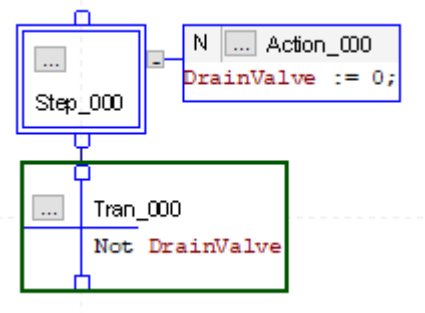
instrução IL. A rotina IL voltará ao início enquanto o bloco estiver habilitado pela variável Reset_Timer estar ATIVA. Também existem maneiras de "pulso" apenas uma vez através do programa (BRYAN; BRYAN, 1997).

### 2.1.5 Sequenciamento Gráfico de Funções

Sequenciamento Gráfico de Funções é bastante importante na programação de sequências de controle. O SFC é uma linguagem gráfica que fornece uma representação diagramática de sequências de controle em um programa. Ele organiza os subprogramas ou sub-rotinas (programadas em LD, FBD, IL e/ou ST) que formam o programa de controle. Isso é particularmente útil para operações de controle sequencial, onde um programa flui de uma etapa para outra conforme uma condição é satisfeita.

Os elementos principais do *framework* de programação SFC são as etapas, transições e ações. Uma etapa é uma fase no processo de controle, enquanto uma transição indica a mudança de uma etapa para outra. Cada etapa pode ou não ter uma ação associada a ela. A lógica sequencial dos SFC's é fácil de seguir, semelhante à leitura de um fluxograma, o que facilita a compreensão e o desenvolvimento de programas de controle complexos (VENTER, 2021).

Figura 7 – SFC utilizando Texto Estruturado no elemento de transição no RSLogix 5000




Fonte: Venter (2021)

Como mostrado na Figura 7 a *tag* de ação, do tipo de estrutura de dados "SFC_ACTION", é automaticamente criada. Na caixa de ação, as instruções a serem executadas pela etapa associada podem ser programadas. Essa lógica é inserida usando Texto Estruturado.

Quando as ações a serem realizadas na etapa específica forem concluídas, o fluxo normal do Gráfico de Função Sequencial será direcionado para o elemento de transição.

É aqui que condições específicas devem ser atendidas para indicar que a etapa, juntamente com suas ações associadas, foi executada com sucesso e, portanto, o fluxo normal no diagrama pode continuar para o próximo elemento. A *tag* de transição é do

tipo de dados "*BOOL*", pois seu propósito é exclusivamente indicar um valor lógico 0 / falso (o fluxo não pode continuar) ou 1 / verdadeiro (o fluxo pode continuar), e também é programada fazendo uso do Texto Estruturado.

## 2.2 Controlador PID

O controlador PID é, de longe, o algoritmo de controle mais comum. A maioria dos circuitos de realimentação é controlada por esse algoritmo ou por variações menores dele. Ele é implementado em muitas formas diferentes, como um controlador independente ou como parte de um pacote de Controle Digital Direto (DDC) ou de um sistema de controle de processos distribuído hierarquicamente. Muitos milhares de engenheiros de instrumentação e controle em todo o mundo estão usando tais controladores em seu trabalho diário. O algoritmo PID pode ser abordado de muitas maneiras diferentes. É visto como um dispositivo que pode ser operado com algumas regras básicas, mas também abordado de forma analítica (ÅSTRÖM; HÄGGLUND, 2006).

Esta estratégia combina três ações de controle básicas: ação proporcional (P), ação integral (I) e ação derivativa (D). O PID é amplamente aplicado em uma variedade de sistemas de controle, desde sistemas industriais até sistemas de controle de temperatura em eletrodomésticos (SEBORG et al., 2016).

Quando o sinal de controle realimentado é linearmente proporcional ao erro do sistema, chamamos o resultado de realimentação proporcional. O controle integral é aquele cuja estratégia de controle se baseia na obtenção do sinal de saída do controlador como uma integral do erro. O erro é obtido pelo cálculo entre o valor setado (*Setpoint* - SP) e o valor medido (*Process Variable*- PV). O controlador derivativo é um tipo de controlador que atua na variável controlada a partir da variação do erro. Isto é, ele considera a diferença entre o erro atual e o erro anterior, isso ajuda a prever as tendências futuras do erro.

A equação do controlador PID pode ser descrita como:

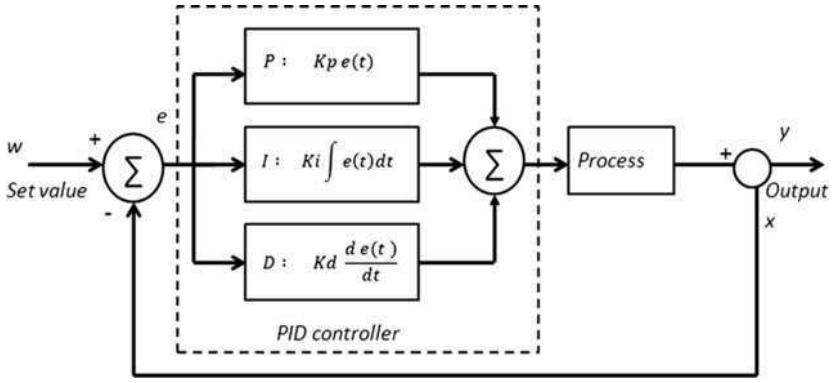
$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\frac{de(t)}{dt}\right) \tag{2.1}$$

Onde:

- $u(t)$: Saída do controlador (sinal de controle) no instante $t$.
- $K$: Ganho proporcional do controlador, que amplifica o erro para produzir a saída do controlador.
- $e(t)$: Erro no instante $t$, definido como a diferença entre o valor desejado (*setpoint*) e o valor real da variável controlada.

- $T_i$: Tempo integral, que determina a contribuição do termo integral no controle. É o inverso da taxa integral.
- $\int_0^t e(\tau)d\tau$: Integral do erro ao longo do tempo até o instante $t$, calculada desde o início do controle.
- $T_d$: Tempo derivativo, que determina a contribuição do termo derivativo no controle.
- $\frac{de(t)}{dt}$: Derivada do erro em relação ao tempo, que representa a taxa de variação do erro no instante $t$.

Figura 8 – Diagrama de blocos do sistema de controle PID em malha fechada




Fonte: Halim e Ismail (2019)

Na Figura 8, a representação do controlador PID em diagrama de blocos é apresentada. A entrada $e$ é fornecida às seções proporcional, integral e derivativa no mesmo instante de tempo e suas saídas correspondentes são adicionadas usando um somador. A entrada somada é fornecida ao processo que define os parâmetros correspondentes que são então aplicados novamente ao sistema. Então, a entrada da variável do processo é comparada com a entrada real para gerar um erro, que por sua vez é usado para ajustar a saída do sistema. Este procedimento continua até que o sinal de erro atinja zero ou quando o valor da variável do processo se iguala ao ponto de ajuste.

# 3 Desenvolvimento

Foi desenvolvida uma nova programação da lógica de controle de uma planta industrial em escala laboratorial. Inicialmente, a planta estava programada utilizando a linguagem *ladder*. No entanto, como parte do desenvolvimento, essa programação foi convertida para FBD e ST.

Houve uma necessidade de sintonização do controlador para a malha de controle do tanque 2 devido à importância de se alcançar um desempenho ideal do sistema de controle. Observou-se que os parâmetros iniciais do controlador não estavam adequadamente otimizados. A sintonização visa ajustar os ganhos do controlador PI para melhorar a estabilidade, reduzir o tempo de resposta e minimizar o erro, garantindo que o tanque 2 opere de forma eficiente e conforme os requisitos desejados. Para isso, utilizou-se o software BR-Tuning (anexo A).

Neste capítulo, serão apresentados detalhes da planta utilizada neste projeto, além de discutir as diferenças entre as linguagens empregadas em sua programação e como foram realizadas a identificação do modelo e a sintonia do controlador.

## 3.1 Descrição da Planta

A planta, composta por quatro tanques numerados da esquerda para a direita, é interligada por meio de válvulas manuais, possibilitando uma variedade de configurações, conforme ilustrado na Figura 9. Esses tanques são abastecidos por um reservatório central de água, fazendo uso de duas bombas hidráulicas acionadas por inversores de frequência. Cada tanque pode receber água de qualquer uma das bombas, de acordo com a configuração definida pelas válvulas.

Os tubos de conexão estão numerados com o número 1 e 2, de acordo com a bomba que os abastecem. Nas saídas de cada bomba, há um sensor de pressão diferencial instalado, usado para medir a vazão.

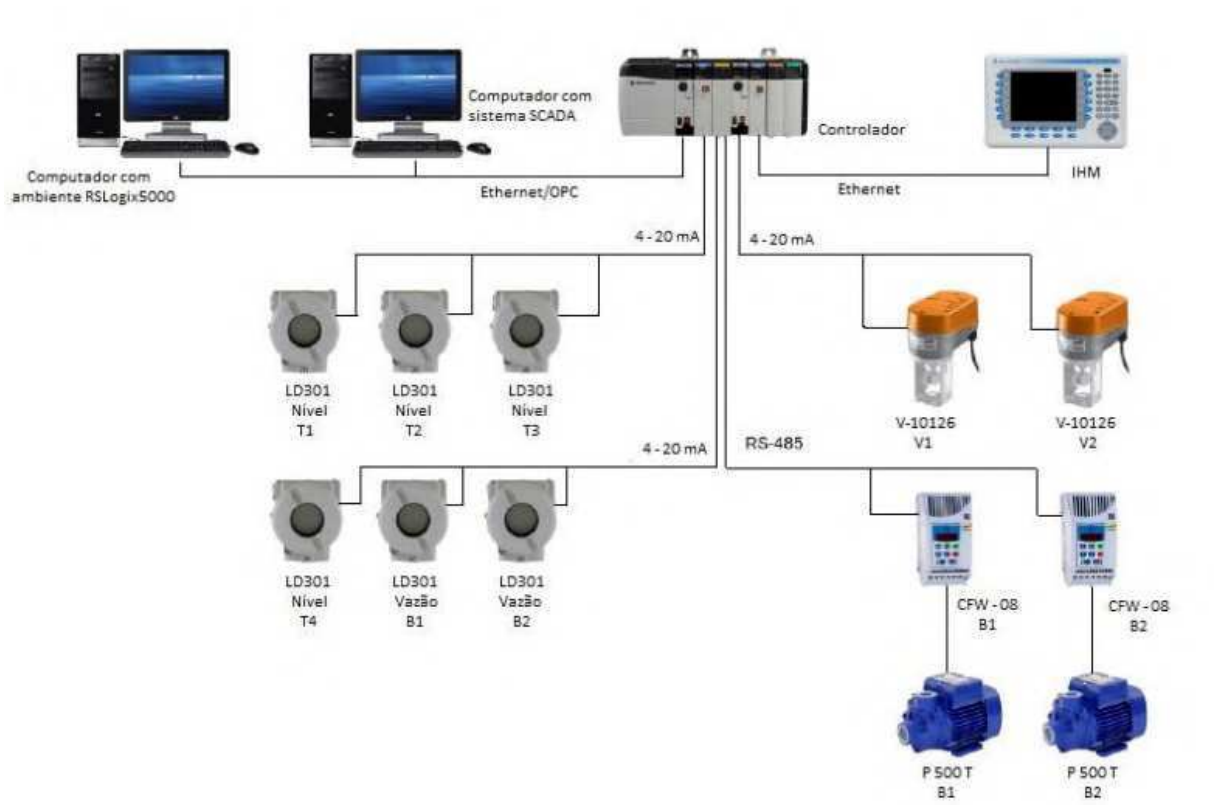
O sistema está equipado com quatro sensores de pressão diferencial, responsáveis por monitorar os níveis de água em cada um dos tanques. Além disso, as saídas dos tanques intermediários são ligadas ao reservatório por meio de válvulas elétricas. Para garantir a segurança operacional, a planta possui boias de segurança em cada tanque e no reservatório, interrompendo o fluxo de água quando atingem a capacidade máxima.

Figura 9 – Planta didática com tanques de nível

Fonte: Junior (2022)

A conexão entre os componentes da planta é representada pelo esquema da Figura 10, que mostra a integração de sensores de pressão diferencial, válvulas elétricas, motores elétricos e outros dispositivos.

Figura 10 – Esquema de conexão dos componentes




Fonte: Maia (2016)

O controle da planta é realizado por um CLP fabricado pela Allen Bradley (Figura 12), utilizando o ambiente de programação RSLogix5000. A comunicação entre os dispositivos é estabelecida por meio de protocolos como RS-845 e Ethernet/OPC, garantindo a coordenação e supervisão eficientes do sistema.

Figura 11 – CLP utilizado na planta


Fonte: Junior (2022)

O líquido utilizado para abastecimento dos tanques foi a água, conforme ilustrado na Figura 12.

Figura 12 – Modelo de tanque presente na planta didática


Fonte: Junior (2022)

As válvulas manuais restringem a passagem do fluido em determinados locais. Além disso, elas tem a função de ajustar e limitar a máxima passagem de fluido em determinada tubulação. As válvulas elétricas utilizadas são da fabricante Belimo, modelo NVF24-MFT-E-50, ilustrada na Figura 13. Com uma faixa de tensão de alimentação de 110V-220V, elas operam eficientemente em temperaturas que variam de -10ºC a 50ºC. Essas válvulas têm uma expectativa de vida de 50.000 ciclos, sendo que cada ciclo corresponde a um período de 12 segundos.

Figura 13 – Válvulas elétricas utilizadas na planta


Fonte: Junior (2022)

Os motores das bombas presentes na planta são ilustrados nas Figuras 14 e correspondem ao modelo HYDROBLOC P500T, fabricados pela KSB Bombas Hidráulicas S/A. Esses motores são trifásicos, apresentando uma potência nominal de 0,5 HP e uma capacidade máxima de elevação de 35 metros.

Figura 14 – Motor-bomba usado na planta

Fonte: Junior (2022)

Cada tanque está equipado com um sensor de pressão projetado para medir a altura da coluna de água dentro dele, que é a variável de interesse no sistema. Estes sensores utilizam o padrão de comunicação 4-20 mA e foram fabricados pela SMAR, modelo LD301, conforme ilustrado na Figura 15. São dispositivos de 3,5kg que operam a uma temperatura máxima de 85°C e possuem exatidão de 0,075%.

Além disso, na parte superior de cada tanque, há um sensor de segurança destinado a prevenir o transbordamento. Quando a altura da água atinge um nível crítico, os motores são desligados automaticamente para evitar danos.

Figura 15 – Transmissor utilizado na planta


Fonte: Junior (2022)

Os inversores de frequência empregados na planta são do modelo SFW08, assim ilustrado na Figura 16, projetados para proporcionar um acionamento de velocidade variável, permitindo o controle preciso da velocidade da bomba hidráulica a eles conectada. Possuem uma faixa de tensão de alimentação de 200 a 480V e uma corrente de saída variando de 1A a 33A.

Figura 16 – Inversores utilizados na planta


Fonte: Junior (2022)

O painel elétrico de controle pode ser visto na Figura 17. Ele é composto por uma estrutura metálica com ligações elétricas classificadas como entradas e saídas. A entrada é encarregada de receber os sinais de comando, enquanto a conexão de saída transmite

pulsos elétricos para a movimentação dos dispositivos. Na porta, existem botões com os quais é possível ligar todo o painel e visualizar o estado de alguns componentes.

Figura 17 – Painel completo da planta

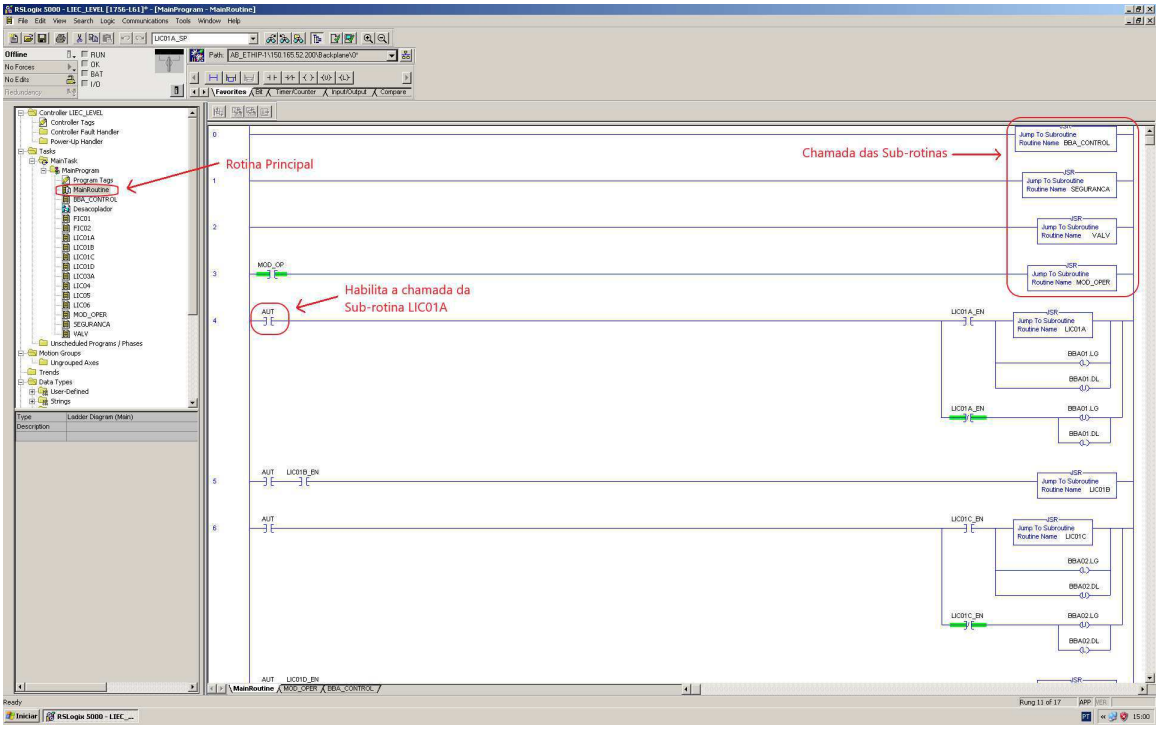

Fonte: Junior (2022)

## 3.2 Programação

Nesta seção, explora-se as implementações em Texto Estruturado e Diagrama de Blocos de Função para a planta, em contraste com a programação que já estava elaborada em *Ladder*. Enquanto a programação em LD já estava estabelecida, as novas abordagens em ST e FBD foram desenvolvidas para oferecer uma perspectiva diferente e possivelmente mais eficiente para a operação da planta. Vale ressaltar que, na implementação em ST, foi escrito apenas as sub-rotinas BBA_CONTROL e LIC06. Essa escolha se deu devido a restrições de tempo e por essas sub-rotinas serem as principais para o entendimento do problema.

A programação foi organizada com uma função principal (*main*) e sub-rotinas. A *main* é onde são chamadas as diversas sub-rotinas que compõem o controle da planta. Ela serve como o ponto de partida da execução do programa, onde são iniciadas as verificações de segurança e o gerenciamento das operações do sistema.

Figura 18 – Rotina Principal




Fonte: Elaborado pelo autor

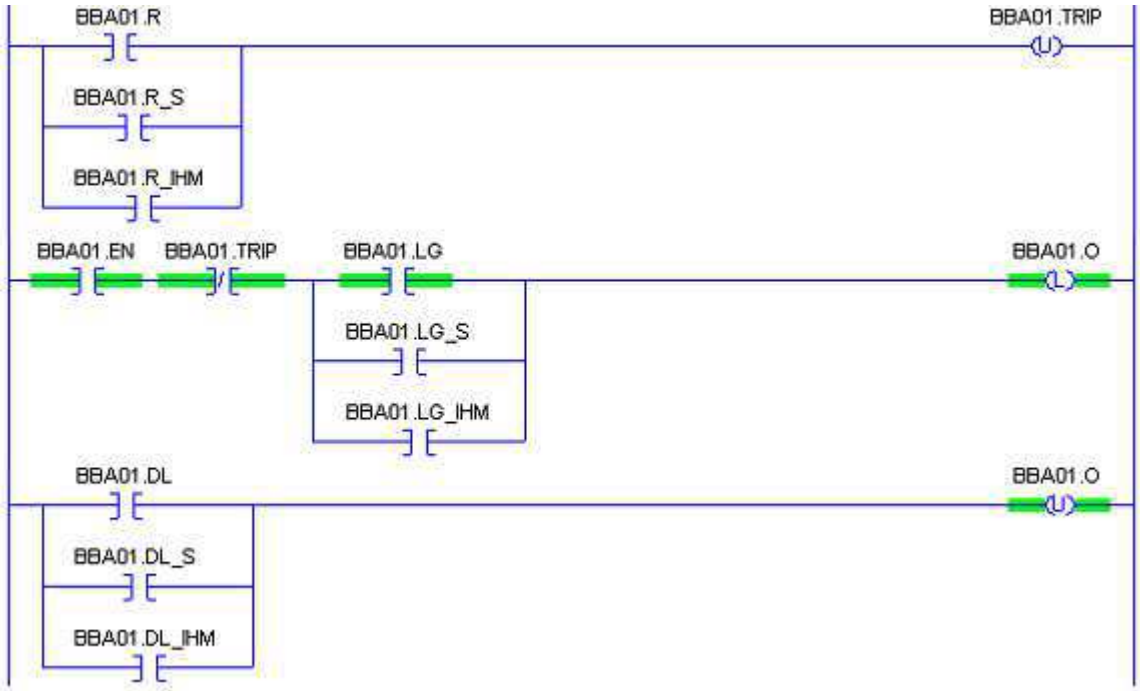
A primeira sub-rotina chamada pela *main* é a BBA_CONTROL, que é utilizada para controlar o acionamento das duas bombas da planta. Os contatos e saídas, listados na Tabela 3, seguem o formato BBA0x.y, onde "x" representa o número da bomba e "y" é o nome da *tag*. Todas essas variáveis pertencem a um tipo de *data type* e não correspondem diretamente a nenhuma entrada ou saída física do CLP, mas estão indiretamente associadas a elas. Um exemplo disso é a relação entre as variáveis BBA01.LG e BB01_LG: a primeira faz parte do *data type* e é utilizada no processo de acionamento da bomba 1, enquanto a segunda está diretamente ligada à saída física do CLP. Esse mesmo padrão de identificação, utilizando "." e "_", é aplicado a outras *tags*.

Tabela 3 – Descrição das *tags*

| Nome (y) | Tipo de dado | Descrição |
|---|---|---|
| YS | REAL | Saída Física Analógica |
| O | BOOL | Saída Física Digital |
| LG | BOOL | Liga Comando via CLP |
| LG_S | BOOL | Liga Comando via Supervisório |
| LG_IHM | BOOL | Liga Comando via IHM |
| DL | BOOL | Desliga Comando Via CLP |
| DL_S | BOOL | Desliga Comando via Supervisório |
| DL_IHM | BOOL | Desliga Comando via IHM |
| R | BOOL | Reset Comando via CLP |
| R_S | BOOL | Reset Comando via Supervisório |
| R_IHM | BOOL | Reset Comando via IHM |
| VAL | REAL | Valor de saída Analógica via CLP |
| VAL_S | REAL | Valor de saída Analógica via Supervisório |
| VAL_IHM | REAL | Valor de saída Analógica via IHM |
| ST | INT | Status do Motor |
| EN | BOOL | Habilita Partida Bomba |
| TRIP | BOOL | Desarme Bomba |
| MAN | BOOL | Bomba em operação Manual |

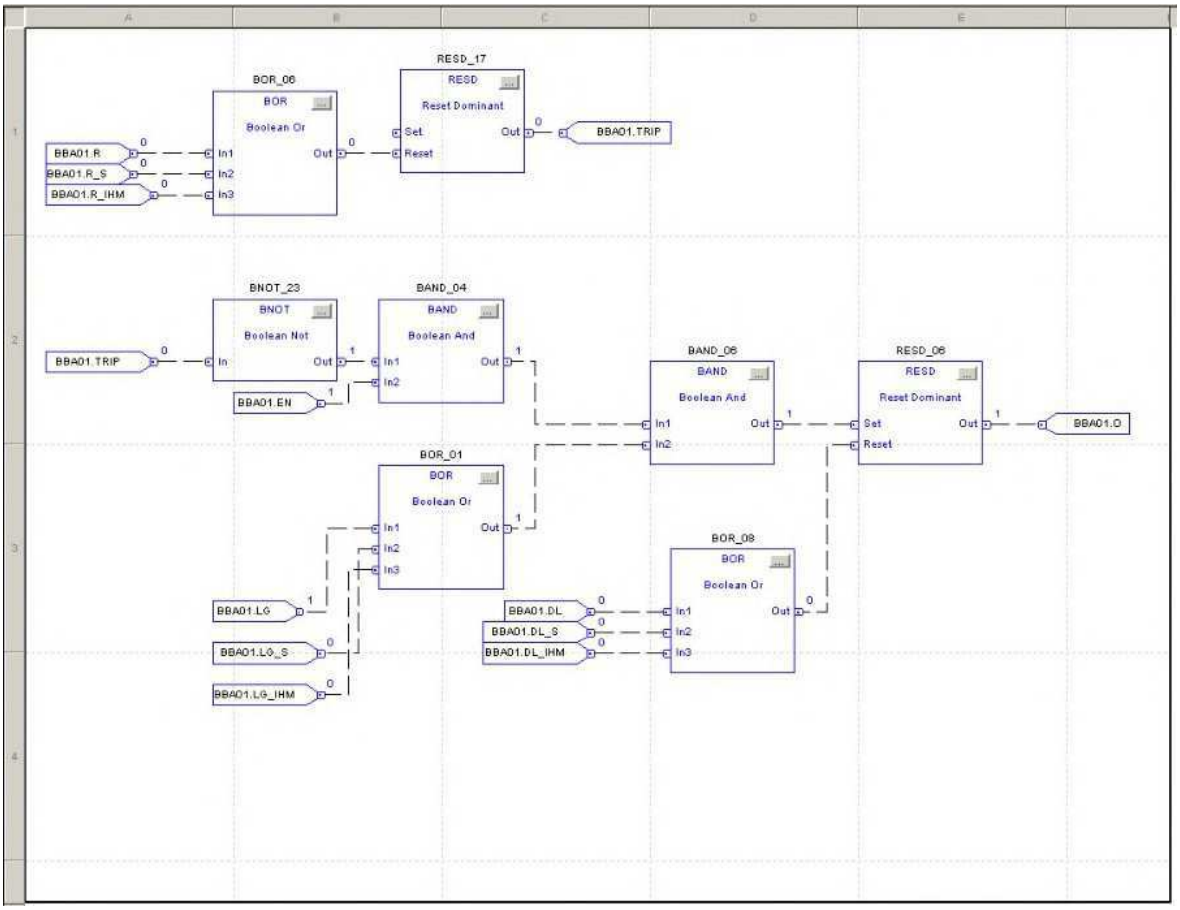
Fonte: Elaborado pelo autor

Figura 19 – BBA_CONTROL em *ladder*




Fonte: Elaborado pelo autor

Figura 20 – BBA_CONTROL em FBD




Fonte: Elaborado pelo autor

Figura 21 – BBA_CONTROL em ST

```
//BBA_CONTROL


//Reset no TRIP
IF BBA01.R OR BBA01.R_S OR BBA01.R_IHM THEN
    BBA01.TRIP := 0;
END_IF;

IF BBA02.R OR BBA02.R_S OR BBA02.R_IHM THEN
    BBA02.TRIP := 0;
END_IF;


//Liga ou Desliga a Bomba 1
IF BBA01.DL OR BBA01.DL_S OR BBA01.DL_IHM THEN
    BBA01.0 := 0;
ELSE
    BBA01.0 := (NOT BBA01.TRIP AND BBA01.EN) AND (BBA01.LG OR BBA01.LG_S OR BBA01.LG_IHM);
END_IF;


//Liga ou Desliga a Bomba 2
IF BBA02.DL OR BBA02.DL_S OR BBA02.DL_IHM THEN
    BBA02.0 := 0;
ELSE
    BBA02.0 := (NOT BBA02.TRIP AND BBA02.EN) AND (BBA02.LG OR BBA02.LG_S OR BBA02.LG_IHM);
END_IF;


//Acionamento das Saidas Digitais
IF BBA01.0 AND BBA01.EN AND (NOT BBA01.TRIP) THEN
    BBA01_LG := 1;
    BBA01_DL := 1;

ELSE
    BBA01_LG := 0;
    BBA01_DL := 0;
END_IF;

IF BBA02.0 AND BBA02.EN AND (NOT BBA02.TRIP) THEN
    BBA02_LG := 1;
    BBA02_DL := 1;

ELSE
    BBA02_LG := 0;
    BBA02_DL := 0;
END_IF;


//Referência do ST (CLP, Supervisório ou IHM) e da Velocidade da Bomba
IF BBA01.0 THEN
    CASE BBA01.ST OF
        0: BBA01.YS := BBA01.VAL;
        1: BBA01.YS := BBA01.VAL_S;
        2: BBA01.YS := BBA01.VAL_IHM;
    END_CASE;

    BBA01_YS := BBA01.YS;
END_IF;
```

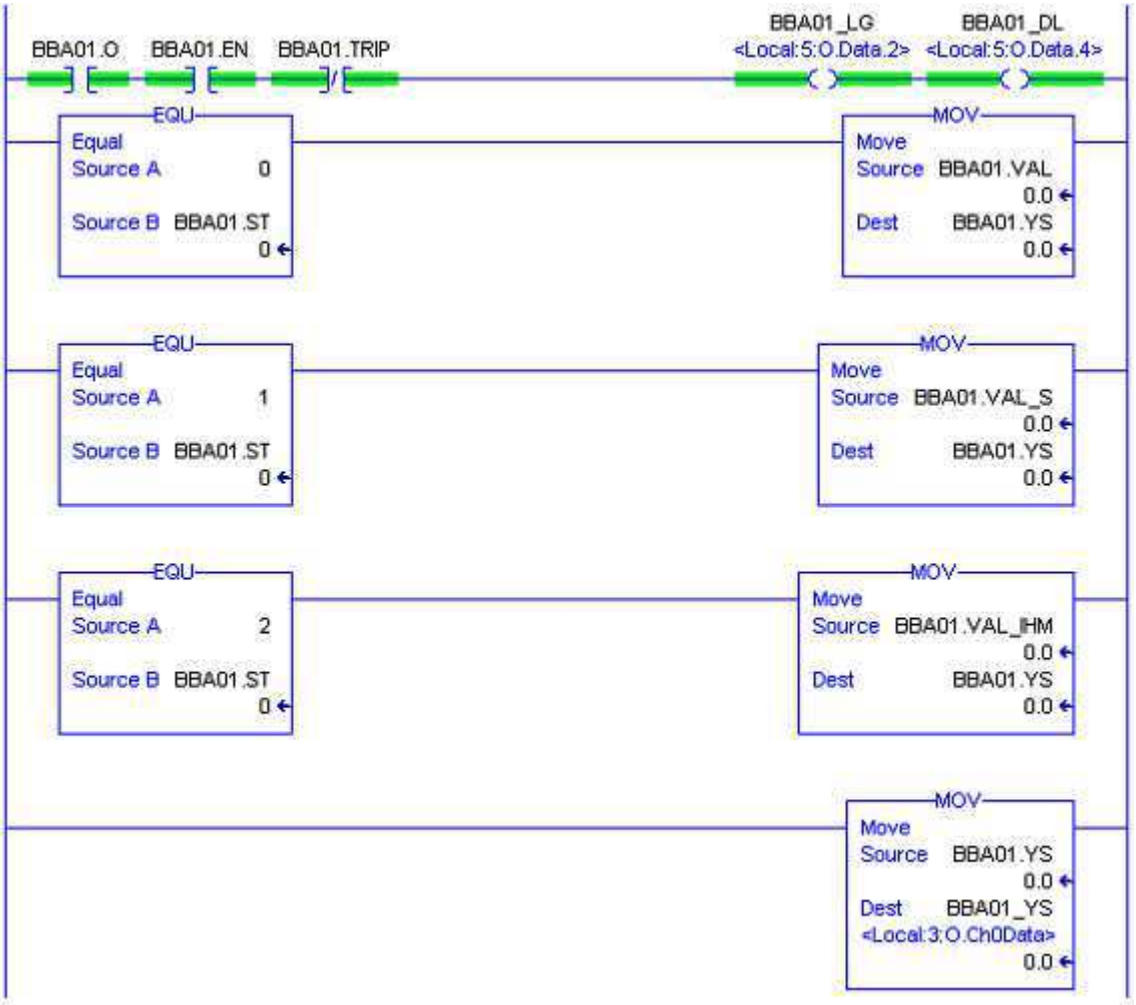
Fonte: Elaborado pelo autor

Nas Figuras 19, 20 e 21, nota-se uma entrada de *RESET* que é usada para permitir o acionamento da bomba. Por padrão, são usadas três entradas que podem acionar o *RESET* e desempenham a mesma função (acionamento manual via software CLP, via supervisório e via IHM). A ativação da saída BBA0x.TRIP, uma variável interna, contribui para desarmar a bomba e deixá-la em estado de espera pelo comando do inversor de

frequência, para isso é usada uma saída do tipo *UNLATCH* (*reset*) em *ladder*, e esse processo só é concluído mais adiante, no momento em que a saída física BBA0x_LG é ativada. Em FBD, usa-se o bloco "RESD", que, ao receber um sinal de entrada de *reset*, automaticamente ajusta a saída para 0. Por outro lado, em ST, a mesma funcionalidade exige a utilização de uma estrutura condicional *IF* para estabelecer a saída como 0.

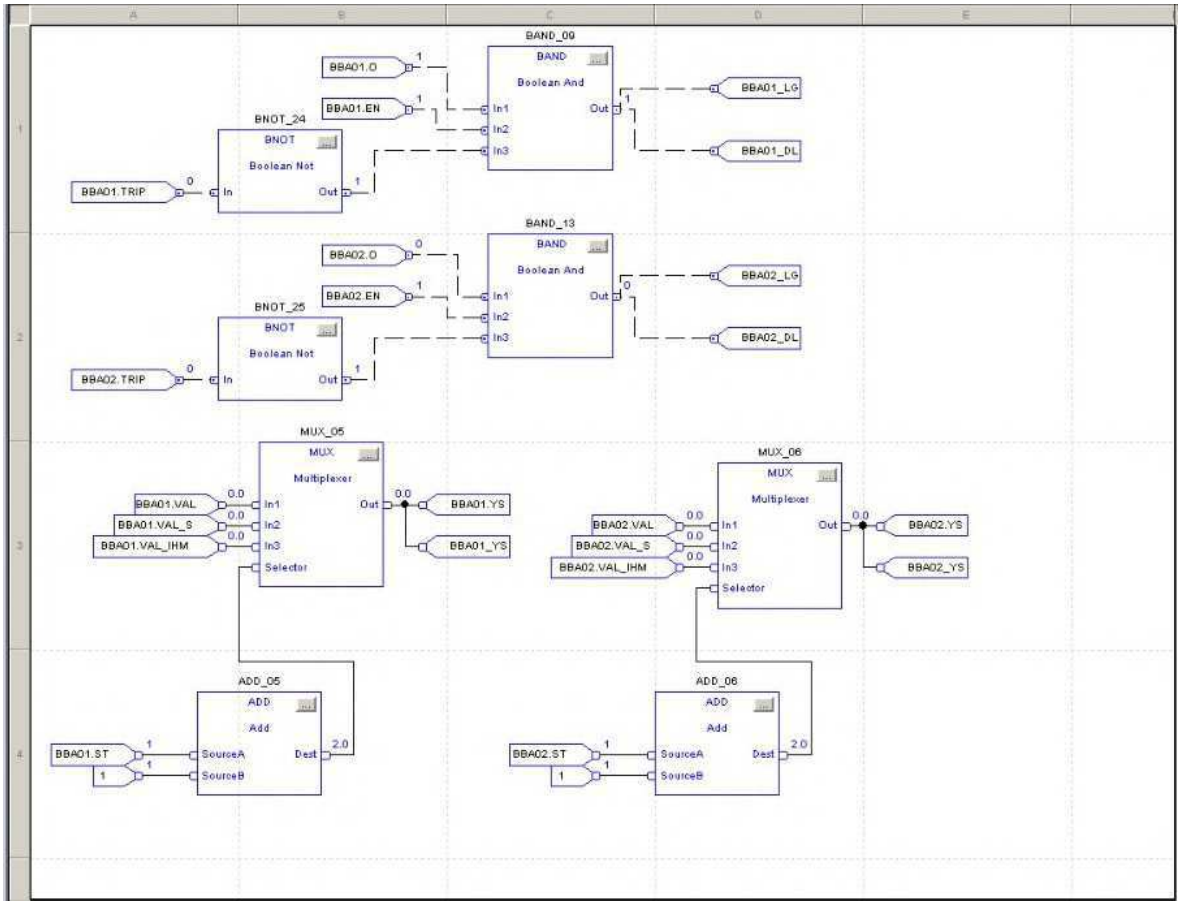
Mais adiante, em LD, a tag BBA0x.TRIP é conectada em série com BBA0x.EN e BBA0x.LG para ativar a saída BBA0x.O, que é uma saída física digital do tipo LATCH, significando que ela permanece ligada uma vez acionada até que seja explicitamente desligada. Logo após, a mesma saída BBA0x.O é novamente utilizada, desta vez com o tipo *UNLATCH*, indicando que ela será desligada quando acionada. Esse desligamento é acionado pelo contato BBA0x.DL, responsável por desligar a bomba. Em FBD, essa lógica é representada por meio dos blocos *BAND* (Boolean AND), *BOR* (Boolean OR) e *RESD* para ativar ou resetar a saída BBA0x.O. Já em ST, a mudança da saída é realizada novamente por meio de estruturas condicionais *IF* e *ELSE*.

Figura 22 – BBA_CONTROL em *ladder*




Fonte: Elaborado pelo autor

Figura 23 – BBA_CONTROL em FBD




Fonte: Elaborado pelo autor

Em seguida, nas Figuras 22 e 23, observa-se três contatos em série (BBA0x.O NA, BBA0x.EN NA e BBA0x.TRIP NF) que ativam duas saídas em série (BBA0x_LG e BBA0x_DL). Essas saídas digitais são responsáveis por ligar e desligar o inversor, respectivamente. Em termos físicos, a saída BBA0x_LG está associada a um contato normalmente aberto, enquanto a BBA0x_DL está associada a um contato normalmente fechado.

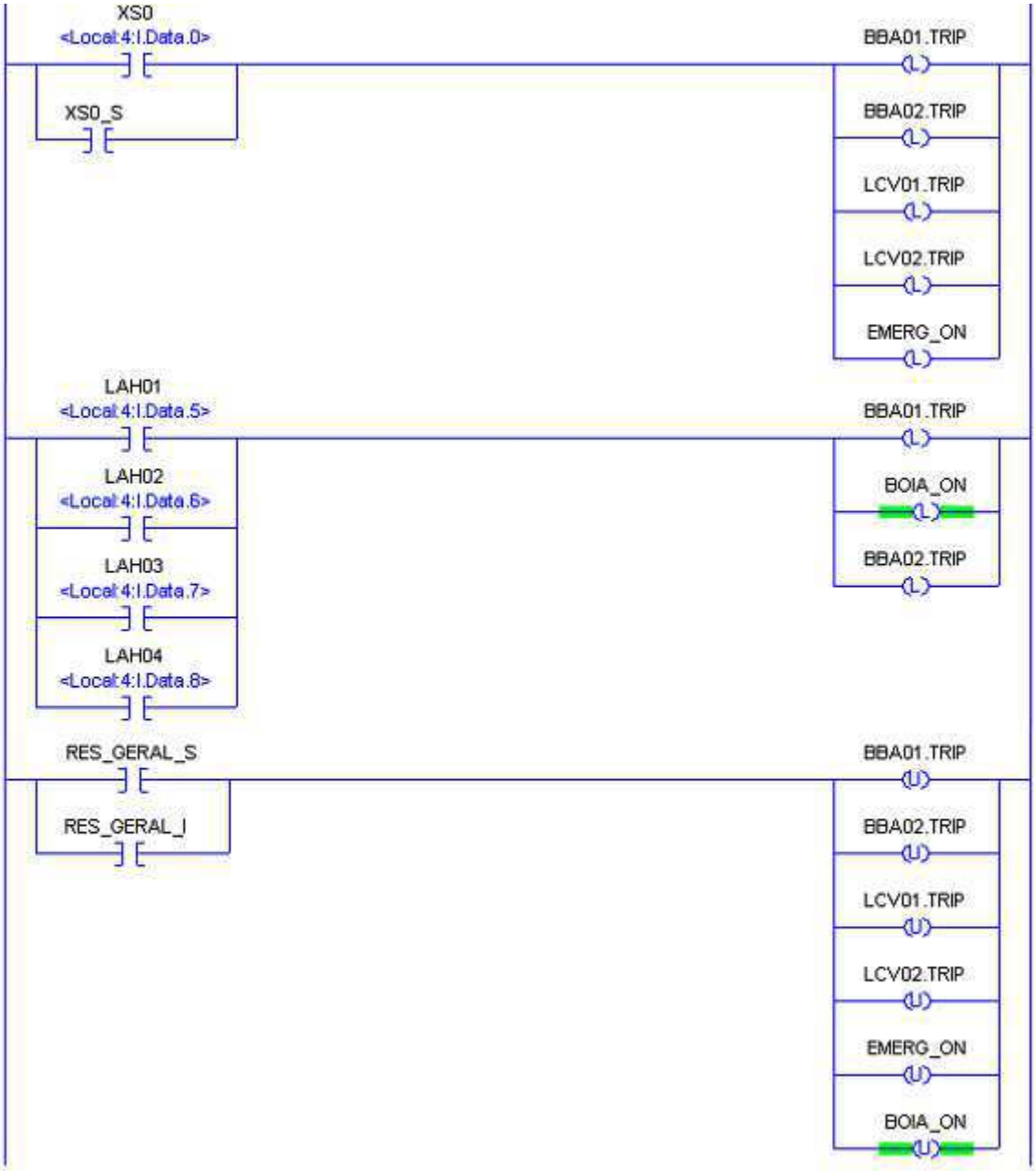
Ainda nessas mesmas Figuras, observamos em LD uma lógica de comparação: a *tag* BBA0x.ST (relé de status do inversor) é comparada com os valores 0, 1 e 2. Cada um desses valores define a fonte da qual vem a referência de velocidade da bomba, que é então transferida para a saída analógica BBA0x_YS (saída analógica do inversor de frequência, responsável por indicar a velocidade da bomba). Se o valor contido em BBA0x.ST for igual a 0, a referência é fornecida via CLP; se for 1, via supervisório; e se for 2, via IHM. Em FBD, é necessário utilizar um bloco *MUX*, associando suas três entradas à VAL, VAL_S ou VAL_IHM. O seletor do *MUX* é o status do motor, e as saídas são .YS e _YS. Já em ST, apenas uma condição *IF* com os *CASE* é necessária, juntamente com a associação correspondente da saída.

Tabela 4 – Descrição das Tags da sub-rotina SEGURANCA

| **Nome (y)** | **Tipo de dado** | **Descrição** |
|---|---|---|
| XS0 | BOOL | Botoeira de Emergência Geral |
| XSO_S | BOOL | Botoeira de Emergência Geral via Supervisório |
| LAH01 | BOOL | Boia de nível máximo tanque 01 |
| LAH02 | BOOL | Boia de nível máximo tanque 02 |
| LAH03 | BOOL | Boia de nível máximo tanque 03 |
| LAH04 | BOOL | Boia de nível máximo tanque 04 |
| RES_GERAL_S | BOOL | Reset Geral via Supervisório |
| RES_GERAL_I | BOOL | Reset Geral via IHM |
| BOIA_ON | BOOL | Alerta de indicação do estado da planta |
| EMERG_ON | BOOL | Sinaliza o estado da planta |

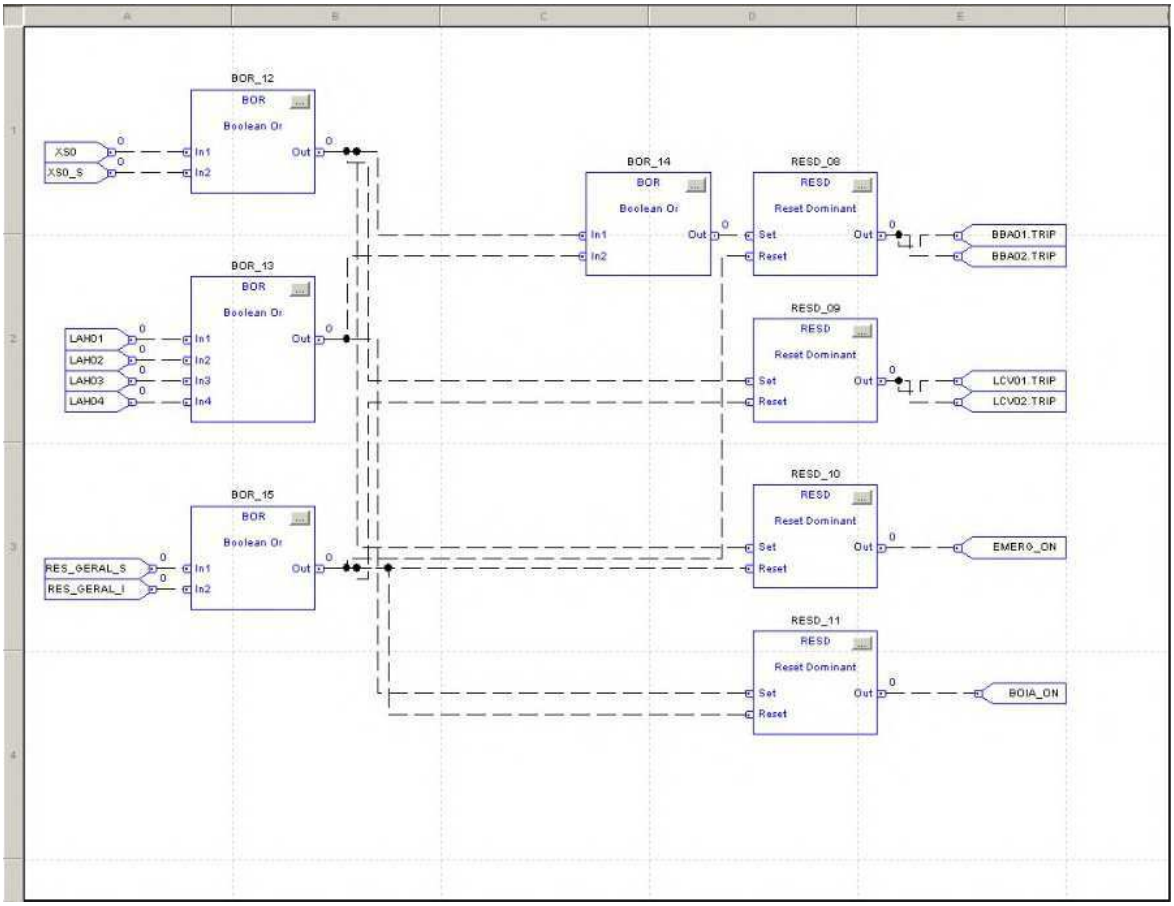
Fonte: Elaborado pelo autor

Figura 24 – SEGURANCA em LD




Fonte: Elaborado pelo autor

Figura 25 – SEGURANCA em FBD




Fonte: Elaborado pelo autor

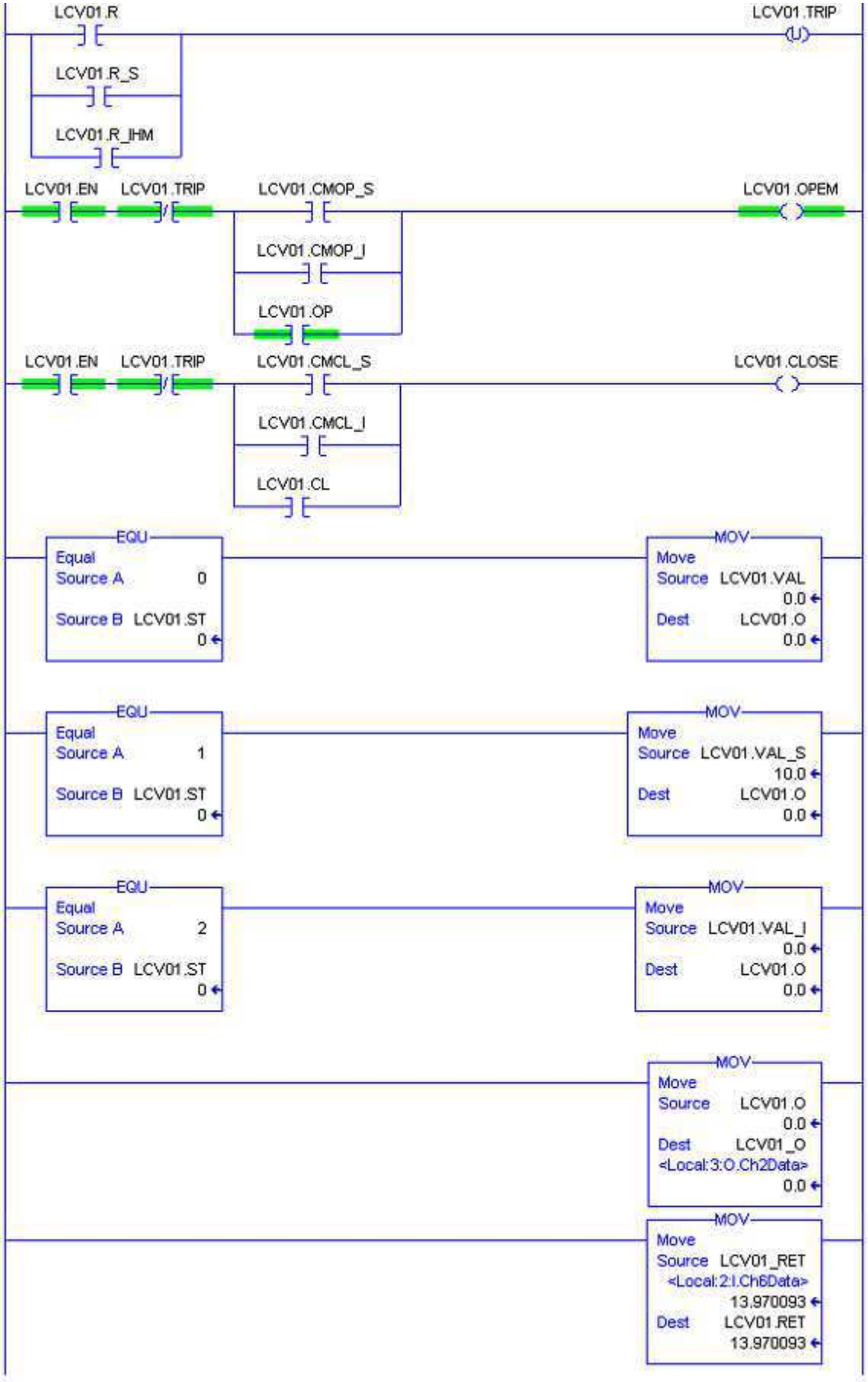
Na tabela 4, pode-se observar as *tags* utilizadas na sub-rotina SEGURANCA, que é a segunda sub-rotina chamada pela *main*. É ilustrado nas Figuras 24 e 25, é possível visualizar as botoeiras de emergência geral, que são usadas para desarmar a bomba e as válvulas elétricas, assim como para acionar o sinalizador do estado da planta. Após isso, os quatro contatos associados aos 4 sensores de nível máximo dos tanques (LAH0x) também são posicionados de modo a desarmar as duas bombas e ativar o alerta de indicação do estado da planta (BOIA_ON). Todas as referidas saídas só são resetadas quando a *tag* de *reset* geral é ativada (RES_GERAL_S ou RES_GERAL_I).

A programação da sub-rotina SEGURANCA, em FBD segue a mesma ideia da implementação do BBA_CONTROL que é utilizando os blocos *BOR* e *RESD*.

A terceira sub-rotina chamada pela *main* realiza o controle das válvulas (VALV). Em LD (Figura 26), as entradas LCV0x.R, LCV0x.R_S e LCV0x.R_IHM são usadas para resetar o desarme da válvula por meio da saída LCV0x.TRIP (UNLATCH). Este mesmo contato é usado em série com o habilita válvula (LCV0x.EN) e com o contato de comando de abertura total da válvula (LCV0x.CMOP_S, LCV0x.CMOP_I e LCV0x.OP) para ativar a saída de controle da válvula, acionando a bobina que a abre (LCV0x.OPEM).

Um comando similar é usado para ativar a saída de controle da válvula por meio da bobina que a fecha (LCV0x.CLOSE), usando para isso os contatos LCV0x.CMCL_S, LCV0x.CMCL_I e LCV0x.CL.

Figura 26 – VALV em LD



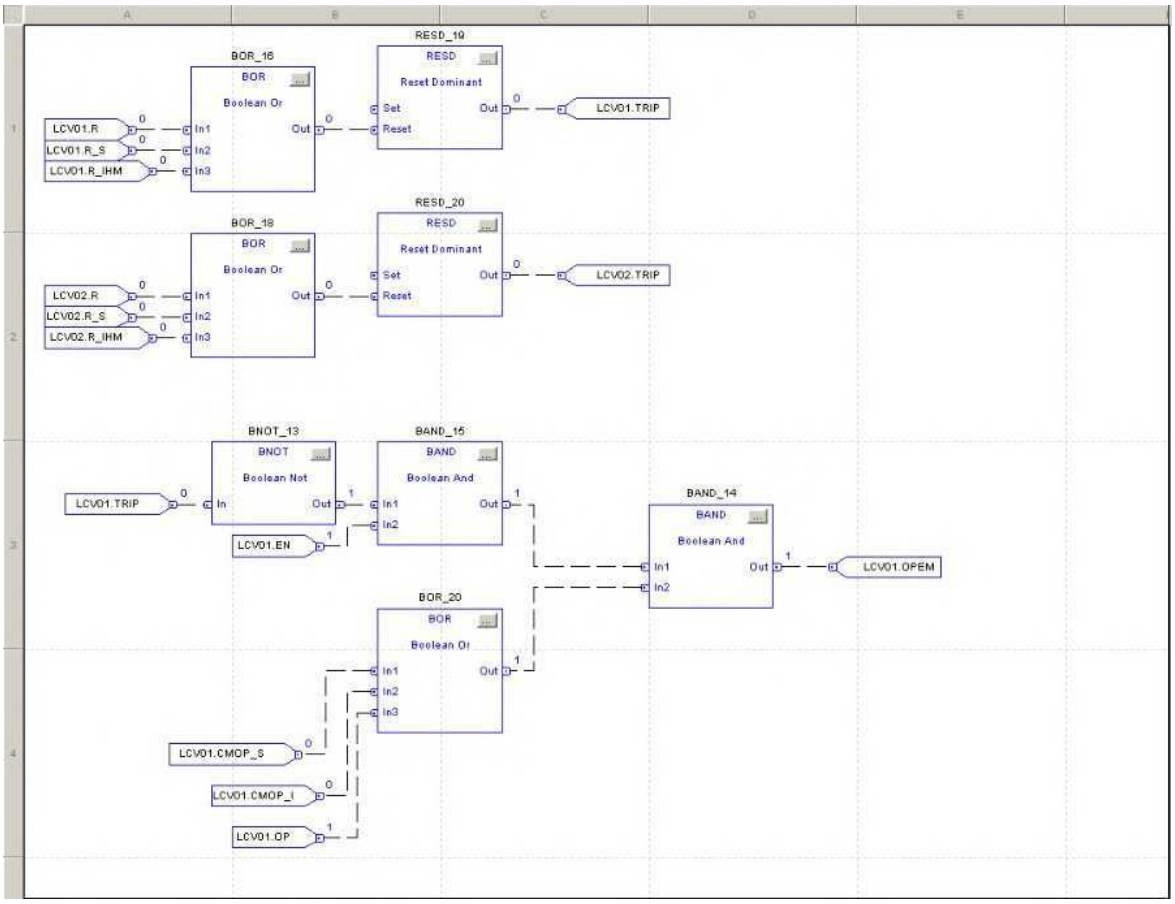

Fonte: Elaborado pelo autor

Após esses procedimentos iniciais, uma lógica de comparação é organizada para definir a fonte responsável por comandar o grau de abertura das válvulas, usando para isso a tag LCV0x.ST. Caso ela seja igual a 0 a fonte será o CLP; se for 1, o supervisório; e se for 2, a IHM. A tag usada para controlar o grau de abertura da válvula (vazão de saída de água dos tanques 2 e 3) é LCV0x_O. A sub-rotina é finalizada com uma instrução de retorno de posição da válvula, usando a tag LCV0x.RET.

Em FBD (Figuras 27 e 28), o *reset* do LCV0x.TRIP é realizado através do bloco *RESD*. Para acionar a saída de controle da válvula (LCV0x.OPEM), é seguida uma lógica semelhante à da saída física digital "O". Para que a válvula seja ativada, é necessário que o sinal "EN" esteja ativo, o "TRIP" esteja desativado, e também que os contatos de comando de abertura total da válvula (LCV0x.CMOP_S, LCV0x.CMOP_I e LCV0x.OP) estejam em condição favorável.

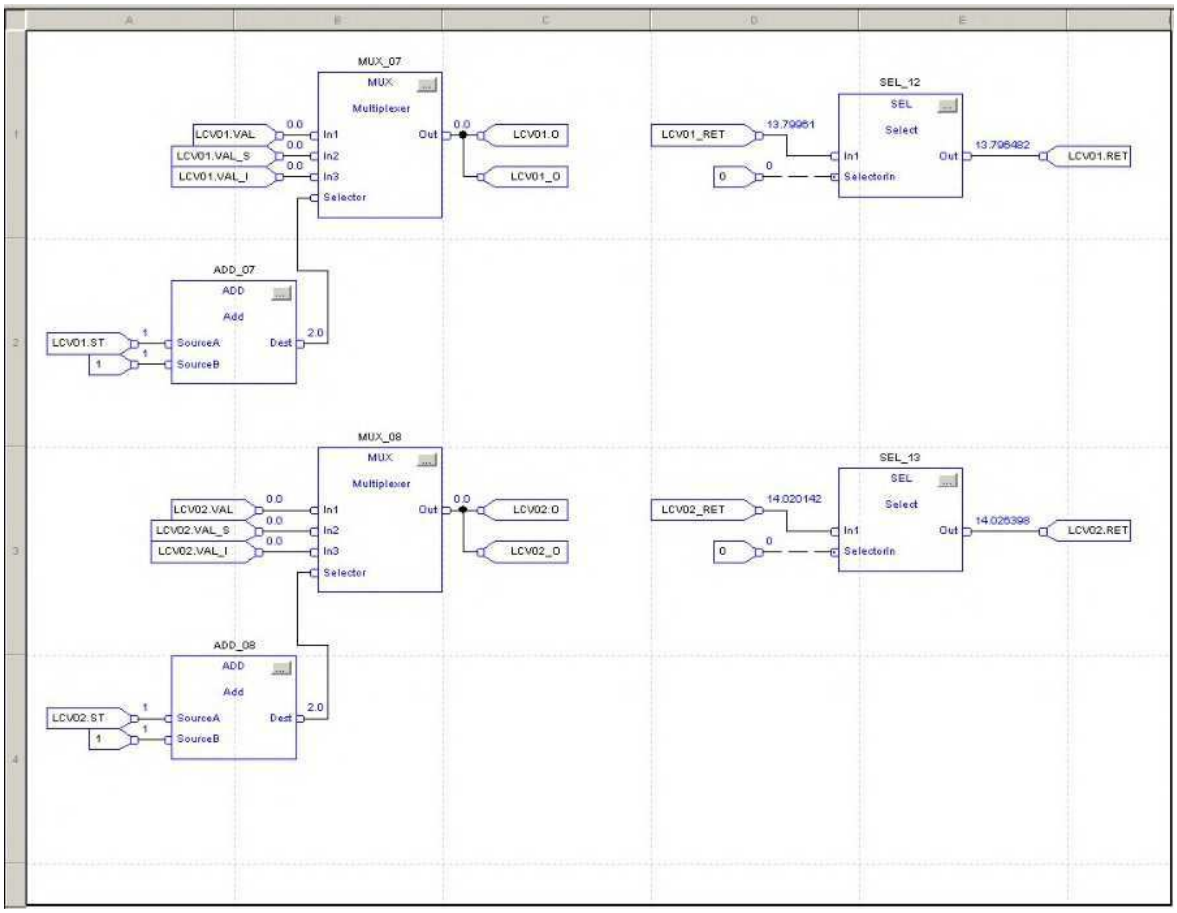
Para controlar o grau de abertura das válvulas, foram empregados MUX's. As associações das saídas são realizadas por meio do bloco SEL, garantindo uma seleção precisa da fonte de entrada de acordo com as condições estabelecidas.

Figura 27 – VALV em FBD




Fonte: Elaborado pelo autor

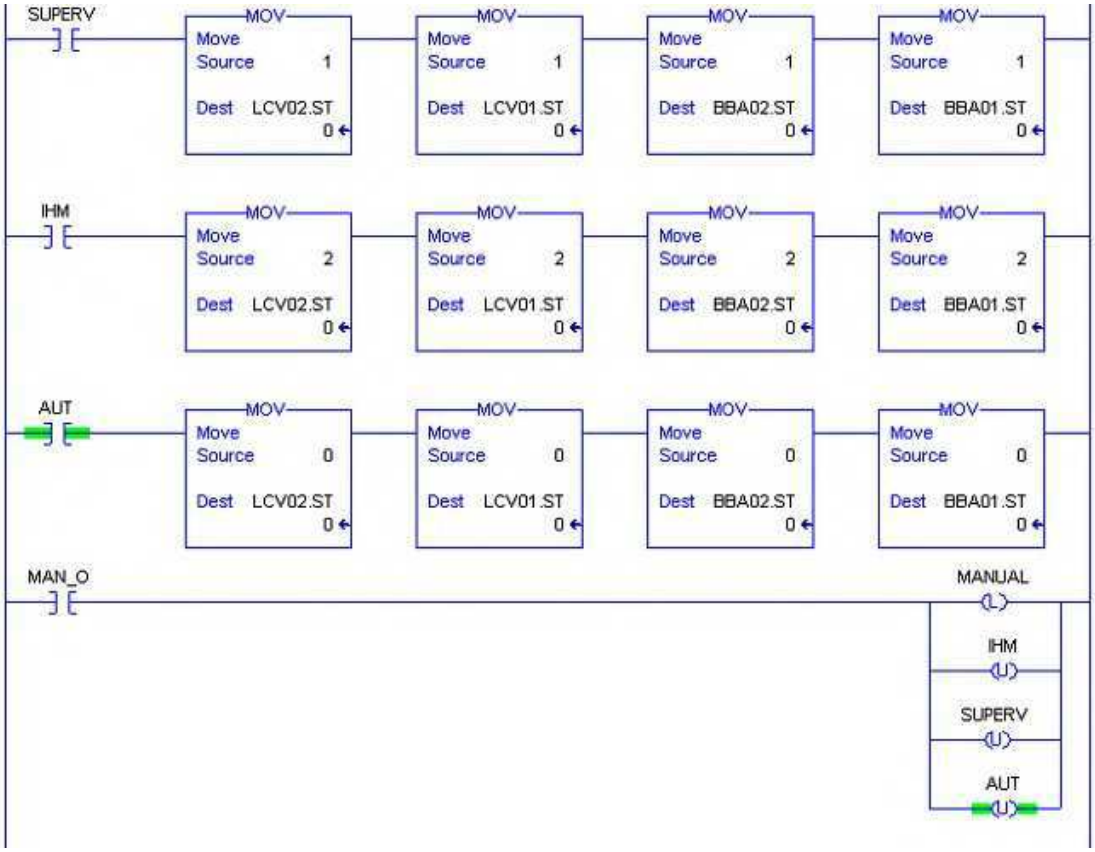
Figura 28 – VALV em FBD




Fonte: Elaborado pelo autor

Finalizada a sub-rotina de controle das válvulas, a rotina principal chama a sub-rotina dos modos de operação (MOD_OPER) (Figuras 29 e 30).

Figura 29 – MOD_OPER em LD




Fonte: Elaborado pelo autor

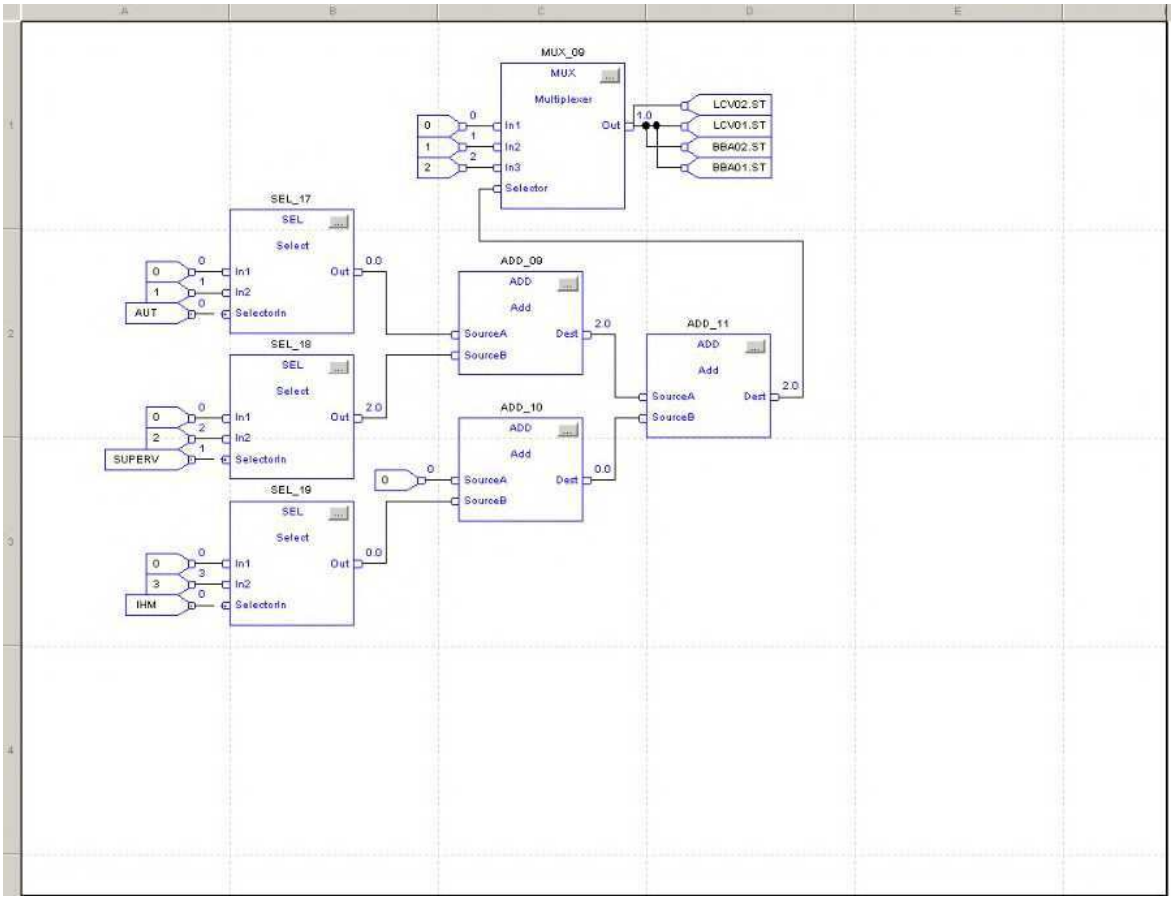
Figura 30 – MOD_OPER em LD


Fonte: Elaborado pelo autor

Em relação à programação em *ladder*, a ativação da *tag* SUPERV habilita o modo supervisório, a ativação da *tag* IHM habilita o modo IHM e a ativação da *tag* AUT habilita o modo automático (CLP). Cada uma dessas *tags* influência no dado transferido para as *tags* usadas nas comparações das rotinas descritas anteriormente, são elas: LCV0x.ST e BBA0x.ST. O contato MAN_O é usado para ativar a indicação do estado de funcionamento MANUAL, enquanto desativa a indicação dos demais estados. O mesmo raciocínio é usado com os contatos SUP_O, IHM_O e AUT_O.

Agora, na abordagem em FBD (conforme mostrado na Figura 31), é necessária uma lógica que utiliza os blocos SEL, ADD e MUX para possibilitar a alteração do status do motor.

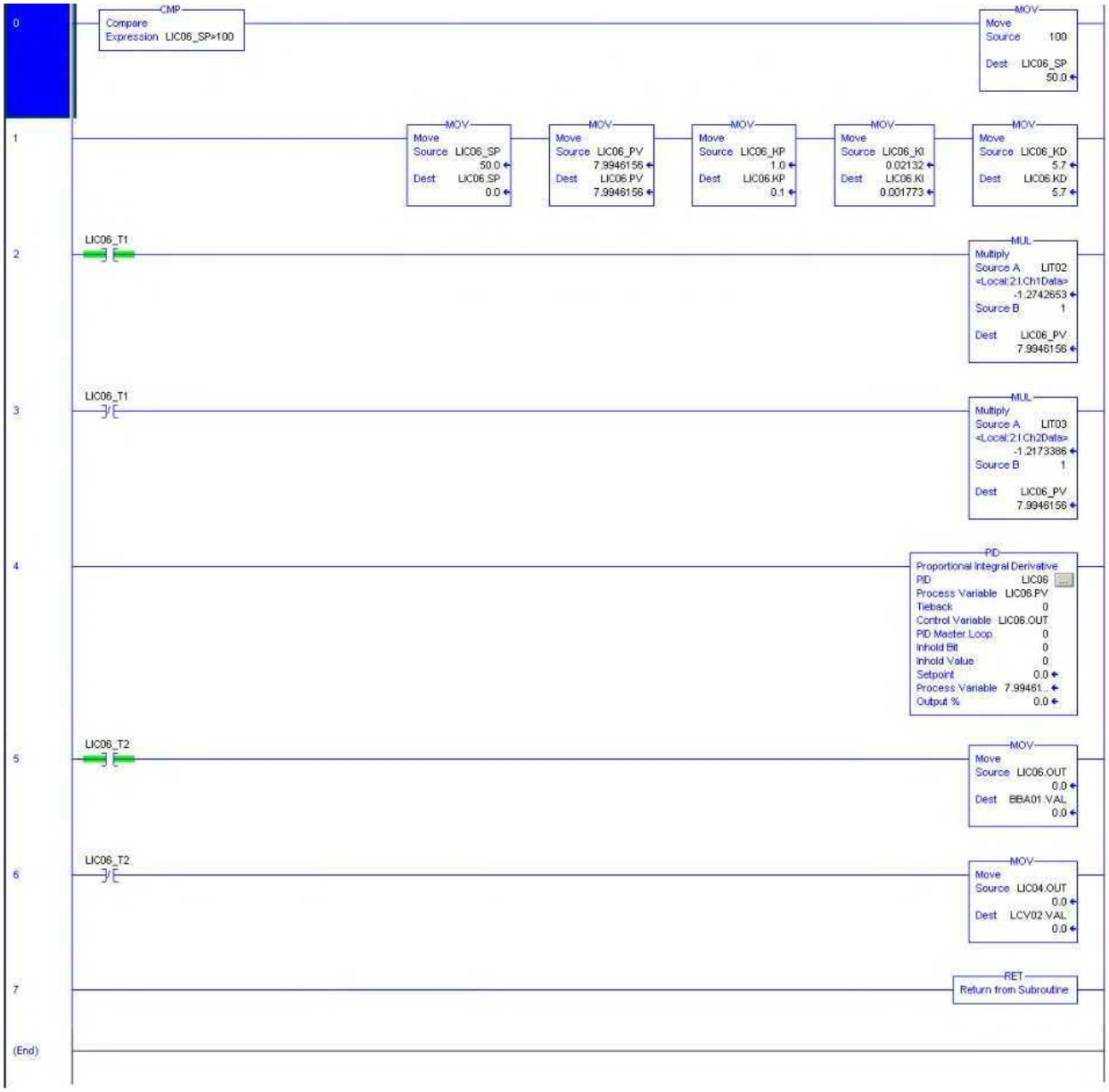
Figura 31 – MOD_OPER em FBD




Fonte: Elaborado pelo autor

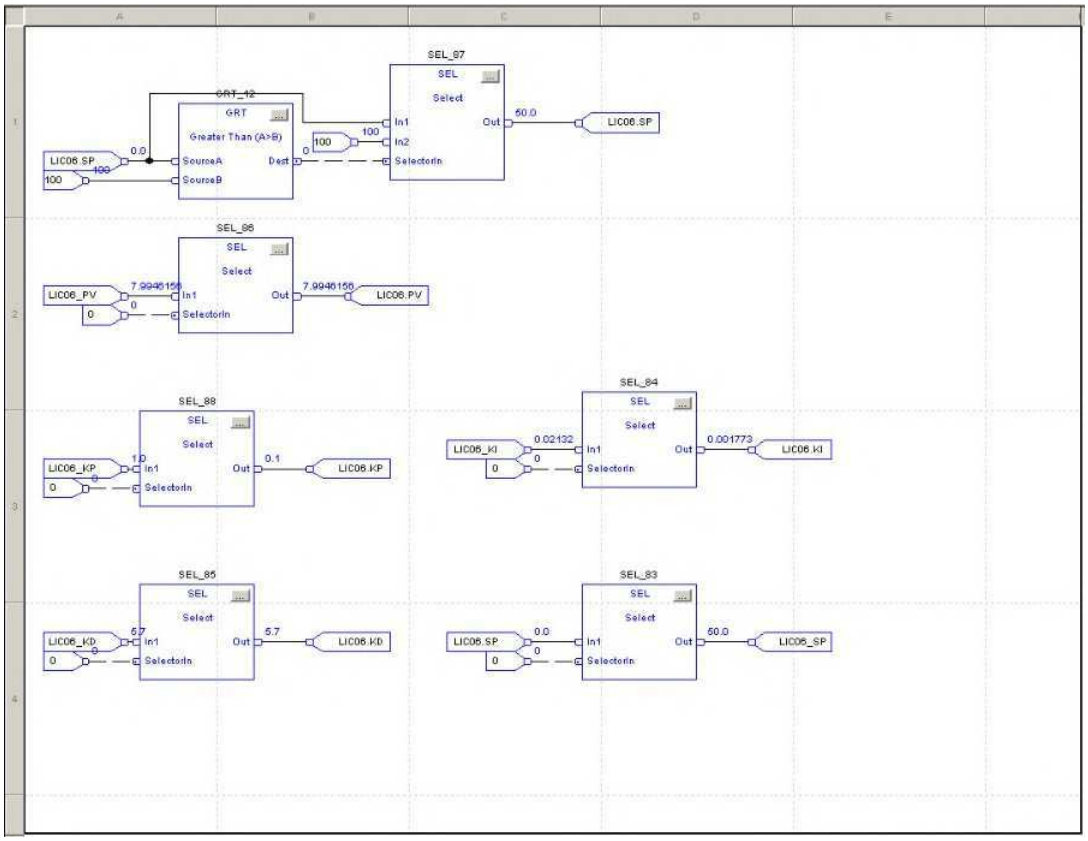
Finalizada a sub-rotina dos modos de operação, a rotina principal dá continuidade a sequência de controle. A partir desse ponto, as sub-rotinas de controle do nível dos tanques são chamadas pela rotina principal partir da habilitação da tag correspondente (LIC01A_EN, LIC01B_EN, LIC01C_EN, LIC01D_EN, LIC03A_EN, LIC04_EN, LIC05_EN e LIC05_T, LIC06_EN e LIC06_T2). Neste projeto, optamos por utilizar apenas a sub-rotina LIC06 (Figuras 32, 33, 34, 35 e 39), devido à maior precisão do sensor de nível do tanque 2. Esta sub-rotina controla os tanques 2 e 3 através da bomba 1 e da válvula 2.

Figura 32 – LIC06 em LD



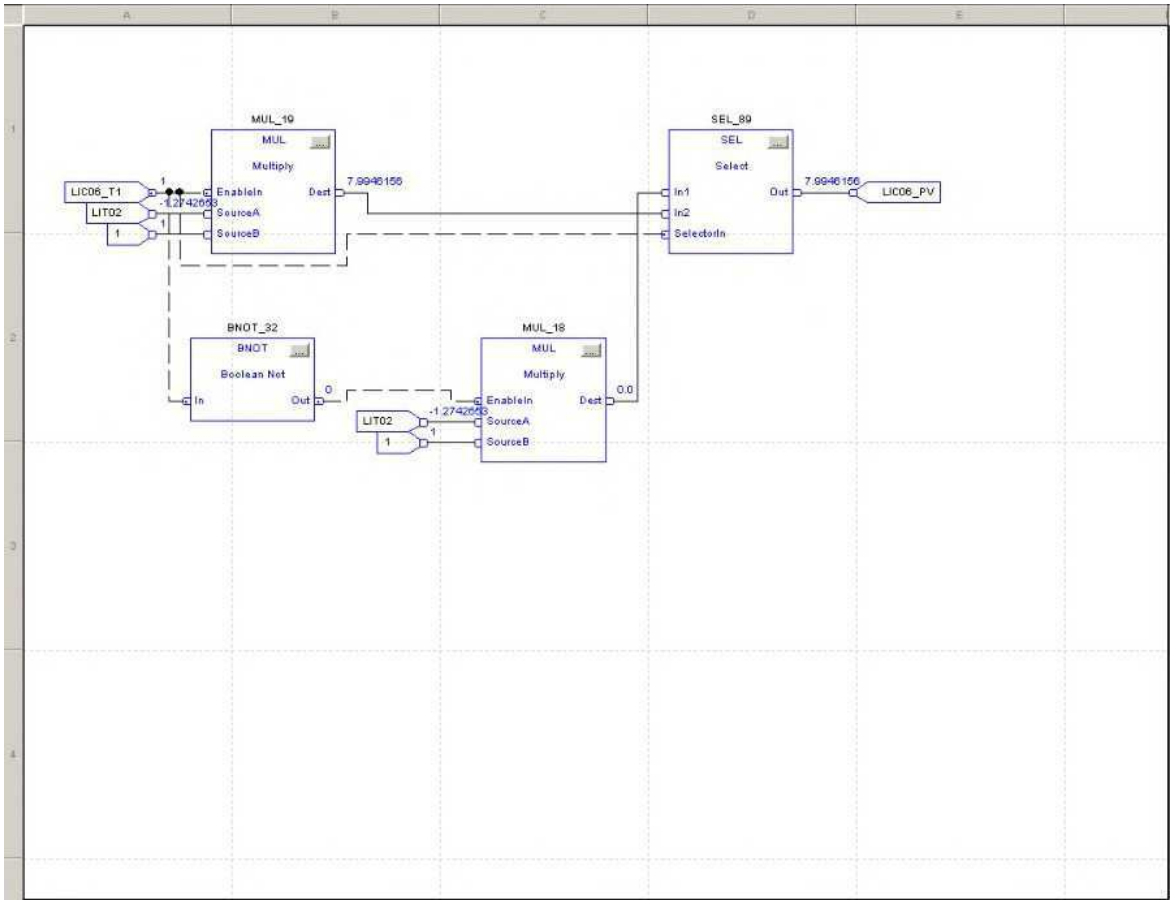

Fonte: Elaborado pelo autor

Figura 33 – LIC06 em FBD



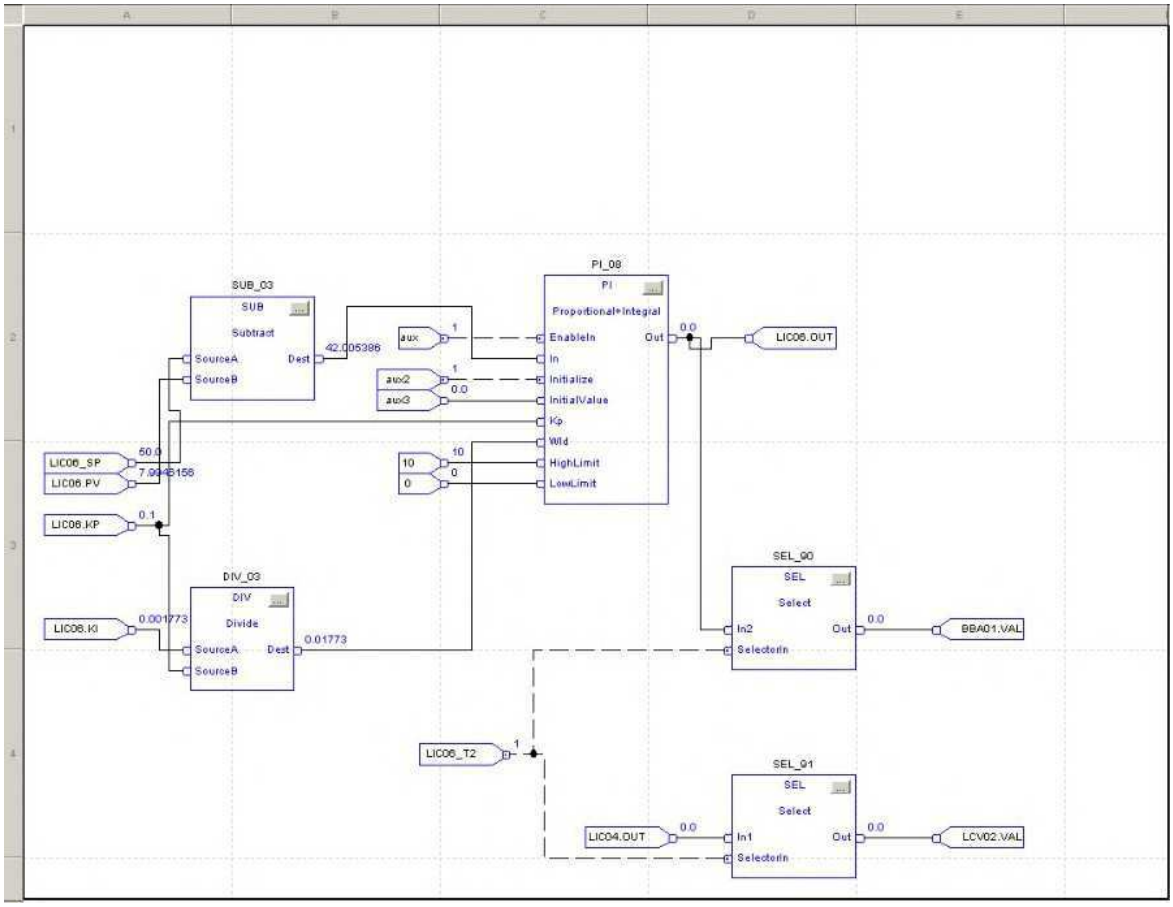

Fonte: Elaborado pelo autor

Figura 34 – LIC06 em FBD



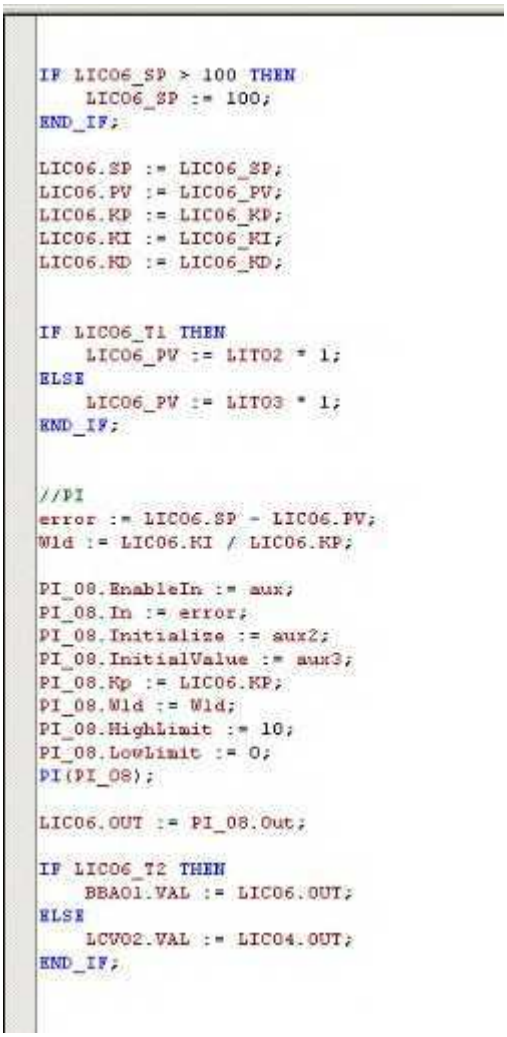

Fonte: Elaborado pelo autor

Figura 35 – LIC06 em FBD




Fonte: Elaborado pelo autor

Figura 36 – LIC06 em ST



```
IF LIC06_SP > 100 THEN
    LIC06_SP := 100;
END_IF;

LIC06.SP := LIC06_SP;
LIC06.PV := LIC06_PV;
LIC06.KP := LIC06_KP;
LIC06.KI := LIC06_KI;
LIC06.KD := LIC06_KD;


IF LIC06_T1 THEN
    LIC06_PV := LIT02 * 1;
ELSE
    LIC06_PV := LIT03 * 1;
END_IF;


//PI
error := LIC06.SP - LIC06.PV;
Wld := LIC06.KI / LIC06.KP;

PI_08.EnableIn := aux;
PI_08.In := error;
PI_08.Initialize := aux2;
PI_08.InitialValue := aux3;
PI_08.Kp := LIC06.KP;
PI_08.Wld := Wld;
PI_08.HighLimit := 10;
PI_08.LowLimit := 0;
PI(PI_08);

LIC06.OUT := PI_08.Out;

IF LIC06_T2 THEN
    BBA01.VAL := LIC06.OUT;
ELSE
    LCV02.VAL := LIC04.OUT;
END_IF;
```

Fonte: Elaborado pelo autor

Nesta sub-rotina, tanto em LD, FBD quanto em ST, uma instrução de segurança é executada inicialmente, realizando uma comparação do *setpoint* da malha de controle de nível. Caso seu valor exceda 100, é reduzido para 100. Em seguida, é efetuada uma multiplicação do valor armazenado na entrada analógica correspondente ao transmissor dos tanques 2 e 3 por 1. O resultado é então armazenado na *tag* LIC06_PV, responsável por conter o valor atual do nível do tanque 2. A escolha do tanque a ser controlado depende da ativação da entrada LIC06_T2: se estiver ativa, a bomba 2 é utilizada; caso contrário, a válvula 2 é acionada para controlar o nível.

Além de permitir que tanto a bomba 2 quanto a válvula 2 sejam utilizadas para controlar o nível do tanque, essa sub-rotina também possibilita selecionar qual tanque pode ser controlado, escolhendo entre os tanques 2 e 3, utilizando a entrada LIC06_T1. A mesma sequência de código é completada pela movimentação de dados para montar o conjunto de dados do *data type* LIC06, responsável pela ação de controle (dados do controlador PID, PV e SP). Na linha seguinte, é utilizado o bloco do controlador PID com

os ajustes necessários.

Em LD, foi empregado um Controlador PID, enquanto em FBD e ST foi utilizado um Controlador PI, devido a problemas com o bloco PIDE em FBD.

## 3.3 Identificação e Controle

Na programação original, o controlador estava sem os parâmetros de sintonia. Assim, para obter os parâmetros foram realizadas a identificação do modelo e a partir desse modelo o controlador foi projetado.

Foi utilizado o *software* BR-Tuning (anexo A) para realizar a identificação e sintonia da malha de controle do controlador. O BR-Tuning é uma ferramenta para análise e otimização de sistemas de controle, oferecendo suporte para diversas técnicas de identificação e sintonia de controladores.

Para a identificação do modelo da planta, foi aplicado a técnica a partir da resposta ao degrau utilizando método dos mínimos quadrados (SREE; SRINIVAS; CHIDAMBARAM, 2004). Esta técnica é amplamente utilizada em sistemas de controle devido à sua eficácia em estimar parâmetros de modelos lineares a partir de dados de entrada e saída.

Após a identificação do modelo da planta, realizamos a sintonia do controlador utilizando três técnicas distintas, cada uma com suas próprias características: SIMC, Zak-Friedman (ZF) e Kano-Ogawa (KO). A escolha das mesmas foi dada devido a disponibilidade do *software* trabalhado e de referências acadêmicas. Cada uma dessas técnicas busca otimizar o desempenho do sistema de controle, mas através de abordagens distintas e com diferentes ênfases .

A técnica SIMC baseia-se no princípio de controle de modelo interno e é conhecida por sua simplicidade e eficácia na obtenção de um bom desempenho de controle, mantendo uma resposta rápida e minimizando o *overshoot* (JELALI, 2012).

A técnica Zak-Friedman propõe um método de sintonia que considera a robustez e a rejeição a distúrbios do sistema. Este método é projetado para obter um desempenho consistente mesmo na presença de variações e incertezas nos parâmetros do modelo (FRIEDMAN, 1994).

A técnica de Kano-Ogawa foca na otimização da resposta do sistema considerando tanto a precisão quanto a estabilidade. Este método oferece uma maneira sistemática de ajustar os parâmetros do controlador, levando em conta o comportamento dinâmico da planta e os requisitos de desempenho desejados (KANO; OGAWA, 2010).

# 4 Resultados e Discussões

O presente capítulo apresenta os resultados da implementação da nova lógica de controle na planta industrial, bem como as discussões relacionadas aos resultados obtidos. Inicialmente, destaca-se a transformação da programação original em *ladder* para diagrama de blocos de função e texto estruturado. Além disso, identificou-se a necessidade de sintonizar o controlador, que anteriormente operava em malha aberta. Essa sintonia foi importante para verificar as diferenças na implementação e no funcionamento do controlador ao utilizar diferentes linguagens de programação.

A ausência de sintonização adequada do controlador foi uma limitação identificada na programação original. Assim, ao sintonizar o controlador através do *software* BR-Tuning, observou-se melhorias significativas no desempenho do sistema. Para determinar os ganhos do controlador PI, conectamos a malha LIC06 ao *software* para identificar e ajustar os ganhos do controlador.

Na configuração, a variável de processo (*Process Variable* - PV) é o nível do tanque 2, enquanto a variável manipulada (*Manipulated Variable* - MV) é a frequência do inversor. No contexto do nosso projeto, uma unidade de amplitude de MV equivale a uma alteração de 6 Hz na frequência.

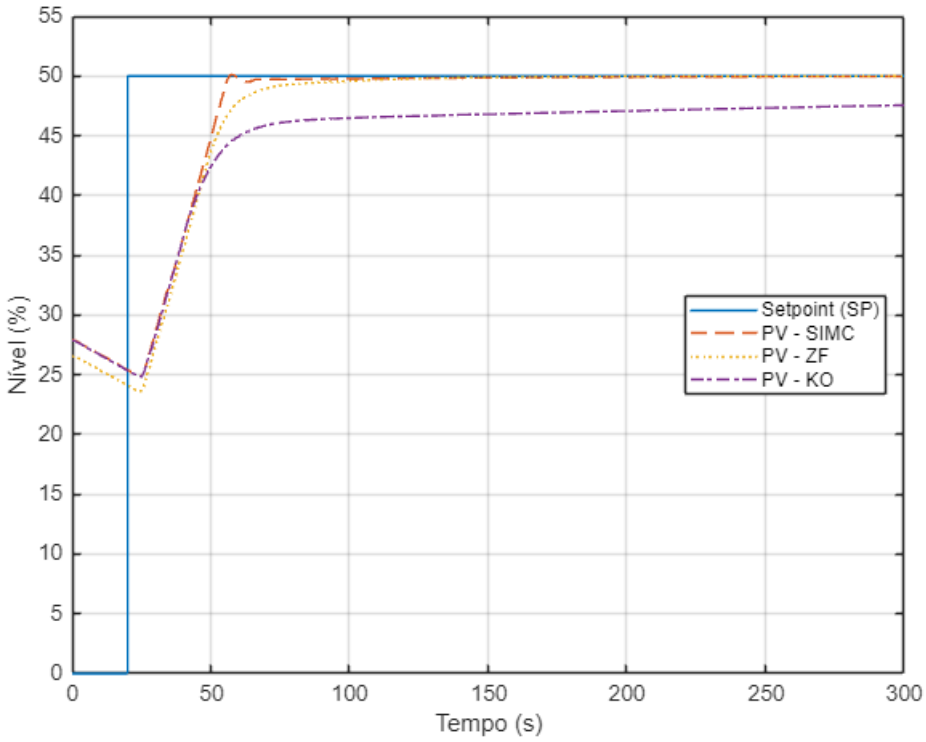
Para o processo de identificação em malha aberta, a partir do experimento de teste do degrau realizado na malha de controle, foi possível realizar o projeto do controlador PI. Foi escolhido arbitrariamente uma frequência de 18 Hz para MV. Após alcançar a estabilização, obtivemos os resultados da Tabela 5.

Tabela 5 – Ganhos do Controlador PI

| Método | Kp | Ki |
|---|---|---|
| SIMC | 4,34 | 0,04014 |
| Zak-Friedman | 1 | 0,02132 |
| Kano-Ogawa | 0,6448 | 0,001773 |

Fonte: Elaborado pelo autor

Figura 37 – Variáveis do processo em cada técnica



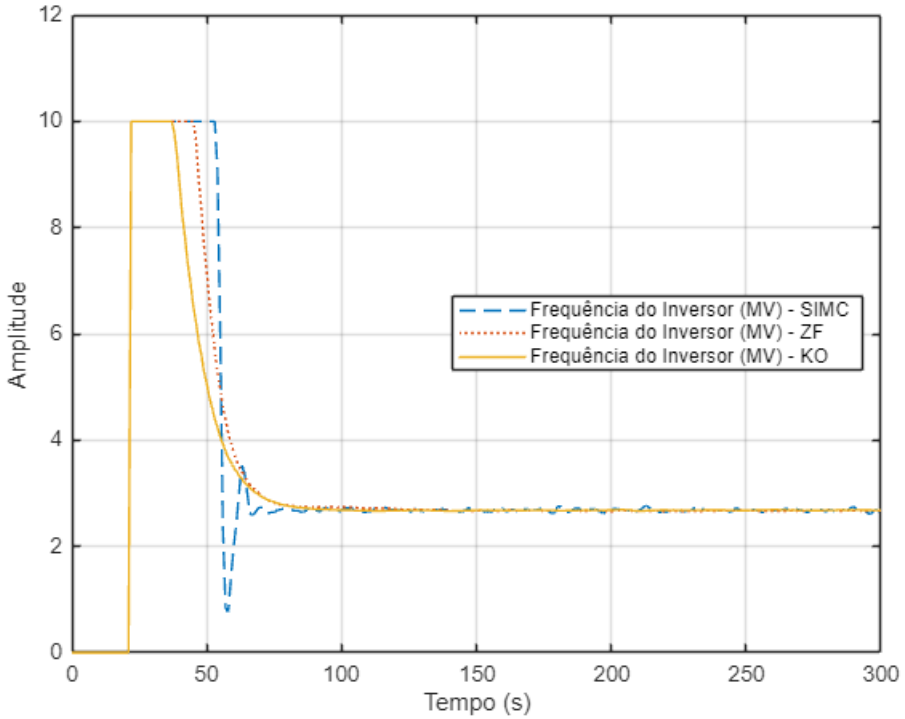

Fonte: Elaborado pelo autor

Os gráficos foram gerados com os dados obtidos após a execução da programação em FBD. Além disso, a implementação utilizando a linguagem em ST também foi realizada. Na Figura 37, visualizamos a comparação entre o *setpoint* e as variáveis do processo obtidos por meio das três técnicas de sintonização utilizadas: SIMC, Zak-Friedman e Kano-Ogawa. O *setpoint* foi aplicado no tempo de 20s para 50% do nível do tanque, que significa o valor desejado para o nível do tanque 2, ou seja, é a referência que o sistema de controle tenta atingir e manter ao longo do tempo.

As variáveis manipuladas são mostradas na Figura 38, onde são visualizadas nas três técnicas de sintonização trabalhadas. As mudanças na frequência do motor quando é colocado o *setpoint* foram bruscas em todos os casos, refletindo uma abordagem agressiva para ajustar o PV ao *setpoint* rapidamente, onde pode resultar em um maior desgaste dos componentes do motor. Esse comportamento ressalta a necessidade de equilibrar a velocidade de resposta do sistema com a vida útil e a integridade dos equipamentos

O gráfico da Figura 38 revela diferenças nas características das variáveis manipuladas entre as técnicas SIMC, Zak-Friedman e Kano-Ogawa.

Figura 38 – Variáveis manipuladas em cada técnica




Fonte: Elaborado pelo autor

Para avaliar o desempenho das técnicas projetadas, foram calculadas diversas medidas que refletem a qualidade e a eficiência do sistema de controle, como ilustrado na Tabela 6. Entre essas medidas, destacam-se o *overshoot* (OS), que representa a porcentagem de ultrapassagem do valor desejado durante a resposta transitória e é calculado pela fórmula:

$$OS = \left( \frac{Y_{\text{pico}} - Y_{\text{final}}}{Y_{\text{final}}} \right) \times 100\% \tag{4.1}$$

Onde $Y_{\text{pico}}$ representa o valor máximo alcançado pela resposta do sistema durante a resposta transitória, e $Y_{\text{final}}$ representa o valor final ou estacionário da resposta do sistema.

O tempo de pico, que é o tempo necessário para que a resposta alcance o pico máximo durante a resposta transitória é calculado pela fórmula:

$$T_p = t_{\text{pico}} - t_{\text{inicial}} \tag{4.2}$$

O tempo de subida ($T_{\text{r}}$), que indica o tempo necessário para o sistema atingir 90% de sua resposta final após uma mudança de referência é calculado pela fórmula:

$$T_r = t_{90\%} - t_{\text{inicial}} \tag{4.3}$$

O tempo de acomodação, que é o tempo necessário para que a resposta do sistema se estabilize dentro de uma faixa de tolerância específica ao redor do valor desejado é calculado pela fórmula:

$$T_s = t_{\text{final}} - t_{\text{estabilização}} \tag{4.4}$$

E o erro absoluto integral (IAE - *Integral Absolute Error*), que representa a soma dos valores absolutos dos erros ao longo de um determinado período de tempo é calculado pela fórmula:

$$IAE = \int_0^\infty |e(t)|\, dt \tag{4.5}$$

Tabela 6 – Medidas de desempenho para cada técnica

| Técnicas | *Overshoot* (%) | Tempo de Pico (s) | Tempo de Subida (s) | Tempo de Acodomação (s) | IAE |
|---|---|---|---|---|---|
| SIMC | 0,2985 | 58 | 50,2866 | 55,0735 | 595 |
| Zak-Friedman | 0,0275 | 295 | 52,2513 | 70,5171 | 671 |
| Kano-Ogawa | 0 | 303 | 51,0060 | 121,1841 | 1366 |

Fonte: Elaborado pelo autor

Primeiramente, observa-se que todas as técnicas conseguiram manter o *overshoot* em valores baixos, o que indica uma resposta do sistema bem controlada e próxima do ponto de ajuste. Esse é um aspecto positivo, pois o *overshoot* elevado pode resultar em instabilidades e oscilações indesejadas no sistema.

Em relação ao tempo de pico, observa-se que a técnica SIMC apresentou o menor valor, seguida por Zak-Friedman e Kano-Ogawa. Isso sugere que a técnica SIMC proporciona uma resposta mais rápida em relação às outras, atingindo o valor máximo mais cedo. No que diz respeito ao tempo de subida, observa-se uma pequena diferença entre as técnicas, com Zak-Friedman apresentando o maior tempo de subida e SIMC o menor. O tempo de acomodação, também apresenta diferenças entre as técnicas, com Kano-Ogawa mostrando o maior tempo de acomodação.

Por fim, o erro absoluto integral (IAE) fornece uma medida quantitativa do desempenho global do sistema ao longo do tempo. Observa-se que a técnica SIMC apresenta o menor valor de IAE, o que sugere um desempenho global mais eficiente em relação às outras técnicas.

Entretanto, todos os ganhos proporcionais (Kp) estavam consideravelmente elevados, resultando em uma resposta agressiva do motor. Esse aumento rápido de frequência, indo

de 0 a 60 quase instantaneamente, pode ter repercussões negativas significativas. Essa abordagem agressiva pode levar a um estresse excessivo nos componentes mecânicos e elétricos do sistema, aumentando a probabilidade de falhas prematuras e reduzindo a vida útil do equipamento.

# 5 Conclusão

Este trabalho evidenciou o desenvolvimento da programação em diagrama de blocos de função e texto estruturado, substituindo a abordagem anterior em *ladder*, para um sistema de controle para uma planta industrial em escala laboratorial.

A utilização de diferentes linguagens de programação, como diagrama de blocos de função e texto estruturado, trouxe benefícios distintos e desafios específicos. Enquanto o diagrama de blocos de função ofereceu uma representação visual intuitiva do processo de controle, facilitando a compreensão, sua complexidade aumentou significativamente com o crescimento do sistema, tornando a manutenção e a modificação mais trabalhosas.

Por outro lado, o texto estruturado proporcionou uma abordagem mais flexível, permitindo uma maior modularidade e reutilização de código. No entanto, o entendimento inicial desta linguagem foi mais trabalhosa.

Além disso, a sintonização do controlador para a malha de controle do Tanque 2 foi realizada com sucesso utilizando o software BR-Tuning. A utilização de diferentes técnicas de sintonia permitiu uma análise comparativa das performances do sistema. Os resultados mostraram que a técnica SIMC ofereceu a melhor performance geral, no entanto, todas as técnicas apresentaram valores elevados de ganhos proporcionais (Kp), indicando uma resposta agressiva do sistema, e prejudicando o funcionamento do motor.

Através deste projeto, destacou-se a importância da sintonização adequada do controlador e os benefícios da utilização de diferentes linguagens de programação para sistemas de controle industrial. O desenvolvimento da programação para FBD e ST, aliada à sintonização eficiente, resultou em um sistema mais robusto e adaptável às demandas operacionais. Portanto, a implementação da programação em outras linguagens no controle da planta foi realizada com sucesso e o funcionamento do processo se manteve conforme o esperado.

Em conclusão, este projeto alcançou seus objetivos iniciais de analisar os requisitos específicos da planta de nível e identificar as funcionalidades e estratégias de controle necessárias para sua programação para melhorar o controle da planta.

## 5.1 Trabalhos futuros

É possível explorar melhorias adicionais e expandir as aplicações do sistema. As sugestões incluem:

- Implementar a programação em lista de instruções e sequenciamento gráfico de

funções;

- Estudar novas técnicas de sintonia de controladores;
- Explorar a criação de soluções que utilizem o padrão OPC para facilitar a interoperabilidade entre diferentes sistemas e dispositivos de controle;
- Realizar o controle de todas as malhas da planta industrial.

# Referências Bibliográficas

ÅSTRÖM, K. J.; HÄGGLUND, T. *Advanced PID control*. Research Triangle Park, NC - 27709: ISA-The Instrumentation, Systems and Automation Society, 2006. Citado na página 13.

BR-TUNING. 2015. Acesso em 4 de maio, 2024. Disponível em: <http://brtuning.com.br/sobre>. Citado 2 vezes nas páginas 53 e 54.

BRYAN, L. A.; BRYAN, E. A. *Programmable controllers: theory and implementation*. Marietta, Georgia - 30067: Industrial Text Company, 1997. Citado 3 vezes nas páginas 1, 11 e 12.

FRIEDMAN, Y. Z. Tuning of averaging level controller. *Hydrocarbon Processing Journal*, 1994. Citado na página 43.

GUIMARÃES, H. C. F. *Norma IEC 61131-3 para Proramação de Controladores Programáveis: Estudo e Aplicação*. Trabalho de Conclusão de Curso — Universidade Federal do Espírito Santo, Vitória, 2005. Graduação em Engenharia Elétrica. Citado na página 6.

HALIM, A. H.; ISMAIL, I. Tree physiology optimization on siso and mimo pid control tuning. *Neural Computing and Applications*, Springer, v. 31, n. 11, p. 7571–7581, 2019. Citado na página 14.

HANSSEN, D. H. *Programmable logic controllers: a practical approach to IEC 61131-3 using CODESYS*. Chichester, United Kingdom: John Wiley & Sons, 2015. Citado 2 vezes nas páginas 8 e 11.

JELALI, M. Control performance management in industrial automation: assessment, diagnosis and improvement of control loop performance. Springer Science & Business Media, 2012. Citado na página 43.

JUNIOR, P. R. R. d. S. J. J. A. N. B. *Modelagem dos tanques de nível do Laboratório de Controle de Processos*. Campina Grande: LIEC, 2022. 23 p. Citado 6 vezes nas páginas 16, 17, 18, 19, 20 e 21.

KANO, M.; OGAWA, M. The state of the art in chemical process control in japan: Good practice and questionnaire survey. *Journal of Process control*, Elsevier, v. 20, n. 9, p. 969–982, 2010. Citado na página 43.

LADDER Logic Programming Examples. 2023. Acesso em 2 de maio, 2024. Disponível em: <https://ladderlogicworld.com/ladder-logic-programming-examples/>. Citado 2 vezes nas páginas 9 e 10.

MAIA, I. D. d. C. D. *Estágio Supervisionado no Laboratório de Instrumentação Eletrônica e Controle (LIEC)*. Campina Grande: LIEC, 2016. v. 1. 29 p. Citado na página 16.

MESQUITA, B. D. de; JEFFERSON, A.; ANDRADE, R. V. Desenvolvimento de um sistema supervisório para uma planta didática de nível. 2012. Citado 2 vezes nas páginas 3 e 4.

NAMEKAR, S. A.; YADAV, R. Programmable logic controller (plc) and its applications. *International Journal of Innovative Research in TechnologY (IJIRT)*, v. 6, n. 11, p. 372–376, 2020. Citado na página 1.

ROMANOV, V. *Introduction to Function Block Programming in RSLogix 5000*. 2020. Acesso em 30 de abril, 2024. Disponível em: <https://www.solisplc.com/tutorials/function-block-programming>. Citado na página 8.

ROMANOV, V. *PLC Programming | How to Read Ladder Logic & Ladder Diagrams*. 2020. Acesso em 1 de maio, 2024. Disponível em: <https://www.solisplc.com/tutorials/how-to-read-ladder-logic>. Citado na página 9.

SEBORG, D. E. et al. *Process dynamics and control*. Chennai, India: John Wiley & Sons, 2016. Citado na página 13.

SREE, R. P.; SRINIVAS, M.; CHIDAMBARAM, M. A simple method of tuning pid controllers for stable and unstable foptd systems. *Computers & chemical engineering*, Elsevier, v. 28, n. 11, p. 2201–2218, 2004. Citado na página 43.

TIEGELKAMP, M.; JOHN, K.-H. *IEC 61131-3: Programming industrial automation systems*. New York: Springer, 2010. v. 166. Citado 3 vezes nas páginas 1, 7 e 8.

VENTER, J. *Getting started in Sequential Function Chart (SFC) Programming in RSLogix5000*. 2021. Acesso em 2 de maio, 2024. Disponível em: <https://www.solisplc.com/tutorials/sequential-function-chart-sfc-programming>. Citado na página 12.

# ANEXO A – BR-Tuning

O BR-Tuning é uma ferramenta completa para identificação, sintonia e avaliação de malhas de controle PID de uma unidade industrial (BR-TUNING, 2015).

Figura 39 – Visão Geral do BR-Tuning


Fonte: Elaborado pelo autor

Apresenta-se como uma poderosa ferramenta para otimização do desempenho das malhas de controle PID. Sendo de grande utilidade na melhoria de desempenho da unidade através da sintonia das malhas de controle PID, bem como no auxilio à partida de novas unidades (BR-TUNING, 2015).

O BR-Tuning foi desenvolvido pelos pesquisadores do Laboratório de Instrumentação Eletrônica e Controle do Departamento de Engenharia Elétrica da Universidade Federal de Campina Grande (LIEC/DEE/UFCG) em conjunto com pesquisadores do Centro de Pesquisas da PETROBRAS (CENPES) desde 2003 e está atualmente instalado e em uso em diversas unidades da PETROBRAS (BR-TUNING, 2015).

O BR-Tuning implementa algoritmos padrões encontrados na literatura e na indústria bem como outros algoritmos desenvolvidos pelas equipes da UFCG e do CENPES.

Com o BR- Tuning, o usuário é capaz de sintonizar, avaliar o desempenho, diagnosticar e efetuar manutenção em controladores PID instalados nas unidades industriais (BR-TUNING, 2015).

As técnicas de sintonia implementadas no BR-Tuning foram agrupadas conforme características comuns da informação necessária para a sintonia do controlador. Foram definidos os seguintes grupos (BR-TUNING, 2015):

- Resposta ao Degrau: técnicas que usam modelos de primeira-ordem com atraso e integrador com atraso.
- Ponto Crítico: técnicas que usam o ganho e período crítico do processo para a sintonia do controlador.
- Margens de Estabilidade: técnicas iterativas que necessitam da estimativa da margem de ganho e/ou da margem de fase da malha fechada para reprojetar o controlador.