

**UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ**

**LUCA PAPARELLA**

**SISTEMA PROGRAMÁVEL COM FERRAMENTA OPEN-SOURCE  
PARA ENSINO DE CONTROLE E AUTOMAÇÃO**

**DISSERTAÇÃO**

**CURITIBA**

**2021**

**LUCA PAPARELLA**

**SISTEMA PROGRAMÁVEL COM FERRAMENTA  
OPEN-SOURCE PARA ENSINO DE CONTROLE E  
AUTOMAÇÃO**

**Programmable system with open-source tool  
for teaching control and automation**

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Elétrica e Informática Industrial (CPGEI), da Universidade Tecnológica Federal do Paraná (UTFPR) como requisito parcial para obtenção do título de “Mestre em Ciências” .

Área de Concentração: Engenharia de Automação e Sistemas .

Orientador: Prof. Dr. Flavio Neves Junior

**CURITIBA**

**2021**

Esta licença permite compartilhamento, remixe, adaptação e criação a partir do trabalho, mesmo para fins comerciais, desde que sejam atribuídos créditos ao(s) autor(es).

Conteúdos elaborados por terceiros, citados e referenciados nesta obra não são cobertos pela licença.



[4.0 Internacional](#)



LUCA PAPARELLA

**SISTEMA PROGRAMÁVEL COM FERRAMENTA OPEN-SOURCE PARA ENSINO DE CONTROLE E AUTOMAÇÃO**

Trabalho de pesquisa de mestrado apresentado como requisito para obtenção do título de Mestre Em Ciências da Universidade Tecnológica Federal do Paraná (UTFPR). Área de concentração: Engenharia De Automação E Sistemas.

Data de aprovação: 10 de Dezembro de 2021

Prof Flavio Neves Junior, Doutorado - Universidade Tecnológica Federal do Paraná

Prof Joao Paulo Lima Silva De Almeida, Doutorado - Instituto Federal de Educação, Ciência e Tecnologia do Paraná (Ifpr)

Prof.a Lucia Valeria Ramos De Arruda, Doutorado - Universidade Tecnológica Federal do Paraná

Documento gerado pelo Sistema Acadêmico da UTFPR a partir dos dados da Ata de Defesa em 10/12/2021.

Dedico este trabalho a minha família, pelos  
momentos de ausência.

## **AGRADECIMENTOS**

A minha família, pelo carinho, incentivo e total apoio em todos os momentos da minha vida.

Ao meu orientador, que me mostrou os caminhos a serem seguidos e pela confiança depositada.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

A curiosidade traz inquietude à mente mesmo quando se está em total segurança e conforto.

Ela é, ainda, um mecanismo catalisador do aprendizado. Quando há o interesse por descobrir a resposta para um mistério, todos os sentidos são aguçados para que se encurte o caminho da descoberta.

(MENESTRINA; BAZZO, 2008)

## **RESUMO**

PAPARELLA, Luca. **Sistema programável com ferramenta open-source para ensino de controle e automação.** 2021. 127 f. Dissertação (Mestrado em Engenharia Elétrica e Informática Industrial) – Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

Esse trabalho desenvolve um sistema educacional para o aprendizado de controle e automação composto por um módulo open-hardware com Arduino, e software open-source como OpenPLC, ScadaBr e Python. O sistema permite aos alunos projetar, construir e testar seus conhecimentos visando despertar a curiosidade em aprimorar tantos os aspectos teóricos quanto práticos dessa área do conhecimento, além de propor atividades laboratoriais de controle que exploram conceitos ensinados em sala de aula. O sistema proposto permite modelagem, análise, projeto e teste de sistemas de controle variados em um ambiente que pode ser virtual ou real. No decorrer da dissertação será explanado o projeto e construção de um núcleo (*shield*) que irá converter a plataforma Arduino Uno R3 em um controlador lógico programável com funcionalidade básicas. Esse protótipo, auxiliado por software open-source, possibilita criar um ambiente versátil para controle e automação industrial. A metodologia proposta visa explorar os principais conceitos de eletrônica básica, programação de códigos de controle, comunicação entre dispositivos e análise de dados. Enfim, experimento práticos e medições, combinados a elementos reais e virtuais, serão feitos para validar e avaliar o sistema proposto, terminando com a discussão dos resultados obtidos.

**Palavras-chave:** Arduino. Microcontrolador. Protótipo. Hardware. Controle.

## ABSTRACT

PAPARELLA, Luca. **Programmable system with open-source tool for teaching control and automation.** 2021. 127 p. Dissertation (Master's Degree in Course Name) – Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

In this work, an open-hardware module with Arduino, and open-source software as Open-PLC, Scadabr and Python are used to engage students in designing, building and testing their own technology in order to arouse curiosity improving research and practical experiments, in addition to proposing laboratory control practices that explore concepts taught in the classroom. This system enables modeling, analysis, design and control testing in an environment that can be virtual or real. In the course of the dissertation will be explained the design and construction of a shield that will convert the Arduino Uno R3 platform into a programmable logic controller with basic functionality. This prototype, aided by open-source software, intends to recreate an industrial automation and control environment. The proposed methodology aims to explore the main concepts of basic electronics, code programming, communication between devices and data analysis. Finally, practical experiments and measurements will be made to validate and evaluate the proposed system, finishing with the discussion of the results obtained.

**Keywords:** Arduino. Microcontroller. Prototype. Hardware. Control.

## **LISTA DE ALGORITMOS**

Algoritmo 1 – Código de exemplo para configurar as saídas do multiplexador . . . . . 57

## LISTA DE ILUSTRAÇÕES

Figura 1 – Interface do Beremiz . . . . .	20
Figura 2 – Interface do OpenPLC Editor . . . . .	21
Figura 3 – Interface do servidor OpenPLC . . . . .	22
Figura 4 – Interface do HMIServer . . . . .	22
Figura 5 – Interface do HMIBuilder . . . . .	23
Figura 6 – Interface do LDMicro . . . . .	24
Figura 7 – Interface do OpenScada . . . . .	26
Figura 8 – Tela de ScadaBR . . . . .	27
Figura 9 – Interface do Proview . . . . .	28
Figura 10 – Interface do PVBrowser . . . . .	29
Figura 11 – Interface do PascalSCADA . . . . .	29
Figura 12 – Arduino Uno R3 . . . . .	31
Figura 13 – ARM STM32 STM32F103C8T6 . . . . .	32
Figura 14 – ESP32 . . . . .	32
Figura 15 – Teensy 4.0 . . . . .	33
Figura 16 – Estrutura básica de uma CLP . . . . .	36
Figura 17 – Princípio de funcionamento – Diagrama em blocos . . . . .	39
Figura 18 – Sinksource . . . . .	40
Figura 19 – Modelo mestre-escravo . . . . .	46
Figura 20 – Formato do Pacote RTU . . . . .	47
Figura 21 – Formato do Pacote ASCII . . . . .	47
Figura 22 – Arquitetura Microcontrolador ATMega328 . . . . .	49
Figura 23 – Comunicação UART . . . . .	51
Figura 24 – Comunicação SPI, mestre e escravos . . . . .	52
Figura 25 – Comunicação TWI, mestre e escravos . . . . .	52
Figura 26 – Conversor ATMega16U2 . . . . .	53
Figura 27 – Prototipo educacional EduArd . . . . .	54
Figura 28 – Remote 8-bit I/O expander for I2C-bus with interrupt . . . . .	55
Figura 29 – Remote 8-bit I/O expander for I2C-bus with interrupt . . . . .	56
Figura 30 – Single 8-Channel Analog Multiplexer/Demultiplexer . . . . .	57
Figura 31 – Circuito da fonte simétrica . . . . .	58
Figura 32 – Entrada 4 a 20mA simulada com Qucs . . . . .	59
Figura 33 – Filtro passa baixa de primeira ordem . . . . .	60
Figura 34 – Resposta do filtro passa baixa . . . . .	61
Figura 35 – Saída 4 a 20mA simulada com Qucs. . . . .	61
Figura 36 – Saída 0 a 10V simulada com Qucs . . . . .	62
Figura 37 – Entrada 0 a 10V simulada com Qucs . . . . .	63
Figura 38 – Entradas digitais 0-12V e 0-24V . . . . .	63
Figura 39 – Interface da saída digital relé . . . . .	64
Figura 40 – Interface saída Transistor 24V . . . . .	65
Figura 41 – BYU Arduino Temperature Control Lab . . . . .	65
Figura 42 – Circuito dos aquecedores . . . . .	66
Figura 43 – Interface configuração slave device OpenPLC . . . . .	67
Figura 44 – Interface Dashboard OpenPLC . . . . .	68

Figura 45 – Tela de criação do Data Source . . . . .	69
Figura 46 – Tela configuração do Data Source . . . . .	70
Figura 47 – Tela configuração do Data Points . . . . .	71
Figura 48 – Sistema de evaporação com 2 estágios . . . . .	72
Figura 49 – Diagrama de blocos do código Python . . . . .	78
Figura 50 – Diagrama da comunicação serial . . . . .	81
Figura 51 – Configuração parâmetros ScadaBR . . . . .	82
Figura 52 – Sinótico do ScadaBR . . . . .	82
Figura 53 – Bloco Ladder para conversão de temperatura . . . . .	83
Figura 54 – Sintonia PID . . . . .	84
Figura 55 – Conversão do valor PID . . . . .	85
Figura 56 – Conversão do valor PID . . . . .	85
Figura 57 – Sinoptico de ScadaBR . . . . .	86
Figura 58 – Vista da conexões. . . . .	122
Figura 59 – Vista 2D da placa. . . . .	122
Figura 60 – Vista 3D do protótipo. . . . .	123
Figura 61 – Conexões do microcontrolador. . . . .	123
Figura 62 – Fonte Linear. . . . .	124
Figura 63 – Reguladores da fonte. . . . .	124
Figura 64 – Entradas 0-10V. . . . .	125
Figura 65 – Saídas 0-10V. . . . .	125
Figura 66 – Entrada 4 a 20mA. . . . .	126
Figura 67 – Saídas 4 a 20mA. . . . .	126
Figura 68 – Saídas transistor e relé. . . . .	127
Gráfico 1 – Volumes dos evaporadores . . . . .	79
Gráfico 2 – Temperatura dos evaporadores . . . . .	80
Gráfico 3 – Concentração do licor negro . . . . .	80
Gráfico 4 – Temperatura no condensador . . . . .	81

## **LISTA DE TABELAS**

Tabela 1 – Funções especificadas no protocolo Modbus . . . . .	48
Tabela 2 – Requisitos da plataforma hardware . . . . .	54
Tabela 3 – Frequências do pino PWM . . . . .	60
Tabela 4 – Ferramentas open-source neste trabalho . . . . .	71
Tabela 5 – Receita dos sensores da planta . . . . .	77
Tabela 6 – Receita dos atuadores da planta . . . . .	77
Tabela 7 – Receita das perturbações da planta . . . . .	77
Tabela 8 – Setpoints do processo . . . . .	79

## **LISTA DE ABREVIATURAS, SIGLAS E ACRÔNIMOS**

### **SIGLAS**

CLP	Controlador Lógico Programável
IHM	Interface Homem Maquina
SCADA	Sistemas de Supervisão e Aquisição de Dados, do inglês Supervisory Control and Data Acquisition
TCP	Protocolo de Controle de Transmissão, do inglês Transmission Control Protocol
PWM	Modulação por largura de pulso, do inglês Pulse-Width Modulation
IoT	Internet das coisas, do inglês Internet of Things
E/S	Entrada/saída
CPU	Unidade central de processamento, do inglês Central Processing Unit
CIP	Protocolo Industrial Comun, do inglês Common INdustrial Protocol

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO . . . . .</b>	<b>15</b>
1.1	OBJETIVOS . . . . .	17
1.2	MOTIVAÇÃO . . . . .	17
1.3	ORGANIZAÇÃO . . . . .	17
<b>2</b>	<b>REVISÃO DA LITERATURA . . . . .</b>	<b>18</b>
2.1	LABORATÓRIOS ACADÊMICOS . . . . .	18
2.2	FERRAMENTA OPEN SOURCE . . . . .	19
2.3	SOFTWARE PARA CONTROLE DE PROCESSOS . . . . .	19
2.4	SOFTWARE PARA INTERFACE SCADA . . . . .	24
2.5	FERRAMENTA OPEN-HARDWARE . . . . .	28
2.6	TRABALHOS RELACIONADOS . . . . .	33
<b>3</b>	<b>TECNOLOGIAS . . . . .</b>	<b>35</b>
3.1	CONTROLADORES LÓGICOS PROGRAMÁVEIS . . . . .	35
3.1.1	Arquitetura e princípio de funcionamento . . . . .	36
3.1.2	Processo Scan . . . . .	38
3.1.3	Modulo E/S . . . . .	39
3.1.4	Programação dos CLPs . . . . .	40
3.1.4.1	Controladores PID . . . . .	41
3.2	COMUNICAÇÃO SERIAL EM CONTROLADORES . . . . .	42
3.2.1	DeviceNets . . . . .	42
3.2.2	ControlNet . . . . .	44
3.2.3	EtherNet/IP . . . . .	44
3.2.4	Foundation Fieldbus . . . . .	45
3.2.5	Profibus-DP . . . . .	45
3.3	PROTOCOLO MODBUS . . . . .	45
3.4	O MICROCONTROLADOR AVR . . . . .	47
3.4.1	ATMega328 . . . . .	48
3.4.2	Comunicação serial do microcontrolador . . . . .	49
3.4.3	UART . . . . .	50
3.4.4	Interface SPI . . . . .	51
3.4.5	Interface TWI . . . . .	52
3.4.6	Interface USB . . . . .	53
3.5	CONCLUSÕES DO CAPÍTULO 3 . . . . .	53
<b>4</b>	<b>ARQUITETURA DO PROTÓTIPO PROPOSTO . . . . .</b>	<b>54</b>
4.0.1	Expansão de portas TWI . . . . .	55
4.0.2	Multiplexador CD4051 . . . . .	56
4.0.3	Fonte simétrica . . . . .	58
4.0.4	Entrada e saídas E/S analógicas . . . . .	58
4.0.5	Entrada e saídas E/S digitais . . . . .	63
4.1	AQUECEDORES CONTROLADOS . . . . .	64
4.2	CONCLUSÕES DO CAPÍTULO 4 . . . . .	65

<b>5</b>	<b>FERRAMENTA SOFTWARE DO SISTEMA . . . . .</b>	<b>67</b>
5.1	OPENPLC . . . . .	67
5.2	SCADABR . . . . .	68
5.3	SOFTWARES AUXILIARES . . . . .	70
5.4	PLANTA DIDÁTICA . . . . .	72
5.5	O SOFTWARE DE GERENCIAMENTO DO SISTEMA . . . . .	76
<b>6</b>	<b>RESULTADOS E DISCUSSÃO . . . . .</b>	<b>78</b>
6.1	ENSAIO VIRTUAL . . . . .	78
6.2	ENSAIO HÍBRIDO . . . . .	81
6.3	ENSAIO COM AQUECEDOR . . . . .	83
6.4	CONCLUSÕES DO CAPÍTULO 6 . . . . .	86
<b>7</b>	<b>CONCLUSÕES E PERSPECTIVAS . . . . .</b>	<b>87</b>
	<b>REFERÊNCIAS . . . . .</b>	<b>88</b>
	<b>APÊNDICES</b>	<b>94</b>
	<b>APÊNDICE A – ARQUIVOS DE PROGRAMAÇÃO . . . . .</b>	<b>95</b>
	<b>APÊNDICE B – ARQUIVOS DO PROTÓTIPO HARDWARE . . . . .</b>	<b>122</b>

## 1 INTRODUÇÃO

Durante a formação acadêmica, em curso de engenharia, são apresentados métodos de aprendizagem genéricos e com um elevado grau de abstração que, com o avançar do percurso, se tornam cada vez mais específicos com a área de especialização. Neste percurso, a parte prática é importante porque permite consolidar conhecimentos teóricos e aplicá-los na resolução de problemas práticos. No entanto, a utilização de sistemas reais em ambiente acadêmico nem sempre é viável devido a vários fatores como a dimensão do espaço físico, características técnicas, requisitos elétricos, condições de segurança e o custo associado a este tipo de montagem. Assim e para ultrapassar estas condicionantes, recorre-se a sistemas didáticos, que são desenvolvidos de forma a terem um funcionamento semelhante aos sistemas reais (SILVA, 2018).

Neste contexto laboratórios didáticos podem ser entendidos como uma complementação acadêmica tendo como objetivo o desenvolvimento da capacidade do aluno de aplicar na prática os conhecimentos teóricos adquiridos, ampliando sua percepção da realidade e compreensão dos fenômenos físicos sobre os quais irá atuar ao longo de sua carreira profissional. Além do mais, essas práticas laboratoriais incentivam o aprendizado ativo, um aprendizado distribuído, em que se tem uma separação de tarefas e responsabilidades entre os alunos e um aprendizado de grupo, no qual o objetivo é o trabalho em equipe. Tendo foco na área de controle e automação, reconhecem-se alguns modelos de laboratórios acadêmicos: laboratório físico, virtual e híbrido, sendo o terceiro ambiente de interesse deste trabalho.

Possuindo em sua infraestrutura equipamento industrial e plantas ou miniaturas de plantas físicas interconectadas com computadores, os laboratórios reais são os que mais se aproximam da realidade da indústria, pois apresentam características muitas vezes desprezadas por simuladores virtuais, como, por exemplo, ruídos (perturbações), limites de controle e efeitos não lineares, além de possuir uma dinâmica. No entanto, essa infraestrutura, devido à complexidade e ao alto custo para aquisição e manutenção, e considerando também que algumas experiências precisam de insumos bastante caros, torna-se muito onerosa para as universidades.

Outro tipo de ambiente são os laboratórios virtuais, que consistem em simuladores de dispositivos físicos por meio de software. Todas as infraestruturas necessárias para o laboratório não são reais, são simuladas em computador. É importante notar nesse tipo de ambiente a flexibilidade que o aluno tem para fazer experiências sob diversas condições, sem medo de danificar equipamentos. Essa infraestrutura é vista como uma alternativa aos onerosos laboratórios físicos

(MARTINS, 2013). No entanto, nem sempre esses ambientes simulados possuem baixas despesas, às vezes o alto valor das licenças dos softwares inviabiliza sua aquisição. Enfim, a exposição excessiva a simulações pode resultar numa desconexão entre a realidade física e a virtual por parte do aluno.

Em Souza (2016) é explanado um terceiro tipo de ambiente: o laboratório híbrido, onde parte da simulação do processo industrial a ser estudado acontece com equipamentos industriais e parte do processo é simulado virtualmente, que também gera alto custo de hardware e software. Diante desse contexto de ambiente híbrido, esta pesquisa tem como objetivo implementar uma solução alternativa de baixo custo, que faça uso de ferramenta open-source. Pretende-se descrever a montagem, configuração e a implementação de um protótipo hardware acoplado com a plataforma Arduino Uno R3 e aliado com ferramenta software, que possa reproduzir experiências de laboratório para o ensino de controle e automação. Frizzarin (2016) destaca que o surgimento das plataformas de prototipagem para microcontroladores, tem permitido que os alunos pratiquem e desenvolvam soluções eletrônicas por conta própria.

As comunidades acadêmicas envolvidas com essas plataformas apoiam disponibilizando projetos hardware, código e tutoriais para que os alunos desenvolvam suas próprias soluções. É o caso da plataforma abordada neste trabalho, o Arduino. O Arduino ou outra plataforma micro controlada de prototipagem tem como objetivo controlar ou automatizar alguma tarefa. O termo automação pode ser definido como um processo onde são realizadas várias operações com o auxílio de diversos dispositivos eletrônicos e/ou mecânicos que controlam os seus próprios processos. Em um contexto mais específico, a automação industrial é a aplicação de técnicas, softwares e/ou equipamentos específicos numa determinada máquina ou processo industrial, com o objetivo de aumentar a sua eficiência e maximizar a produção com o menor consumo de energia e/ou matérias. Portanto neste cenário, entende-se que na automação, que além do hardware existe uma outra componente importante, o software. O software desenvolve tarefas de programação e controle e gerenciamento de hardware por meio de linguagens específicas. Linguagens de programação para a automação industrial foram padronizadas por meio da norma IEC 61131-3, publicada pela Comissão Eletrônica Internacional (IEC), que define um conjunto de normas e especificações técnicas com o objetivo de padronizar os autômatos programáveis.

## 1.1 OBJETIVOS

O objetivo deste trabalho é projetar um sistema hardware e software com a finalidade de estruturar um ambiente didático para simulações de controle de plantas virtuais de processos industriais ou pequenas plantas físicas, levando em conta alguns requisitos como programação simples e utilização de ferramenta open-source para desenvolvimento. O protótipo hardware auxiliado por software open-source de controle e gerenciamento será capaz de reproduzir algumas tarefas básicas dos controladores industriais. Com o intuito de ser uma ferramenta de aprendizado de baixo custo, esse hardware foi planejado para hospedar um módulo Arduino Uno R3 ou compatível no seu interior por meio de uma conexão de encaixe, sendo assim configurável, deixando à criatividade do aluno a tarefa de resolver problemas, como controle de modelos de plantas acadêmicas. Além disso, é de interesse nessa dissertação avaliar o potencial de alguns softwares open-source disponíveis atualmente para o desenvolvimento de projeto de automação acadêmicos.

## 1.2 MOTIVAÇÃO

Em função do alto custo do equipamento industriais e software proprietários e do grande interesse da comunidade acadêmica em projetar alternativas de baixo custo aliadas ao uso conjunto de hardware e software open-source, julgou-se importante participar desse movimento acadêmico desenvolvendo uma ferramenta capaz de auxiliar os alunos no desenvolvimento prático dentro dos laboratórios acadêmicos de cursos de controle e automação.

## 1.3 ORGANIZAÇÃO

Este trabalho está organizado em outros seis capítulos além da introdução, na qual é apresentada toda a contextualização do tema. No capítulo 2 são apresentados os conceitos teóricos básicos necessários ao desenvolvimento do sistema de controle e algumas ferramentas de hardware e software disponíveis atualmente. No capítulo 3 aborda uma introdução sobre Controladores lógicos programáveis e sua arquitetura. Os capítulos 4 e 5 apresentam a metodologia utilizada para desenvolvimento do hardware e utilização da ferramenta open-source escolhida. Posteriormente, no capítulo 6 são apresentados os ensaios e analisados e discutidos os resultados. Por fim, no capítulo 7 são apresentadas as conclusões.

## 2 REVISÃO DA LITERATURA

Neste capítulo, relata-se a fundamentação teórica necessária ao desenvolvimento do protótipo, a começar pelos controladores lógicos programáveis, que estão na base da automação, elucidando a tecnologia envolvida, as entradas e saídas analógicas e digitais, a alimentação, os protocolos de comunicação e os softwares.

### 2.1 LABORATÓRIOS ACADÊMICOS

Atividades desenvolvidas em laboratórios didáticos podem ser entendidas como uma complementação às atividades acadêmicas tendo como objetivo o desenvolvimento da capacidade do aluno de aplicar na prática os conhecimentos teóricos adquiridos, ampliando sua percepção da realidade e compreensão dos fenômenos físicos sobre os quais irá atuar ao longo de sua carreira profissional. Além disso, essas práticas laboratoriais incentivam o aprendizado ativo, um aprendizado distribuído, em que se tem uma separação de tarefas e responsabilidades entre os alunos e um aprendizado de grupo, no qual o objetivo é o trabalho em equipe. Considerando o ensino de engenharia, em especial a área de controle e automação, reconhecem-se alguns modelos de laboratórios acadêmicos: laboratório físico, virtual e híbrido, sendo o terceiro ambiente de interesse nesta pesquisa.

Possuindo em sua infraestrutura equipamentos industriais e plantas ou miniaturas de plantas físicas interconectadas com computadores, os laboratórios reais são o que mais se aproximam da realidade da indústria, pois apresentam características muitas vezes não consideradas por simuladores virtuais, como, por exemplo, ruídos (perturbações), limites de controle e efeitos não lineares, além das plantas possuir uma dinâmica. No entanto, essa infraestrutura, devido à complexidade de instalação e ao alto custo para aquisição e manutenção, e considerando também que algumas experiências precisam de insumos bastante caros, torna-se muito onerosa para as universidades.

Outro tipo de ambiente são os laboratórios virtuais, que consistem em simulações de dispositivos físicos por meio de software. Todas as infraestruturas necessárias para o laboratório não são reais, são simuladas em computador. É importante notar nesse tipo de ambiente a flexibilidade que o aluno tem para fazer experiências sob diversas condições, sem medo de danificar equipamentos. Essa infraestrutura é vista como uma alternativa aos onerosos laboratórios físicos

(MARTINS, 2013). No entanto, nem sempre esses ambientes simulados são de baixo custo, às vezes o alto valor das licenças dos softwares inviabiliza sua instalação. Enfim, a exposição excessiva a simulações pode resultar numa desconexão entre a realidade física e a virtual por parte do aluno.

Em Souza (2016) é explanado um terceiro tipo de ambiente: laboratório híbrido. Nesse tipo de laboratório, o qual é objeto de estudo deste trabalho, parte da simulação do processo industrial a ser estudado acontece usando equipamentos reais como nos laboratórios físicos e parte do processo é simulado em ambiente virtual. Enfim, em Silva (2018) é explanada outro tipo de subdivisão de laboratórios acadêmicos: Laboratório local e Laboratório remoto.

Em quanto o laboratório local pode ser comparado ao laboratório físico, já conceituado, o laboratório remoto consiste num ambiente em que uma ou mais experiências podem ser realizadas sem intervenção direta, sendo comandada remotamente e os resultados obtidos por medições e/ou visualização de imagens em tempo real da experiência. O sistema proposto tem funcionalidades para ser usado remotamente ou localmente, neste trabalho serão abordadas só as configurações para trabalho local.

## 2.2 FERRAMENTA OPEN SOURCE

Open source é um termo em inglês que significa código aberto. Isso diz respeito ao código-fonte de um software, que pode ser adaptado para diferentes fins. O termo foi criado pela OSI (Open Source Initiative) que o utiliza sob um ponto de vista essencialmente técnico. Na grande maioria dos casos, essas ferramentas são compartilhadas online pelos desenvolvedores, podendo ter acesso a elas qualquer pessoa, sem restrições. São exemplos deste tipo de software: Linux, Eclipse, Mysql, Apache entre outros.

## 2.3 SOFTWARE PARA CONTROLE DE PROCESSOS

Software para controle de processos tem, assim como o nome sugere, a função de controlar e monitorar os parâmetros requeridos para as diferentes atividades automatizadas. Nessa pesquisa foram encontrados os seguintes softwares open-source: Beremiz (BEREMIZ, 2017), MBLogic (MBLOGIC, 2010) e LDMicro (WESTHUES, 2016). Estes software são resumidos a seguir.

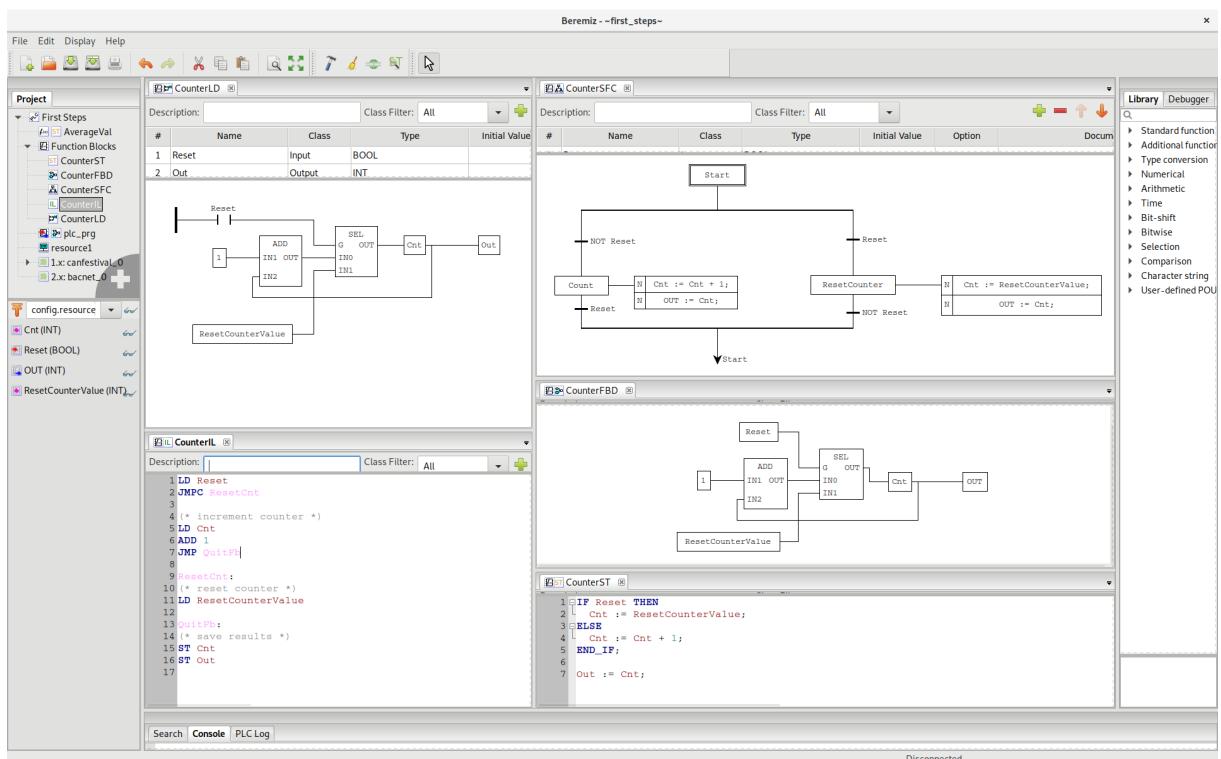
- O Beremiz é um IDE de código fonte aberto, multiplataforma, para o desenvolvimento

de programas de automação em conformidade com o disposto na norma IEC 61131-3, disponibilizada ao público livremente sob a licença de software GNU GPLv2.1 ou posterior. Esse software permite a substituição do CLP por qualquer processador. Foi desenvolvido de forma modular e baseia-se nos subprojetos seguintes:

- PLCOpen Editor - Editor dos programas IEC 61131-3
- MATIEC - Compilador IEC 61131-3 -> ANSI-C
- CanFestival - CANOpen Framework para interface com I/O físicos
- SVGUI - Ferramenta para integração de HMIs

O compilador MATIEC converte o projeto desenvolvido nas linguagens de programação definidas na norma IEC61131-3 em código C equivalente. A figura 1 mostra a interface integrada do programa, o PLCOpen Editor, o qual permite criar códigos escolhendo uma linguagem entre aquelas permitidas pela norma IEC 61131-3.

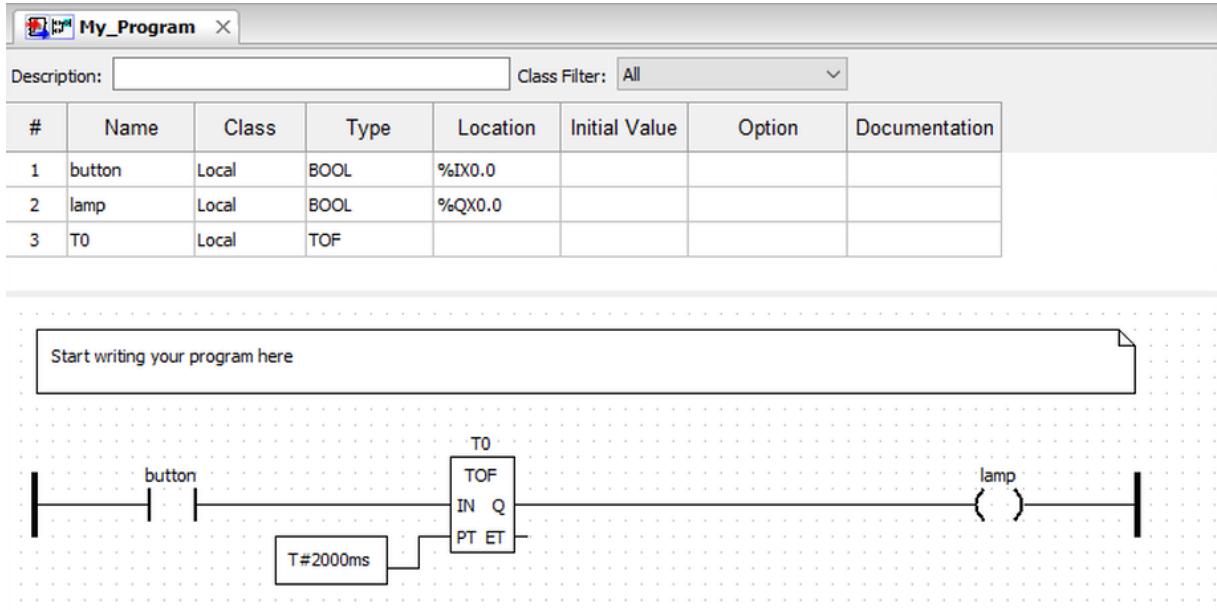
**Figura 1 – Interface do Beremiz**



**Fonte: autoria própria**

- O openPLC também é um softPLC open-source, criado de acordo com o standard IEC 61131-3, o software pode ser executado em várias plataformas como computadores com Windows ou Linux e dispositivos como Raspberry PI.

**Figura 2 – Interface do OpenPLC Editor**

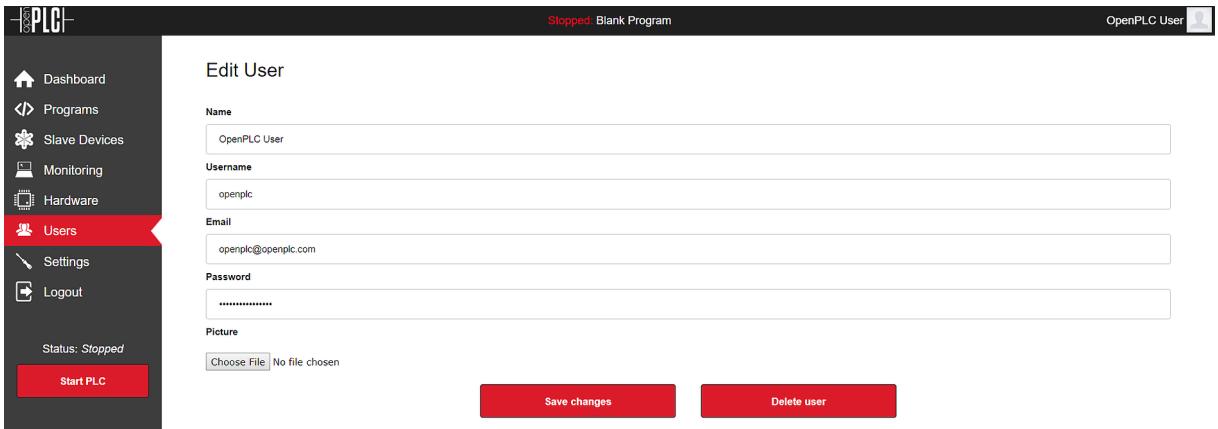


**Fonte: Autoria própria**

Por meio do editor OpenPLC Editor, figura 2, é possível desenvolver algoritmos de programação em todas as cinco linguagens definidas pela norma IEC 61131-3. Atualmente o programa oferece suporte às seguintes plataformas: Arduino, Raspberry Pi, UniPi Industrial Platform, dispositivos escravos Modbus, ESP8266 e PiXtend. Ainda funciona como CLP virtual nos sistemas operacionais Linux e Windows, permitindo a integração com outros softwares através de interfaces adequadas. Atualmente o programa está na versão 3.0, que foi a utilizada neste trabalho (ALVES *et al.*, 2014). Este é um software licenciado na forma GPL (acrônimo inglês para General Public License) para uso comercial, modificação, distribuição, uso privado e uso de patente de colaboradores. O OpenPLC pode ser acessado através da rede industrial Modbus/TCP. Isto significa que é possível ler e escrever dados no CLP virtual através de dispositivos Modbus, bem como pode ser integrado a Sistemas de Supervisão e Aquisição de Dados (SCADA). O servidor do OpenPLC, figura 5, é acessado através de um navegador web. Através da tela inicial, pode-se interagir com o CLP virtual inicializando ou paralisando o seu funcionamento, visualizando logs e carregando arquivos de configuração do CLP.

- O projeto MBLogic pretende ser uma plataforma completa de automação que segue a norma IEC 61131-3 (PLCOPEN, 1993). Essa plataforma contém um conjunto de pacotes de software utilizados para a comunicação e controle industrial de processos. Este conjunto de pacotes pode ser utilizado individualmente ou em conjunto, conforme o desejado pelo

**Figura 3 – Interface do servidor OpenPLC**

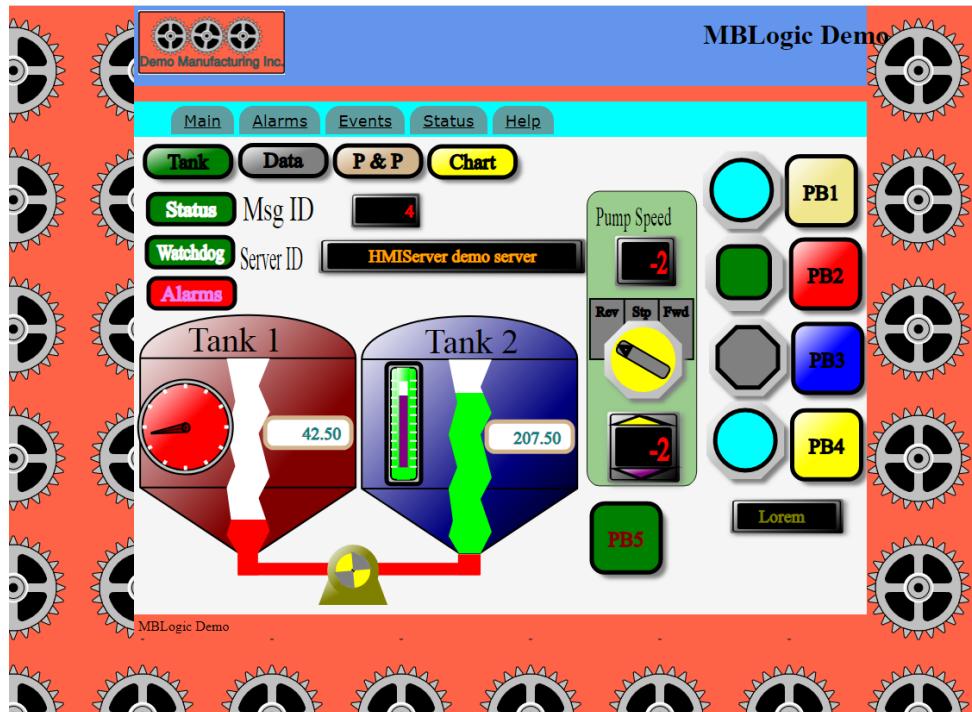


**Fonte:** Autoria própria

utilizador. Os pacotes contidos nos projetos são os seguintes:

- MBLogic Soft Logic: este pacote é uma plataforma completa e independente para o controle de plataformas industriais. Incorpora um controlador virtual com IHM baseada em web, campo para conexão de entradas e saídas e ajuda on-line.
- HMI Server: pacote independente que fornece ao utilizador um sistema IHM para a monitorização e controle de dispositivos industriais como o CLP, por exemplo.

**Figura 4 – Interface do HMI Server**



**Fonte:** autoria Propriá

- HMIBuilder: pacote que proporciona a criação automática de páginas web IHM sem a necessidade de programação. As telas web são montados usando a ferramenta gráficas fornecidas pelo software, utilizando o método arrasta e solta, e configurados através de menus onde se selecionam as opções desejadas.

**Figura 5 – Interface do HMIBuilder**

## Ladder Editor Demo

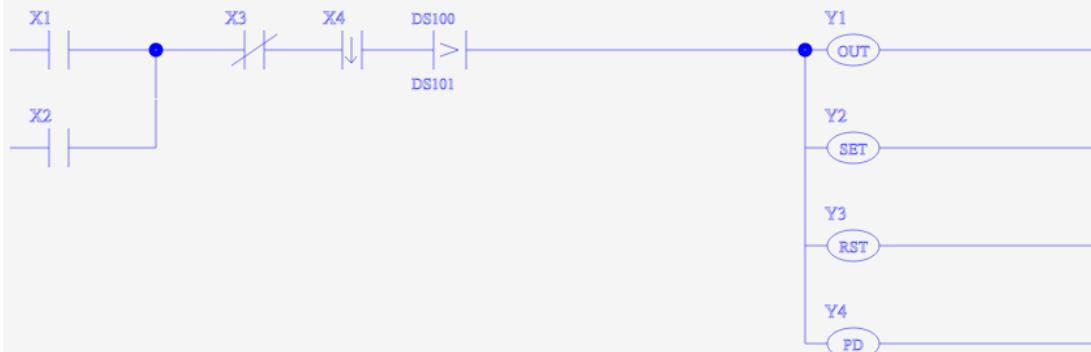
### SBR LadderDemo

Demonstrate different ladder instructions. This subroutine doesn't actually do anything useful. Click on the subroutine title bar above in order to edit this comment.

#### Rung: 1

[Insert Rung](#)

Click on the rung title bar to edit this rung.



#### Rung: 2

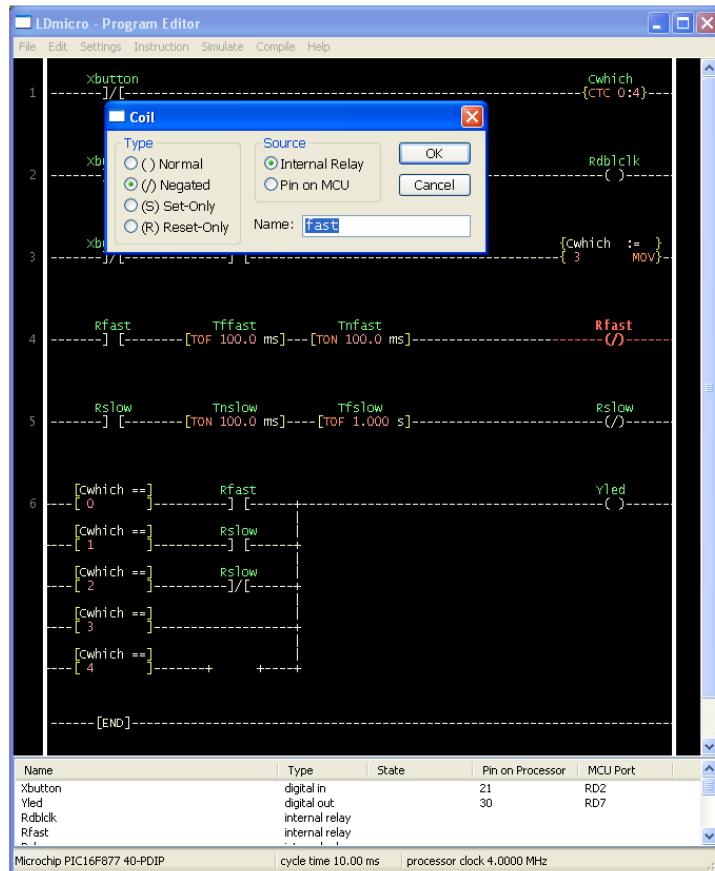
[Insert Rung](#)

**Fonte: Autoria própria**

- MBLogic Tools é o pacote que fornece ferramentas para teste e solução de problemas das aplicações. Para além destas funcionalidades, fornece também servidores que permitem que os clientes comuniquem-se uns com os outros. Algumas das ferramentas existentes neste pacote são: MBAsyncServer (servidor proxy Modbus/TCP), MBProbe (ferramenta cliente web based para a leitura e escrita de um servidor Modbus/TCP para teste e troubleshooting, manualmente), e MBPoll (linha de comandos para testar servidores Modbus TCP/IP).
- MBLogic Libraries é um pacote para o desenvolvimento de aplicações em Python. Contém um driver de cliente Modbus/TCP e um sistema soft logic que podem ser incluídos como bibliotecas nos programas a serem desenvolvidos pelo utilizador. Estas bibliotecas são utilizadas tipicamente para a produção de sistemas de teste.

- O ultimo projeto open-source analizado é o LDMicron (figura 6). No site do desenvolvedor não é especificado se atende a norma IEC 61131-3, o programa é descrito como um compilador que permite gerar um código nativo para microcontroladores PIC16 e AVR a partir de um diagrama em linguagem ladder.

**Figura 6 – Interface do LDMicro**



**Fonte: Autoria própria**

## 2.4 SOFTWARE PARA INTERFACE SCADA

Os sistemas supervisórios são sistemas digitais que têm a finalidade de supervisionar e monitorar as variáveis de processo dos equipamentos ou plataformas industriais a eles conectadas. Os dados coletados podem ser guardados em bancos de dados locais ou remotos para fins de registro histórico (MORAES; CASTRUCCI, 2007). De acordo com MCA Sistemas (2013), um sistema SCADA deve ser capaz de:

- Fazer a aquisição de variáveis de diferentes plataformas externas através de vários protocolos de comunicação, como o Ethernet e o Modbus, por exemplo;

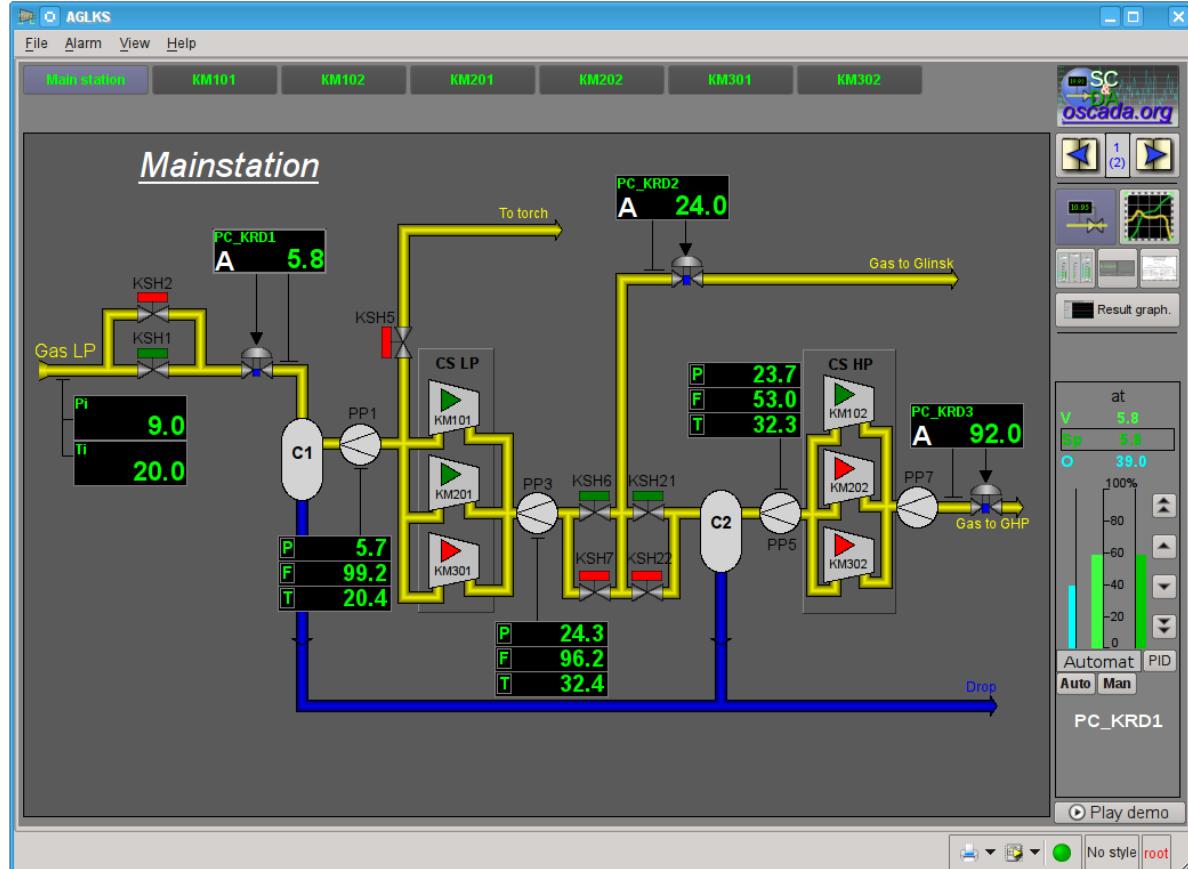
- Apresentar dados em tempo real para a monitorização do processo;
- Fornecer interfaces gráficas que apresentem o estado atual do processo, permitindo assim uma fácil visualização e análise das variáveis do processo e dos alarmes. Permite também interação com determinadas variáveis do processo;
- Gerar relatórios e planejamento de tarefas;
- Apresentar gráficos de histórico do comportamento das variáveis, que deve ficar guardado, para permitir ao utilizador o estudo do mesmo, definindo posteriormente alguma ação a tomar de maneira a melhorar a produtividade;
- Disparar alarmes que podem ser geridos pelo usuário. Os alarmes devem ficar registados em um arquivo de registro de eventos, para o usuário poder aceder ao seu histórico. É ainda possível haver uma hierarquização de alarmes e distribui-los por usuário;

Atualmente a elaboração de um sistema SCADA não requer qualquer conhecimento de programação uma vez que toda a sua configuração é feita através de ambientes gráficos. Relativamente às ferramentas existentes para o desenvolvimento de interfaces SCADA, encontraram-se inúmeras soluções open-source, e algumas dessas serão apresentadas em seguida.

- O OpenSCADA é uma implementação open-source de um sistema SCADA com base em Linux, figura 7 (SAVOCHENKO, 2018). A arquitetura deste sistema consiste numa organização em subsistemas, cada um com as suas funções:
  - Segurança: contém a lista de usuário e grupos de utilizadores. Verifica quem é que pode aceder a cada elemento do sistema;
  - Banco de dados: acesso à base de dados;
  - Comunicação: providencia a comunicação com o ambiente através de diferentes interfaces de comunicação;
  - Protocolo de Comunicação: está ligado ao subsistema de transporte e dá suporte a vários relatórios de troca com sistemas externos;
  - DAQ – fornece dados das fontes exteriores como controladores, medidores, etc;
  - Arquivos: contém arquivos de dois tipos: arquivos de mensagens e arquivos de valores. O modo de troca de arquivos é definido pelo algoritmo incorporado neste módulo.

- User Interface – contém as funções de interface do usuário.

**Figura 7 – Interface do OpenScada**



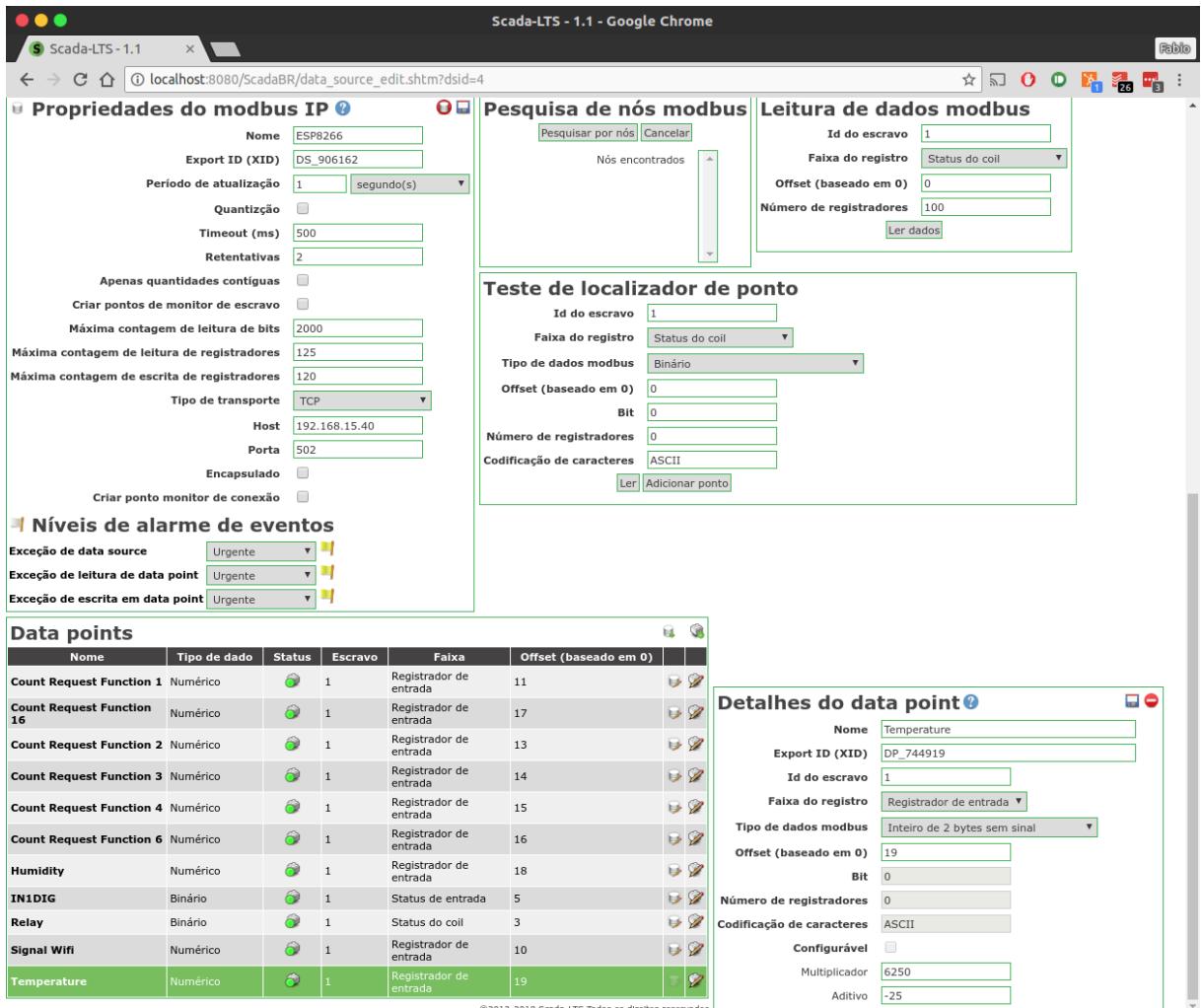
**Fonte:** Autoria própria

- O ScadaBR mostrado na figura 8 é uma iniciativa nacional no desenvolvimento de um software SCADA no modelo de software livre. Ele se baseia no projeto Mango, que é um sistema SCADA open-source canadense, o qual possui todas as funcionalidades do Mango mais as características adicionadas no projeto brasileiro. Além de aberto, já é distribuído e funciona por meio de servidores web, possibilitando de maneira simples se integrar com outros sistemas e controlar plantas remotamente (MCA Sistemas, 2013). O software é multiplataforma, baseado em Java, ou seja, computadores rodando Windows, Linux e outros sistemas operacionais podem executá-lo a partir de um servidor de aplicações.

As principais características são:

- Comunicação com os dispositivos (configuração dos “datasources” e “data points”);
- Gerenciamento de Alarmes;
- Históricos e Banco de Dados;

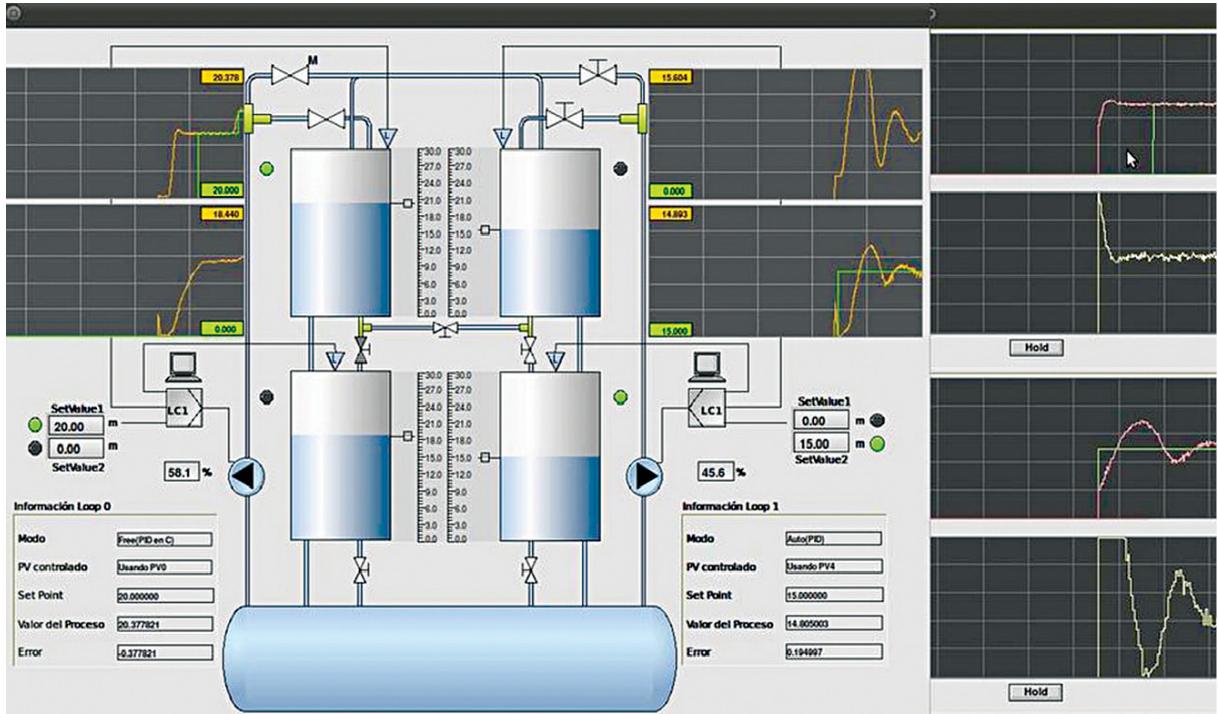
**Figura 8 – Tela de ScadaBR**



**Fonte: (NASCIMENTO; LUCENA, 2003)**

- Lógicas de programação interna ou controle;
- Interface gráfica;
- Geração de Relatórios;
- Registro de eventos;
- Linguagem de programação;
- O Proview Open Source Process Control é um projeto desenvolvido na Suécia pela Mandator e pela SSAB Oxelösund como um sistema de controle de processos baseado em computadores padrão. O sistema tornou-se num sistema completo, integrado e uma solução barata que corre em PCs standard que tenham Linux como sistema operacional (PROVIEW, 2020). O software Proview é open-source e a licença é GNU/GPL, figura 9.

**Figura 9 – Interface do Proview**



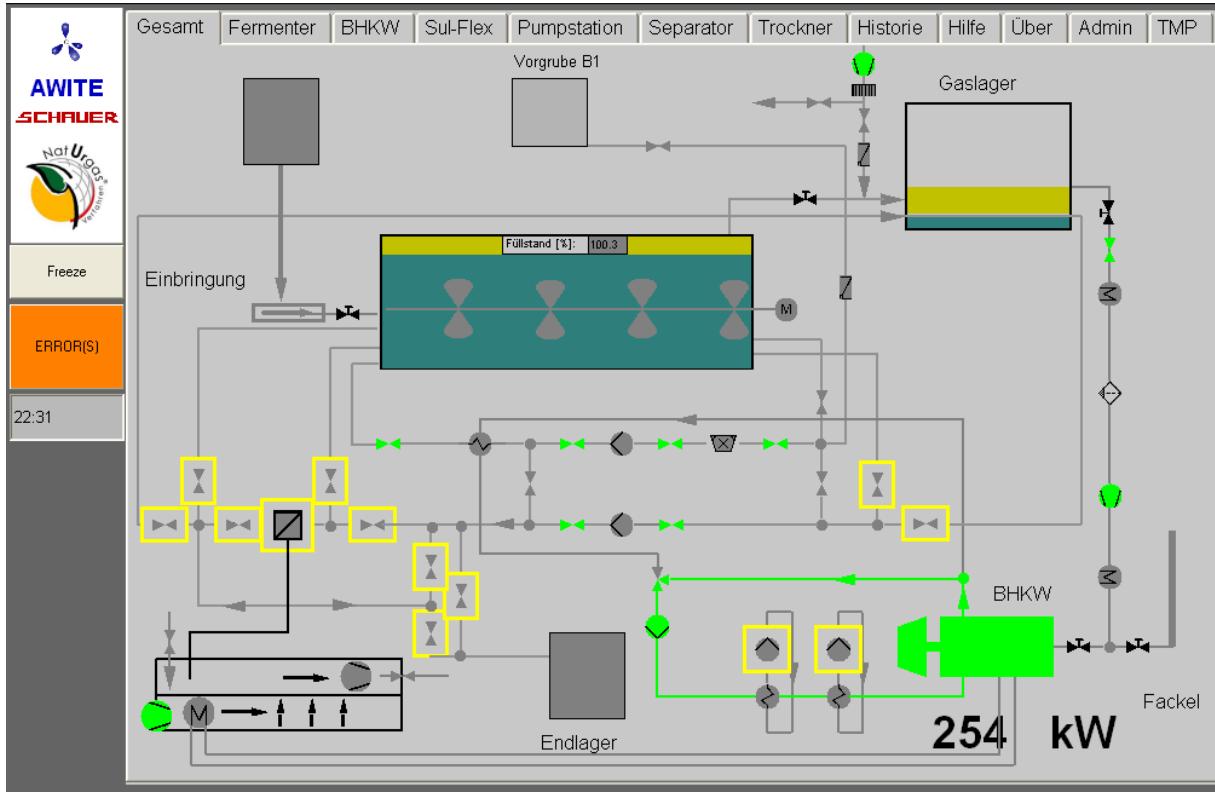
**Fonte:** Autoria própria

- O PVBrowser é uma aplicativo alemão que permite ao usuário a criação de aplicações SCADA. Providencia ao usuário um navegador especializado para o cliente (pvbrowser) e um IDE (pvdevelop) para a criação de servidores (pvserver) que implementam a visualização a ser apresentada ao cliente (MULLER, 2020). Funciona com o princípio cliente/servidor e está sob a licença LGPL, figura 10.
- PascalSCADA é um framework para Delphi/Lazarus cuja principal característica é o rápido desenvolvimento de aplicações SCADA (PASCALSCADA, 2018). Este software funciona em Linux, Windows (ambos em 32 e 64 bits) e em FreeBSD (32 bits), figura 11. Com o PascalSCADA é possível comunicar com CLP, criar interfaces, registrar variáveis e alarmes do processo e controlar utilizadores da aplicação.

## 2.5 FERRAMENTA OPEN-HARDWARE

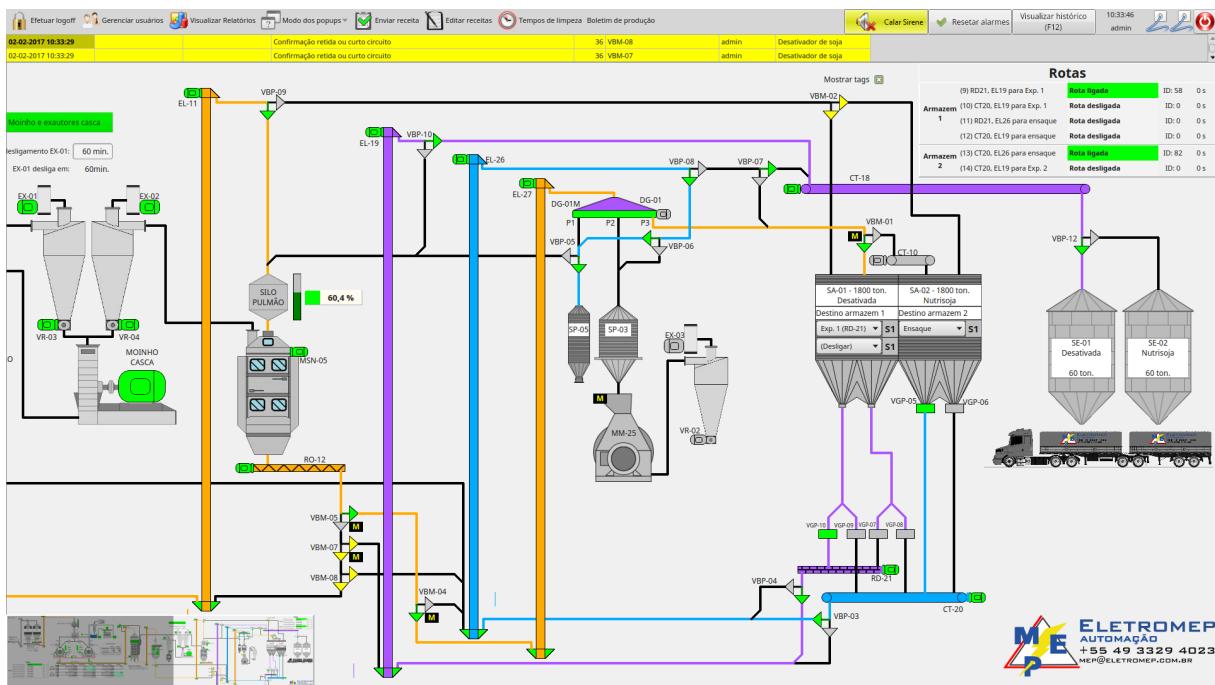
Projetos de hardwares, como o utilizado nesse trabalho, não são cobertos por licenças de copyright, mas sim por patentes; no entanto o fato de o hardware não ser protegido por copyright não significa que é impossível licenciar o projeto com licenças abertas. De acordo com a OSHWA (2012), o hardware se diferencia do software no sentido de que recursos físicos devem sempre

**Figura 10 – Interface do PVBrowser**



Fonte: Autoria própria

**Figura 11 – Interface do PascalSCADA**



Fonte: Autoria própria

ser empregados na produção de bens físicos. Desse modo, pessoas ou empresas produzindo itens (“produtos”) sob uma licença OSHWA têm uma obrigação de não impor que esses produtos

sejam fabricados, vendidos, garantidos ou sancionados de qualquer modo pelo desenvolvedor original e também de não fazer uso de registros comerciais pertencentes a esse desenvolvedor. O uso de plataformas micro-controladas nas escolas técnicas e nos cursos de Engenharia se iniciou na década de 1980 com circuitos complexos e a necessidade de conhecimento técnico avançado, limitando, assim, a difusão desse conhecimento em outras áreas da ciência. Atualmente, projetos open-hardware estão se impondo no meio acadêmico como uma alternativa de baixo custo para laboratórios universitários (FOULIS, 2018). Algumas das mais conhecidas serão apresentadas em seguidas.

- Arduino: é baseado numa simples placa microcontroladora e num ambiente de desenvolvimento de código aberto que se fundamenta tanto em hardware quanto em software de fácil utilização (HUGHES, 2016).

Segundo o fabricante (MCROBERTS, 2018), o Arduino Uno, que é a plataforma mais conhecida, é uma placa de microcontrolador baseado no ATmega328 que permite a ligação com um computador (figura 12); além disso, contém outro terminal que permite a conexão com outros dispositivos externos. Dispõe de 14 pinos digitais de entrada/saída, dos quais seis podem ser utilizados como saídas PWM, e também possui seis entradas analógicas, uma conexão USB, um conector de energia, um cabeçalho ICSP e um botão de reiniciar. O hardware e o software do Arduino são ambos de fonte aberta, o que significa que o código, os esquemas, o projeto, etc. podem ser utilizados livremente por qualquer pessoa que quer desenvolver projetos nas áreas de eletrônica e automação (MONK, 2017). Pode-se salientar como o microcontrolador Arduino se tornou uma das principais ferramentas no processo de ensino devido também à enorme quantidade de Shields, que são módulos ou protótipos hardware de encaixe ou de fácil conexão com a plataforma Arduino para estender as funcionalidades (GALADIMA, 2014; GARRIGOS; MARROQUI, 2017). Como exemplos, em Monteiro (2016) o Arduino é usado junto a linguagem Processing no ensino de Física. Em Pereira (2018) são investigadas as possibilidades de ensino e aprendizagem usando a plataforma Arduino e a metodologia PBL (problem based learning) para o ensino de Ciências, em particular para os professores em formação, tendo como temática o conceito energia. Em Rubim (2014), o Arduino é utilizado no ensino para aprendizagem sobre o tema luz e cor. Em Guedes (), o Arduino é proposto como ferramenta para motivar e expandir o conhecimento de estudantes iniciantes sobre Python. Em Nunes (2018) é desenvolvido um kit experimental produzido com materiais de baixo

custo associados a uma plataforma de aquisição automática e interpretação de dados Arduino. Em Cardoso (2016), o Arduino é utilizado para estudo de técnicas e ferramentas utilizadas no desenvolvimento de trajetórias para robôs móveis. Em Mello (2017), o Arduino é utilizado para criar um ambiente de baixo custo para controle de anomalias no ambiente.

**Figura 12 – Arduino Uno R3**

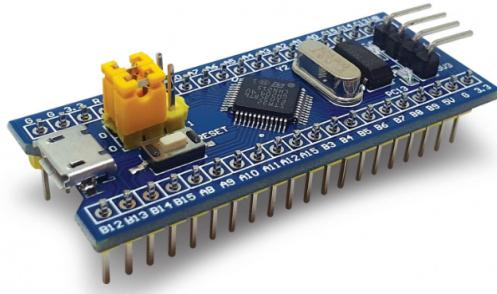


**Fonte:** (ARDUINO, 2018)

Todas as placas Arduino já possuem programação de Bootloader (programa de gerenciamento do processo de inicialização do microcontrolador) de fábrica, uma rotina de inicialização que facilita o envio de programas para a memória interna do chip do microcontrolador. Tal envio é feito, nas placas atuais, por meio de conexão no computador via porta USB (ARDUINO, 2018). Os programas para Arduino podem ser escritos em qualquer linguagem que possui compiladores que convertem o código em linguagem de máquina para o processador da placa em uso.

- STM32F103C8T6: Trata-se de uma placa de desenvolvimento de alto desempenho que incorpora um núcleo RISC de 32 bits ARM Cortex M3 e opera a uma frequência de 72MHz (STMicroelectronics, 2018). Possui um regulador de tensão de 3.3v de 100ma, um botão de reinicialização e dois Jumpers (conector de 2 pontos) para ativar ou desativar o modo de programação do Bootloader. Colocando um resistor de 1k5 ou 1k8 Ohm diretamente entre os pinos A12 e 3.3V pode-se utilizar a interface USB nativa que o chip possui. Para gravar códigos pode-se utilizar a IDE do Arduino.
- ESP32: O módulo ESP32 é um módulo de alta performance para aplicações envolvendo

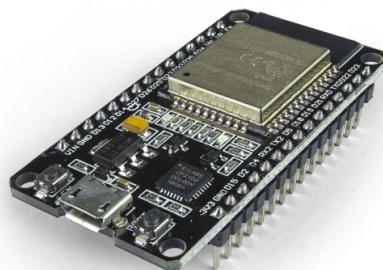
**Figura 13 – ARM STM32 STM32F103C8T6**



**Fonte:** (STMicroelectronics, 2018)

wifi, contando com um baixíssimo consumo de energia (ESPRESSIF, 2018). Com 4 MB de memória flash, o ESP32 permite criar variadas aplicações para projetos de IoT, acesso remoto, webservers e dataloggers, entre outros, figura 14. A placa ESP32 possui uma antena embutida, uma interface usb-serial e um regulador de tensão 3.3V. A programação pode ser feita em LUA, a qual é uma linguagem de programação interpretada, projetada por Tecgraf da PUC-Rio em 1993, ou usando a IDE do Arduino através de um cabo micro-usb (TECGRAF PUC-RIO, 2021).

**Figura 14 – ESP32**

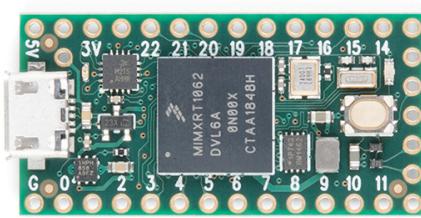


**Fonte:** (ESPRESSIF, 2018)

- O Teensy 4.0 é: Trata-se de uma placa de prototipação rápida e é suportada pelo ambiente Arduino (PJRC, 2019). O microcontrolador que comanda a placa Teensy 4.0 é um Cortex-M7 capaz de operar até a 600 MHz, e apresenta uma capacidade para executar duas instruções no mesmo ciclo, além de diversas características que permitem o desenvolvimento de algoritmos que demandam criptografia ou cálculos complexos, como o

acelerador criptográfico e gerador de números aleatórios em chip e FPU(Unidade de ponto flutuante), o qual é um hardware dedicado a executar operações matemáticas de dados representados em ponto flutuante, que suporta Double de 64 bits e Float de 32 bits.

**Figura 15 – Teensy 4.0**



**Fonte:** (PJRC, 2019)

## 2.6 TRABALHOS RELACIONADOS

Para acompanhar o desenvolvimento massivo e crescente das áreas de engenharia elétrica e informática, o sistema educacional deve ser capaz de reagir às tecnologias atualmente utilizadas e disponíveis no mercado e incluí-las no processo educacional. Atender às novas tendências torna-se crucial ao produzir pessoas com visão, experiência e visão ampla (HURTUK *et al.*, 2017). Hoje em dia, as tecnologias baseadas em Arduino estão ganhando grande popularidade entre as comunidades interessadas nas áreas de soluções eletrônicas e de hardware, por sua disponibilidade no mercado, relativamente fácil de usar e, por último, mas não menos importante, por seu preço barato e grande acessibilidade a várias informações, componentes e tutoriais já feitos. Portanto, torna-se cada vez mais crucial incluir o Arduino no processo educacional dentro das disciplinas ministradas nas escolas, de engenharia elétrica e informática.

Em Foulis (2018) explana-se o desenvolvimento de um kit de laboratório projetado no departamento de Engenharia de Automação, ATEI de Thessaloniki, Grécia, para atender às necessidades de ensino nos cursos de controle e automação de graduação e pós-graduação. O autor enfatiza como a falta de recursos financeiros e de pessoal suficiente para a compra e manutenção de equipamentos de laboratório, principalmente devido à recente crise econômica, levou a instituição a uma reforma drástica dos cursos práticos de laboratório, que agora se baseiam em kits de baixo custo para levar para casa, equipados com base em Arduino e sensores

compatíveis. O autor afirma não ter dúvida que esses equipamentos despertam o interesse do aluno pela teoria e prática de controle, permitindo que ele possa se aprimorar nos estudos também em seu próprio lar.

Em Pacheco *et al.* (2019) explica-se como a popularidade do Arduino tem crescido nos últimos anos, principalmente como parte da Internet das Coisas, que está produzindo um impacto relevante em diversos setores da economia (indústria, transportes, energia, agricultura, automação residencial, etc.). Várias políticas nacionais e europeias foram definidas para treinar as empresas da UE para a adoção e difusão das tecnologias IoT. O autor descreve o desenvolvimento de um laboratório remoto de Arduino para dar suporte a ambientes de experimentação de aprendizagem de IoT on-line, considerados importantes para fornecer programas de educação on-line de qualidade em IoT.

Em Li (2018) é proposto uma configuração de laboratório de sistema de controle. A ferramenta Arduino é utilizada no curso de controle e automação como um controlador digital. O autor afirma que este projeto de laboratório de sistema de controle pode reforçar a experiência prática do aluno para o ensino de graduação em engenharia. Neste artigo, o autor fornece dois exemplos demonstrativos para experimentos práticos, um é o controle PID e o outro é o controle fuzzy. A partir dos resultados experimentais, afirma que o aluno pode aprender os assuntos relacionados, alcançando objetivo das disciplinas estudadas.

### 3 TECNOLOGIAS

#### 3.1 CONTROLADORES LÓGICOS PROGRAMÁVEIS

O controlador lógico programável, ou simplesmente CLP, como normalmente é chamado no meio industrial, é um computador especializado baseado em um microprocessador que foi desenvolvido para substituir circuitos de relés que integravam o antigo painel industrial. Nesses painéis, para efetuar uma modificação da lógica dos comandos, por qualquer motivo era necessário um rearranjo na montagem, via de regra cansativo, demorado e dispendioso, modificação que, às vezes, implicava uma reforma total dos armários elétricos (MORAES; CASTRUCCI, 2007).

Com o controlador programável basta modificar o programa, mantendo o hardware para automatizar processos industriais, como leituras de sensores, acionamento de válvulas, acionamento de motores, dentre outros dispositivos não inteligentes. Um controlador lógico programável automatiza uma grande quantidade de ações com precisão, confiabilidade, rapidez e pouco investimento. Informações de entrada são analisadas, decisões são tomadas e comandos são transmitidos (FRANCHI; CAMARGO, 2008). Para aplicação industrial de um CLP de forma ampla, é necessário um controlador com as seguintes características:

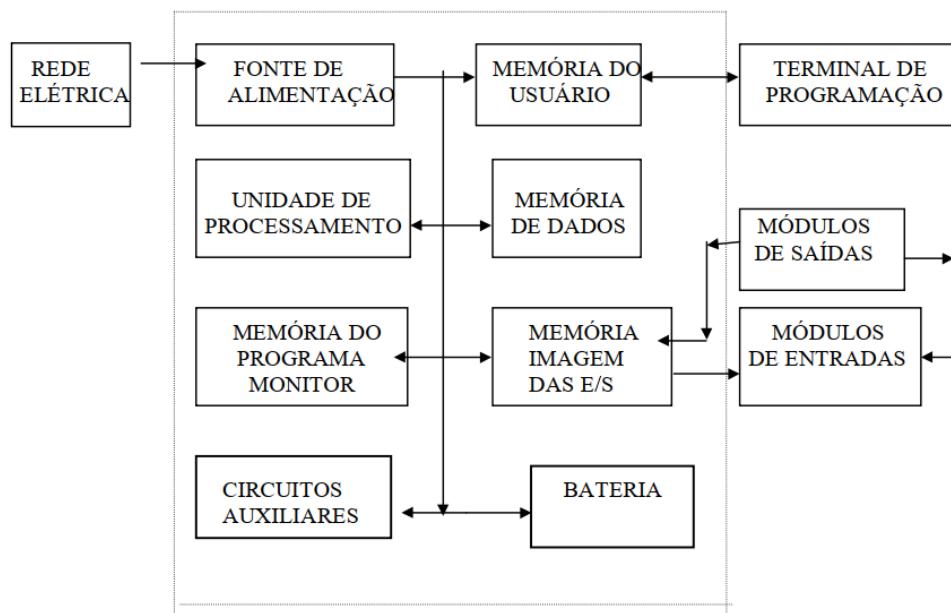
- Facilidade de programação e reprogramação, preferivelmente na planta, linguagem de alto nível e amigável com relação ao operador;
- Possibilidade de manutenção e reparo, com toda a fiação resumida em conjuntos de entradas e saídas;
- Confiabilidade operacional, para que possa ser utilizado em um ambiente industrial;
- Memória programável e possibilidade de expansão;
- Possibilidade de integração dos dados de processo do CLP em bancos de dados gerenciais, para tornar disponíveis informações sobre o chão de fábrica para os departamentos envolvidos com o planejamento da produção;
- Por meio de comunicação em rede permitir coleta de dados e de troca de dados;

- Informar ao operador em caso de defeito, por meio de sinalizadores visuais no CLP, a parte do sistema que está defeituosa.

### 3.1.1 Arquitetura e princípio de funcionamento

O CLP é um equipamento que pode ser programado para executar instruções que controlam motores, dispositivos, máquinas e operações de processos; tais instruções são implementação de funções específicas, como lógica de controle, sequenciamento, controle de tempo, operações aritméticas, controle estatístico, controle de malha, transmissão de dados etc. (FRANCHI; CAMARGO, 2008). Em instalações industriais, os CLPs devem suportar altas temperaturas, umidade, ruídos elétricos, poluição atmosférica, entre outros fatores, portanto, devem ser projetados e construídos para operar em ambientes agressivos. Como mostrado a Figura 16 e conforme explicado por (PETRUZELLA, 2014) e (ZANCAN, 2011), um CLP é dividido em partes:

**Figura 16 – Estrutura básica de uma CLP**



**Fonte:** (ZANCAN, 2011)

- a) Fonte de alimentação: tem por finalidade converter a tensão de alimentação (110 a 220 Vca) para a tensão de alimentação dos circuitos eletrônicos, sendo responsável pelo fornecimento da energia necessária nos níveis de tensão exigidos em corrente contínua CC para a alimentação da CPU e dos módulos de entrada e de saída plugados ao rack, bem como manter a carga da bateria;

- b) Unidade central de processamento: é o cérebro do CLP e consiste, geralmente, de um microprocessador que, coadjuvado das memórias EPROM e da memória RAM, é responsável pela implementação lógica e controle das comunicações entre os módulos e utiliza as memórias para armazenar os resultados das operações lógicas executadas. A CPU, por meio do programa de lógica introduzido pelo usuário, controla todas as atividades do processo por meio de uma rotina repetitiva de varredura ou exploração chamada de SCAN;
- c) Memória do programa monitor: é responsável por gerenciar todas as atividades do CLP. Este programa não pode ser alterado pelo usuário, sendo armazenado em memórias do tipo PROM, EPROM ou EEPROM e funciona de forma semelhante ao sistema operacional dos computadores;
- d) Memória do usuário: nesta memória é armazenado o programa desenvolvido pelo usuário, o qual pode ser alterado, tornando flexível a programação. Esse programa geralmente é armazenado em memórias do tipo RAM, EPROM, EEPROM e FLASH-E PROM, cuja capacidade varia de acordo com a marca e o modelo de CLP;
- e) Memória de dados: tem por finalidade armazenar os dados do programa do usuário, tais como valores de temporizadores, contadores, senhas etc. Geralmente, a memória de dados utiliza partes da memória RAM do CLP;
- f) Memória imagem das entradas e saídas: esta memória armazena informações dos estados das entradas e saídas do CLP, funcionando como uma tabela em que a CPU buscará informações durante o processamento do programa de usuário;
- g) Seção de entrada/saída (E/S): são as conexões digitais e analógicas para sensores e atuadores. O sistema de E/S tem finalidade de condicionar os vários sinais recebidos ou enviados para os dispositivos de campo externos;
- h) Unidade de comunicação: responsável pela comunicação com dispositivos de programação;
- i) Bateria: tem por finalidade manter a alimentação do circuito do relógio de tempo real e manter parâmetros ou programas na memória do tipo RAM quando faltar energia elétrica;
- j) Circuitos auxiliares: circuitos responsáveis pela proteção de falhas na operação do CLP, tais como:

- POWER ON RESET: quando se energiza um equipamento eletrônico digital não é possível prever o estado lógico dos circuitos internos. Para que não ocorra um acionamento indevido de uma saída, que pode causar um acidente, existe um circuito encarregado de desligar as saídas no instante em que se energiza o equipamento. Assim que o microprocessador assume o controle do equipamento, esse circuito é desabilitado.
- POWER DOWN: o caso inverso ocorre quando um equipamento é subitamente desenergizado. O conteúdo das memórias pode ser perdido. Existe um circuito responsável por monitorar a tensão de alimentação e em caso de o valor desta cair abaixo de um limite predeterminado o circuito é acionado, interrompendo o processamento para avisar o microprocessador e armazenar o conteúdo das memórias em tempo hábil;
- WATCHDOG TIMER: para garantir, no caso de falha do microprocessador, que o programa não entre em loop, o que seria um desastre, existe um circuito denominado “cão de guarda”, que deve ser acionado em intervalos de tempo predeterminados . Caso não seja acionado, ele assume o controle do circuito sinalizando uma falha geral.

De acordo com Petruzella (2014) também é importante ressaltar que quando se fala em arquitetura do CLP se faz referência ao equipamento, ao programa do CLP ou a uma combinação dos dois. Um sistema de arquitetura aberta permite que o equipamento seja conectado facilmente aos dispositivos e programas de outros fabricantes e utilizam-se componentes de prateleira que seguem padrões aprovados. De outro lado, um sistema com arquitetura fechada é aquele cujo projeto é patenteado, tornando-o mais difícil de ser conectado a outros sistemas. Considerando-se que a maioria dos sistemas de CLP é patenteada, torna-se necessário verificar se o equipamento ou programa genérico que será utilizado é compatível com um CLP específico (PETRUZELLA, 2014).

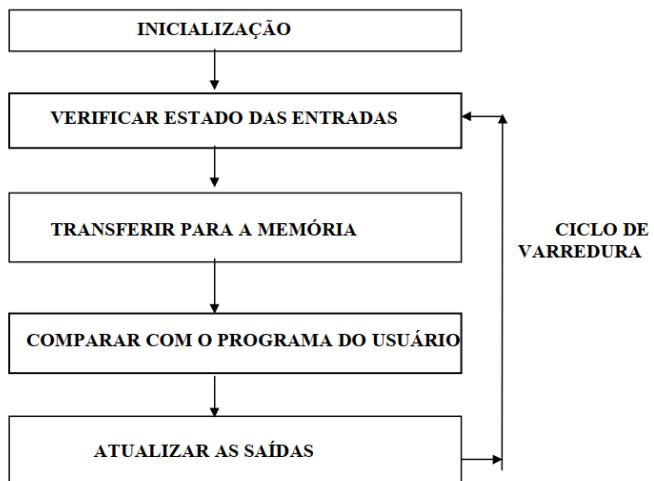
### 3.1.2 Processo Scan

Conhecido como Scan, pode ser definido como um processo de varredura, leitura ou exploração, cuja função básica é ler, por meios das entradas, os valores dos dispositivos de campo. Em seguida, o CLP fará uma análise desses dados por meio do processador, que, de

acordo com a lógica de controle inserida pelo usuário, ligará ou desligará, por meio da interface de saída, os dispositivos de campo (Figura 17).

O processo de Scan é normalmente um processo contínuo e sequencial que precisa de um tempo para ser concluído. O tempo total que o controlador precisa para completar esse ciclo de varredura, processamento da lógica e atualização da porta E/S é chamado de Scan Time, o qual depende das memórias utilizadas pelo programa de controle e do tipo de instruções usadas por este, variando de alguns décimos de segundo até 50 milissegundos. No entanto, esse tempo pode aumentar dependendo de uso de subsistemas remotos, pois o CLP precisará de um tempo maior para transmitir e receber dados desses sistemas remotos. (BRYAN; BRYAN, 2002).

**Figura 17 – Princípio de funcionamento – Diagrama em blocos**



**Fonte:** Autoria própria

### 3.1.3 Modulo E/S

Entende-se como processo de controle um método no qual estão presentes variáveis digitais (discretas) e variáveis analógicas (BRYAN; BRYAN, 2002). No entanto, para que um CLP controle adequadamente um determinado processo é necessário que possua dispositivos de entrada/saída compatíveis com as variáveis do processo:

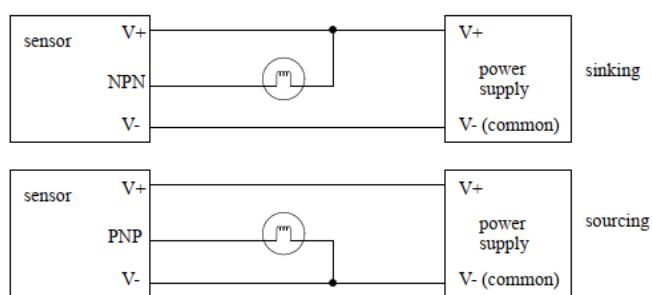
- a) Entradas/saídas digitais: essa interface fornece a conexão física para os dispositivos de campo transmitirem e receberem sinais digitais, os quais podemos definir como sinais não contínuos de apenas dois estados, ON e OFF. Essa interface é apta a identificar a presença ou não de um sinal elétrico provindo de um determinado dispositivo, dentro de uma determinada faixa de valores, reconhecendo a presença

do sinal, mas não sua amplitude;

- b) Entradas/saídas analógicas: essa interface, que também mantém uma conexão física como os dispositivos de campo, geralmente é empregada em processos que exigem um controle mais preciso; esses sinais elétricos podem ser de tensão ou corrente, cuja faixa de valores mais utilizada é, respectivamente, 0 a 10V e 4mA a 20mA.

Ainda segundo (ZANCAN, 2011), as entradas digitais possuem isolação elétrica por meio de acopladores ópticos e podem ser do tipo PNP ou tipo fonte (Sourcing) e NPN ou tipo dreno (Sinking), como mostrado na Figura 18. Neste trabalho será apresentada só a configuração de tipo fonte.

**Figura 18 – Sinksource**



**Fonte:** (ZANCAN, 2011)

### 3.1.4 Programação dos CLPs

As linguagens de programação "de alto nível", assim chamadas por serem mais próximas da linguagem utilizada para comunicação entre pessoas, reduziram drasticamente o tempo de programação do CLP por não terem o inconveniente de obrigar o programador a conhecer detalhadamente a arquitetura do microprocessador do CLP.

A programação do CLP é um conjunto de instruções ou comandos desenvolvidos pelo usuário do equipamento para que ele execute determinada ação. As linguagens de programação estabelecem regras para combinar as instruções de forma a atender o que é desejado, no entanto, foi criada a norma IEC 61131-3 para que cada desenvolvedor de CLP não venha a ter a sua própria linguagem de programação; discrimina os tipos de linguagem de programação textual que devem ser usadas (PRUDENTE, 2008; BOLTON, 2006):

- Lista de instruções – IL (instruction list): este modo de programação utiliza uma sequência de instruções tipo texto de baixo nível. É constituída por uma série de códigos “máquina”.

ou seja, utiliza as instruções do processador diretamente;

- Texto estruturado – ST (structural text): este modo de programação utiliza, como o seu próprio nome indica, texto estruturado de alto nível, ou seja, linguagens de programação no nível de Pascal, C++, Basic, entre outras. É o tipo de linguagem mais potente para CLP, aquela de mais versatilidade, mas tem contra si o fato de ser a mais complicada de entender e programar. Requer alguns conhecimentos de programação avançada;
- Diagrama ladder – LD: como o nome indica, do inglês ladder (escada), o tipo de representação desta linguagem faz-se tipo escada, com uma barra vertical do lado esquerdo chamada barra de alimentação, uma barra vertical do lado direito chamada de retorno comum ou massa e uma linha horizontal dividida em duas partes, sendo que a do lado esquerdo é a zona destinada às entradas e a do lado direito é a zona destinada às saídas. É talvez a linguagem de programação mais utilizada mundialmente para a programação de CLPs;
- Diagrama de blocos funcionais – FBD: é uma linguagem constituída por uma série de blocos que utilizam simbologia lógica combinatória (AND, OR, XOR, etc). É mais utilizada por quem tem prática na eletrônica digital e lógica, em que a simbologia é a mesma;
- Sequential function chart – SFC: primeiramente chamada Grafcet (acrônimo do francês graphe fonctionnel de commande, étapes transitions), este tipo de linguagem é diferente das anteriores pelo modo como são utilizados os diagramas funcionais, tendo até uma norma apenas para descrever essas regras, a Norma IEC 60848, que se baseia no sequenciamento das tarefas a executar pelo CLP de forma automática, ou seja, as instruções contidas nas transições apenas são cumpridas quando as condições inseridas no passo anterior estiverem satisfeitas. Dessa forma, cria-se uma sequência de eventos que são executados passo a passo. É escrita com linguagem gráfica e utiliza comandos alfanuméricos.

### 3.1.4.1 Controladores PID

Controladores PID (proporcionais, integrais, derivativos), são os mais utilizados na indústria e por isso objeto de estudo na área de automação. Eles vieram para minimizar a característica de oscilação dos controladores de liga/desliga e são empregados nas aplicações em

malha fechada, em que a característica fundamental do processo controlado deve ficar invariável (AGUIRRE, 2019). O objetivo desses controladores é que a ação do controle é composta por uma parcela proporcional ao erro, uma integral ao erro e uma proporcional derivada do erro. Esse erro se deve à comparação da variável controlada do processo com a referência (PRUDENTE, 2008). Assumindo  $u(t)$  como sinal de saída, pode-se estabelecer o algoritmo do PID de acordo com a equação 1:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \quad (1)$$

em que  $K_p$  é o ganho proporcional,  $K_i$  é o ganho integral,  $K_d$  é o ganho derivativo,  $e$  o erro,  $t$  o tempo e  $\tau$  é o tempo de integração.

### 3.2 COMUNICAÇÃO SERIAL EM CONTROLADORES

A comunicação de dados serial é implementada usando padrões como RS-232, RS-422 e RS-485. O RS significa padrão recomendado, e especifica as características elétricas, mecânicas e funcionais para comunicações seriais. As interfaces de comunicação serial são integradas ao módulo do processador ou vêm como um módulo de interface de comunicação separado. O tipo mais simples de conexão é a porta serial RS-232.

As interfaces RS são usadas para se conectar os dispositivos como sistemas de visão, código de barras leitores, e terminais de operação que devem transferir quantidades de dados a uma taxa razoavelmente alta entre o controle remoto do dispositivo e o CLP. O tipo RS-232 de transmissão serial é projetado para se comunicar entre um computador e um controlador e geralmente é limitado a comprimentos de até 15 metros. Tipos de transmissão serial RS-422 e RS-485 são projetados para se comunicarem entre um computador e vários controladores, têm um alto nível de imunidade a ruído, e são geralmente limitados a comprimentos de 200 metros (para RS-485) ou 500 metros (para RS-422).

#### 3.2.1 DeviceNets

DeviceNet é uma rede aberta em nível de dispositivo. Tem uma velocidade relativamente baixa, mas eficiente no tratamento de mensagens curtas de e para os módulos de E/S. À medida que os PLCs se tornaram mais poderosos, eles estão sendo necessários para controlar um número crescente de dispositivos de campo de E/S. Portanto, às vezes pode não ser prático conectar

separadamente cada sensor e atuador diretamente nos módulos de E/S. Os sistemas convencionais têm racks de entradas e saídas com cada dispositivo de E/S conectado de volta ao controlador. O protocolo DeviceNet reduz drasticamente os custos integrando todos os dispositivos de E/S em uma rede tronco de 4 fios com dados e condutores de energia no mesmo cabo. A conectividade direta reduz o custo e o consumo de tempo fiação. A função básica de uma rede de barramento DeviceNet E/S é comunicar informações, bem como fornece energia para, os dispositivos de campo que estão conectados ao barramento. O CLP aciona os dispositivos de campo diretamente com o uso de um Scanner de rede em vez de módulos de E/S (BOLTON, 2006).

O módulo Scanner se comunica com dispositivos DeviceNet na rede para:

- Ler as saídas de um dispositivo;
- Escrever nas entradas de um dispositivo;
- Atualizar dados de configuração;
- Monitorar o status operacional de um dispositivo.

O módulo scanner se comunica com o controlador para trocar informações que incluem:

- Dados de I / O do dispositivo;
- Informações de status;
- Dados de configuração.

O DeviceNet também tem a característica única de poder alimentar a rede. Isso permite que dispositivos com requisitos de energia se alimentem diretamente da rede, reduzindo ainda mais os pontos de conexão e o tamanho físico. O DeviceNet usa o protocolo industrial comum, chamado CIP, que é estritamente orientado a objetos. Cada objeto tem atributos (dados), serviços (comandos) e comportamento (reação a eventos). Dois tipos diferentes de objetos são definidos na especificação CIP: objetos de comunicação e objetos específicos do aplicativo. Uma rede DeviceNet pode suportar até 64 nós e a distância de ponta a ponta da rede é variável, com base na velocidade da rede.

Comunicações de dados são transportados por dois fios com um segundo par de fios para alimentação. Os dispositivos de campo que estão conectados à rede podem comunicar não apenas o status de ligado/desligado, mas também informações de diagnóstico sobre seu estado operacional. Por exemplo, pode-se detectar através da rede que um sensor fotoelétrico

está perdendo margem por causa de uma lente suja, e pode-se corrigir a situação antes que o sensor falhe em detectar um objeto.

### 3.2.2 ControlNet

O ControlNet está posicionado um nível acima do DeviceNet. Isto usa o Protocolo Industrial Comum (CIP) para combinar a funcionalidade de uma rede E/S e uma rede ponto a ponto, fornecendo desempenho de alta velocidade para ambas as funções. Essa rede aberta de alta velocidade é altamente determinística e repetível. Determinismo é a capacidade de prever com segurança quando os dados serão entregues e a repetibilidade garante que os tempos de transmissão sejam constantes e não sejam afetados por dispositivos que se conectam ou saem da rede. As Electronic Data Sheets (EDS), o quais são arquivos ASCII que descrevem como um aparelho pode ser usado em uma rede, descrevem os objetos, os atributos e os serviços disponíveis no dispositivo. Esses arquivos são obrigatórias para cada dispositivo ControlNet pois, durante a fase de configuração, o scanner ControlNet configura cada dispositivo de acordo com os arquivos EDS.

### 3.2.3 EtherNet/IP

EtherNet / IP (protocolo industrial Ethernet) é um sistema de protocolo de comunicação aberto baseado na camada Common Industrial Protocol (CIP) usada em DeviceNet e ControlNet. Ele permite que os usuários veiculem informações perfeitamente entre dispositivos executando o protocolo EtherNet/IP sem hardware personalizado. Alguns dos recursos importantes do EtherNet/IP são:

- Compartilhamento de aplicativo em camada plug-and-play entre ControlNet, DeviceNet e Ethernet/IP permite que dispositivo de vários fornecedores se conversem. Conexão plug-and-play refere-se à capacidade de um sistema de computador para configurar dispositivos automaticamente. Isso permite a conexão um dispositivo sem se preocupar em configurar interruptores DIP, jumpers e outros elementos de configuração;
- EtherNet/IP fornece uma comunicação full-duplex padronizada utilizando a máxima largura de banda possível. A largura de banda se refere a transferência de dados, taxa suportada por uma rede, comumente expressa em termos de bits por segundo. Quanto

maior for a largura de banda, maior será o desempenho geral;

- EtherNet/IP permite interoperabilidade de sistemas industriais, dispositivos de automação e equipamentos de controle na mesma rede usada para aplicativos e navegação na internet.

### 3.2.4 Foundation Fieldbus

Fieldbus é um sistema de comunicação aberto, serial e bidirecional que interconecta equipamentos de medição e controle, como sensores, atuadores e controladores. No nível de base na hierarquia de redes de plantas, serve como uma rede para dispositivos de campo usados no controle de processo. Existem várias topologias possíveis para redes fieldbus. Com esta topologia, o cabo Fieldbus é roteado de dispositivo para dispositivo. As instalações que usam esta topologia requerem conectores, de modo que a desconexão de um único dispositivo é possível sem interromper a continuidade de todo o segmento.

### 3.2.5 Profibus-DP

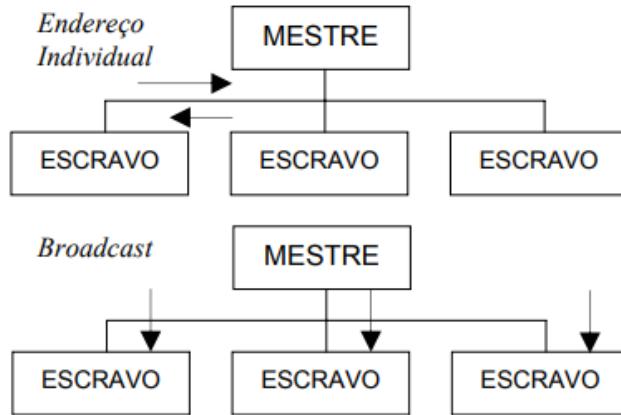
PROFIBUS-DP (onde DP significa Descentralização Periférica) é um padrão de comunicação fieldbus internacional aberto que suporta tanto sinais analógicos quanto discretos. É funcionalmente comparável ao DeviceNet. Os meios físicos são definidos via RS-485 ou fibra, que é uma tecnologia de transmissão óptica. PROFIBUS-DP se comunica em velocidades de até 12 Mbps em distâncias de até 1200 metros.

## 3.3 PROTOCOLO MODBUS

O protocolo Modbus foi desenvolvido pela Modicon Industrial Automation Systems (atualmente Schneider Electric). É um protocolo de comunicação aberto, baseado no paradigma de comunicação mestre-escravo, em que o mestre é o responsável por coordenar diversos escravos (figura 19). No campo das redes industriais, tornou-se muito popular devido à sua simplicidade de implementação, sendo hoje um dos protocolos mais utilizados em controladores e ferramentas para desenvolvimento de sistemas supervisórios (MODBUS-IDA, 2004).

A comunicação Modbus pode ser do tipo transmissão de série assíncrona, executando trocas de pacotes por meio das interfaces RS232 e RS485, e de tipo Modbus/TCP. No tipo serial a comunicação pode ser realizada de duas formas: RTU (*remote terminal unit*) e ASCII (*American*

**Figura 19 – Modelo mestre-escravo**



Fonte: (NASCIMENTO; LUCENA, 2003)

*standard code for information interchange).* Esses dois modos de comunicação diferenciam-se na representação dos seus dados, tendo um formato de pacote específico cada um. Os pacotes Modbus são compostos de seis campos, caracterizados por dados binários no modo RTU, e por caracteres no modo ASCII (KOBAYASHI, 2009):

- Início: aponta o começo do pacote;
- Endereço: aponta qual dispositivo receberá ou enviará o pacote. A faixa de endereços válidos dos dispositivos varia de 0 a 247, sendo o endereço 0 (zero) utilizado para mensagens que são enviadas para todos os escravos (broadcast), representado por 8 bits no modo RTU ou por dois caracteres no modo ASCII;
- Função: este campo indica qual ação deve ser realizada;
- Dados: campo onde existem informações relacionadas com o código da função no campo de funções, como, por exemplo, o número de variáveis discreteas a serem lidas ou ativadas;
- Controle: é responsável pela detecção de erros no pacote (checksum);
- Fim: indica que a comunicação entre os dispositivos foi terminada.

Como mostrado nas figuras 20 e 21, no modo RTU o início e o fim são representados por um intervalo de silêncio e os campos de endereço e função são representados por 8 bits, enquanto a verificação de controle é realizada por meio do algoritmo CRC (*cyclic redundancy check*), sendo representado por 16 bits. No modo ASCII, o início é representado pelo caractere ":" (0x3A em hexadecimal), os campos de endereço e função são representados por dois caracteres

e a verificação de controle é realizada por meio do algoritmo LRC (*longitudinal redundancy check*), enquanto o fim é representado por um par de caracteres de CR (*carriage return*) e de LF (*line feed*) (0x0D e 0x0A em hexadecimal, respectivamente).

**Figura 20 – Formato do Pacote RTU**

Início	Endereço	Função	Dados	CRC	Fim
Silêncio	8 bits	8 bits	n × 8 bits	16 bits	Silêncio

Fonte: Kobayashi (2009)

**Figura 21 – Formato do Pacote ASCII**

Início	Endereço	Função	Dados	LRC	Fim
: (0x3A)	2 caracteres	2 caracteres	n caracteres	2 caracteres	CRLF

Fonte: Kobayashi (2009)

Como já mencionado, o mecanismo de controle de acesso é de tipo mestre-escravo ou cliente-servidor. A estação mestre, a qual é responsável para iniciar as transações, envia uma mensagem (*query*) para os escravos, os quais retornam uma resposta (*response*) (FREITAS, 2014; SOUZA, 2016). Na tabela 1 são mostradas as principais funções do protocolo Modbus, as quais variam de 1 a 255 (01H a FFH), mas apenas a faixa de 1 a 127 (01H a 7FH) é utilizada, já que o bit mais significativo é reservado para indicar respostas de exceção (NASCIMENTO; LUCENA, 2003), sendo que mais detalhes sobre implementações das funções podem ser obtidos na especificação do protocolo (MODBUS-IDA, 2004).

### 3.4 O MICROCONTROLADOR AVR

Um microcontrolador é um sistema microprocessado que possui vários periféricos, tais como memória de programa, memória de dados, memória RAM, temporizadores/contadores, gerador interno de clock, conversores analógicos-digitais(ADCs), saídas PWM e interfaces de comunicação. Basicamente um microcontrolador é um circuito integrado com uma arquitetura repleta de dispositivo voltadas a aplicações de controle(LIMA, 2012). No mercado existe uma grande variedade de microcontroladores, cada um com sua características, portanto a escolha desse depende exclusivamente das exigências do projeto. Neste trabalho foi escolhido o microcontrolador AVR, sendo ele integrado na placa UNO R3, a qual teve mais sucesso dentre os produtos da família Arduino. O microcontrolador AVR(Alf-Veegard-RISC), cuja sigla advém

**Tabela 1 – Funções especificadas no protocolo Modbus**

Código de função	Descrição
01	Ler o estado dos registradores booleanos de saída
02	Ler o estado de entradas discretas
03	Ler o conteúdo de um bloco de registradores de espera
04	Ler registradores de entrada
05	Escrever em um registrador booleano de saída
06	Escrever em um registrador de espera
07	Ler o conteúdo de oito estados de exceção
08	Prover uma série de testes para verificação da comunicação e erros internos
11	Obter o contador de eventos
12	Obter um relatório de eventos
15	Escrever em uma sequência de saídas
16	Escrever em um bloco de registradores
17	Ler algumas informações do dispositivo
20	Ler informações de um arquivo
21	Escrever informações em um arquivo
22	Modificar o conteúdo dos registradores de espera através de operações lógicas
23	Combina ler e escrever em registradores numa única transação
24	Ler o conteúdo da fila FIFO de registradores

**Fonte:** Kobayashi (2009)

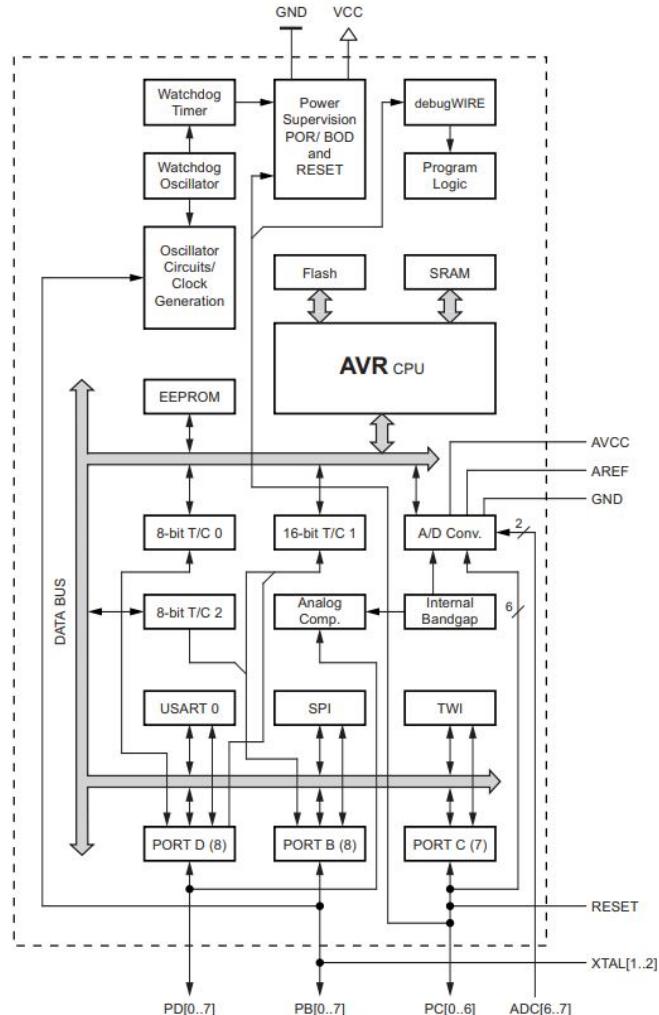
dos nomes dos desenvolvedores faz uso de 8 bits e contém um processador RISC. A tecnologia RISC tem um desempenho melhor da CISC devido sua uma arquitetura, em que as instruções são de tamanho fixo gastando o mesmo tempo de execução e acessando a memória de forma simples e uniforme. O processador AVR de 8 bits, trabalha com dados de 8 bits, sendo suas instruções de 16 bits, 32 bits ou mais. A maioria dessas instruções é executada em 1 ou 2 períodos de relógio, assim com um cristal de 16MHz, comuns na plataformas Arduino, o microcontrolador consegue executar aproximadamente 16 milhões de instruções por segundo (ZELENOVSKY; MENDONÇA, 2019). É importante salientar que o processador faz uso da arquitetura Harvard, a qual permite manter separada a memória do programa e de dados. Isto permite um processo de programação simples, ou seja, uma estrutura voltada a programação em C, permitindo a produção de código compacto.

### 3.4.1 ATMega328

O ATMega328, mostrado na figura 22, é o microcontrolador integrado na plataforma UNO R3 da família Arduino. A principal vantagem desse microcontrolador é possuir uma memória Flash (memória de programa) de 32 kbites, isto é, uma memória de maior tamanho se comparados a microcontroladores AVR como mesmos números de pinos. Possui 23 terminais I/O, 6 entradas analógica de 10 bit e saída digitais com modulação PWM, comunicação serial com interface UART, SPI, TWI e 3 timers. Pode ser alimentado com tensões de 2,7 até 5V e um clock

que pode chegar até 20MHz. Além da memória Flash, dispõe de uma memória SRAM de 2 kbites e uma EPROM de 1 kbite para armazenar variáveis mesmo com a alimentação interrompida.

**Figura 22 – Arquitetura Microcontrolador ATMega328**



**Fonte:** (LIMA, 2012)

### 3.4.2 Comunicação serial do microcontrolador

Um meio muito utilizado para comunicação entre dois dispositivos é o modo serial. Esse tipo de comunicação envia dados serialmente, isto é bit a bit, ocasionando uma diminuição de fios a serem utilizados se comparado a comunicação paralela. Pode-se caracterizar em 2 tipos:

- Comunicação serial síncrona: onde existe um sinal de clock que marca o instante que cada bit é disponibilizado, e faz uso de 3 fios: um para dados, um para clock e um para referência de terra.

- Comunicação serial assíncrona: onde não existe um sinal de clock para validar os dados. Também faz uso de 3 fios: um para transmissão, um para recepção e um para referência de terra.

Na comunicação assíncrona, antes de iniciar-se a comunicação, precisa-se definir o Baudrate, isto é a taxa de bits transmitido por segundo. Além do mais, para que o receptor possa entender o início e o final da mensagem usa-se um bit de partida e um bit de parada. Enfim, é importante destacar que a comunicação síncrona, é uma comunicação Half Duplex, isto é unidirecional, pois não é possível fazer transmissão e recepção ao mesmo tempo, pois o transmissor é responsável para o clock. Já na comunicação síncrona, tendo 2 linhas de dados é possível transmitir e receber ao mesmo tempo, sendo essa simultaneidade de transmitir/receber chamada de Full Duplex.

### 3.4.3 UART

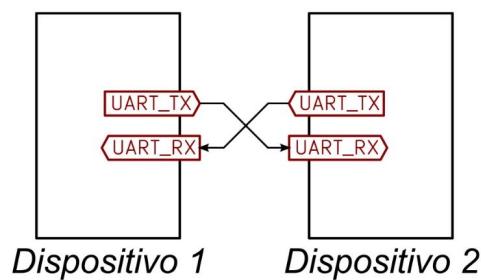
Um periférico de comunicação padrão em processadores AVR, entre outros, é o UART (Universal Asynchronous Receiver/Transmitter). Essa interface é uma versão generalizada dos protocolos EIA, RS-232, RS-422 e RS-485, cujo objetivo é ser customizável, podendo ser utilizado em qualquer um desses protocolos. Todavia, enquanto os protocolos citados trabalham com tensões positivas e negativas, no UART as tensões são compatíveis como os níveis de tensão do microcontrolador: 3,3 ou 5V (ALMEIDA *et al.*, 2017). As principais características são:

- Operação Full Duplex (transmissão e recepção simultânea);
- Operação síncrona e assíncrona;
- Operação síncrona com clock mestre ou escravo;
- Gerador de Baud rate (taxa de bits) com alta resolução;
- Operação com dados de 5, 6, 7, 8 ou 9 bits e 1 ou 2 bits de parada;
- Gerador de paridade par ou ímpar e conferencia de paridade por hardware;
- Detecção de colisão de dados e erros de frames.
- Filtro para ruído para partidas falsas;
- três interrupções: Tx completa, Registrador de dados vazio e Rx completa;

- Modo de comunicação assíncrono com velocidade duplicável;

Como mostrado na figura 23, a comunicação entre dois dispositivos por meio de UART é realizada conectando o sinal do terminal de transmissão do primeiro dispositivo com o terminal receptor do segundo dispositivo, e o sinal do terminal de transmissor do segundo dispositivo com o terminal receptor do primeiro, além disso os dois dispositivos devem ter a mesma referência de terra.

**Figura 23 – Comunicação UART**



Fonte: (ALMEIDA *et al.*, 2017)

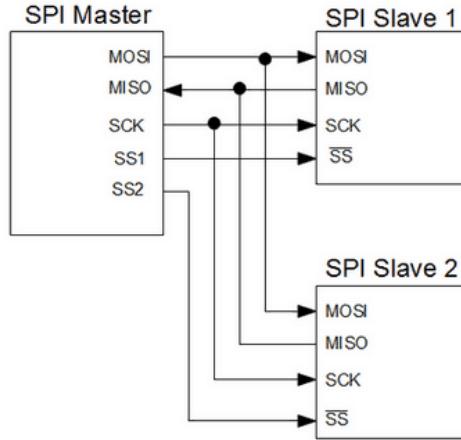
#### 3.4.4 Interface SPI

A diferença da UART, que faz uso de 2 pinos: Rxd e Txd, a interface SPI (Serial Peripheral Interface) é uma comunicação serial síncrona de curta distância Full Duplex utilizada geralmente em arquitetura mestre escravo, que utiliza 4 pinos:

- MOSI (Master Output, Slave Input), transmissão pelo mestre e recepção pelo escravo;
- MISO (Master INput, Slave Output), recepção pelo mestre e transmissão pelo escravo;
- SCK (SerialClock), gerado pelo mestre;
- SS (Slave Select), seleção do escravo.

Essa interface, que tem como padrão a comunicação entre microcontroladores e periféricos, possui uma taxa de operação superior a 20 MHz e pode enviar mensagem de qualquer tamanho com conteúdo e finalidade arbitrária. Em uma rede SPI, figura 24, somente um dispositivo pode ser configurado como mestre, os outros devem ser configurados como escravos.

**Figura 24 – Comunicação SPI, mestre e escravos**



**Fonte:** Autoria própria

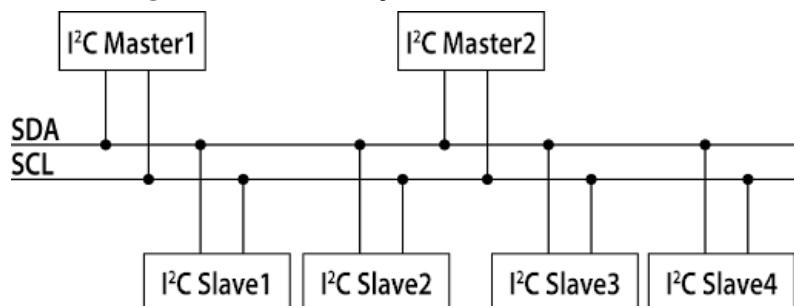
### 3.4.5 Interface TWI

A terceira interface serial é a TWI (Two Wire Interface). Esse tipo de comunicação é compatível com a I<sup>2</sup>C (Inter-Integrated Circuit), o qual é um barramento serial multimestre desenvolvido pela Philips, figura 25. Possui uma arquitetura mestre escravo e endereço de dispositivo de 7 bits, permitindo a conexão de até 127 dispositivos. Caracterizada para ser uma comunicação serial síncrona, o clock é enviado junto com o sinal e possui uma velocidade máxima de 400KHz. Essa interface faz uso de 2 pinos:

- SCL (Serial Clock) sinal de clock;
- SDA (Serial Data) comunicação de dados;

Em uma rede TWI, cada dispositivo recebe um identificador para evitar erros no envio de informações

**Figura 25 – Comunicação TWI, mestre e escravos**

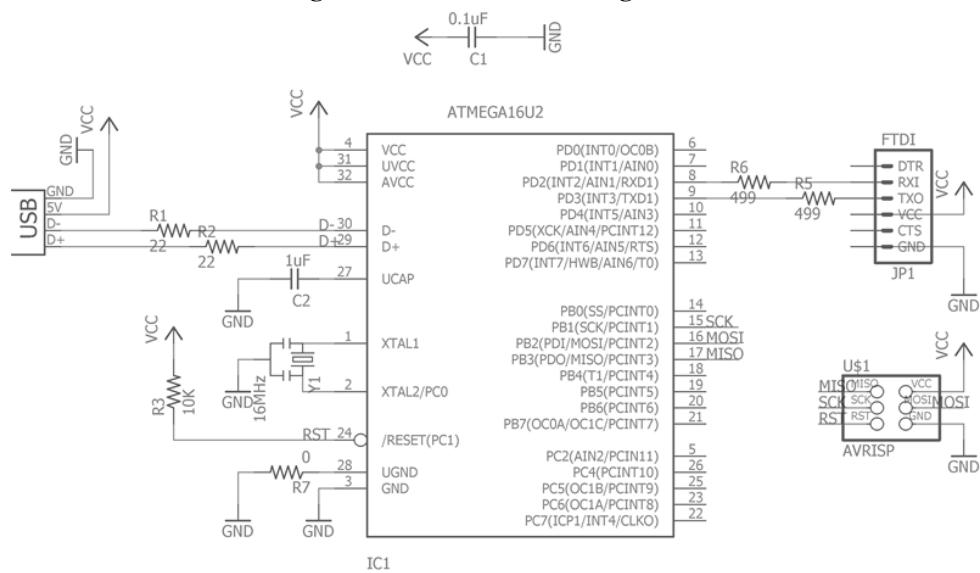


**Fonte:** Autoria própria

### 3.4.6 Interface USB

A interface USB (Universal Serial Bus) é um padrão que simplificou a comunicação entre dispositivos, periféricos e computadores, visando estabelecer modelo de cabos, conectores, protocolos de comunicação e sistema de alimentação para dispositivos. Devido a essa padronização, não é mais necessário obter driver para cada dispositivo. Todavia, nem todos os microcontroladores tem periférico USB nativo, portanto em ausência desse periférico, são associados conversores UART/USB aos microcontroladores, os quais são compatíveis com computadores, facilitando a programação do código para leitura de sinais dos microcontroladores. (ALMEIDA *et al.*, 2017). Na figura 26 é mostrado um exemplo do circuito conversor presente na plataforma Arduino Uno R3.

**Figura 26 – Conversor ATmega16U2**



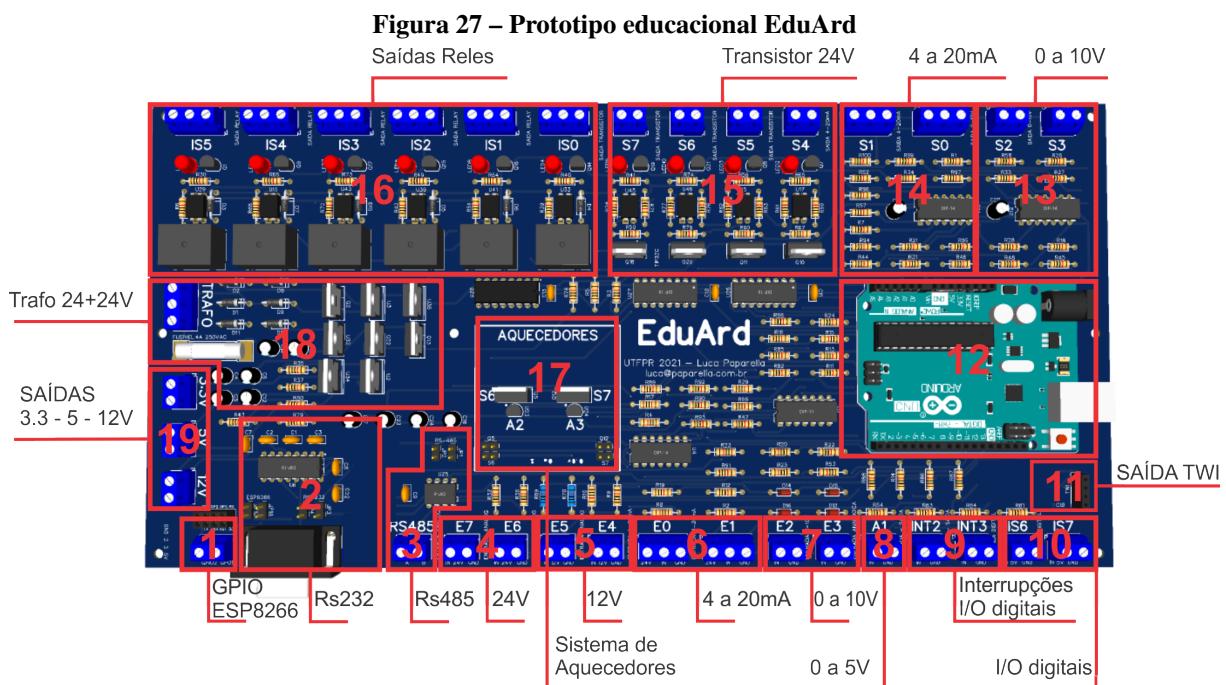
**Fonte:** (ARDUINO, 2018)

## 3.5 CONCLUSÕES DO CAPÍTULO 3

Este capítulo apresentou as tecnologias essenciais de um CLP, como arquitetura, linguagem de programação e comunicação de rede. Dentre as tecnologias apresentadas, o protótipo que será apresentado sucessivamente utilizará algumas dessas configurações, como módulos E/S, Watchdog Timer, sintonia PID. Enfim um protocolo de comunicação Modbus permitirá ao protótipo de se comunicar com o supervisório.

## 4 ARQUITETURA DO PROTÓTIPO PROPOSTO

Esta seção tem por objetivo apresentar a metodologia empregada no desenvolvimento da plataforma hardware doravante denominada EDUARD (Educational Arduino), proposta nesta dissertação mostrada na figura 27, bem como o desenvolvimento do software embarcado no microcontrolador e do software de gerenciamento do sistema. O sistema foi planejado para atuar como controlador lógico programável ou transmissor de dados. A especificações deste protótipo estão de acordo com o requisitos mostrados na tabela 2.



Fonte: Autoria própria

Tabela 2 – Requisitos da plataforma hardware

Símbolo	Parâmetro
Alimentação	Fonte linear com transformador externo
Modulo microcontrolador	Arduino Uno ou compatível
Entradas analógicas	4 a 20mA, 0 a 5V e 0 A 10V
Saídas analógicas	4 a 20mA e 0 a 10V
Entrada digitais	0-5V, 0-12V e 0-24V
Saídas digitais	transistor e relay
Comunicação	Serial TTL, RS232, RS485 I2C

Fonte: Autoria própria

Nesse contexto, com os requisitos definidos, a metodologia se desenvolve seguindo os procedimentos descritos:

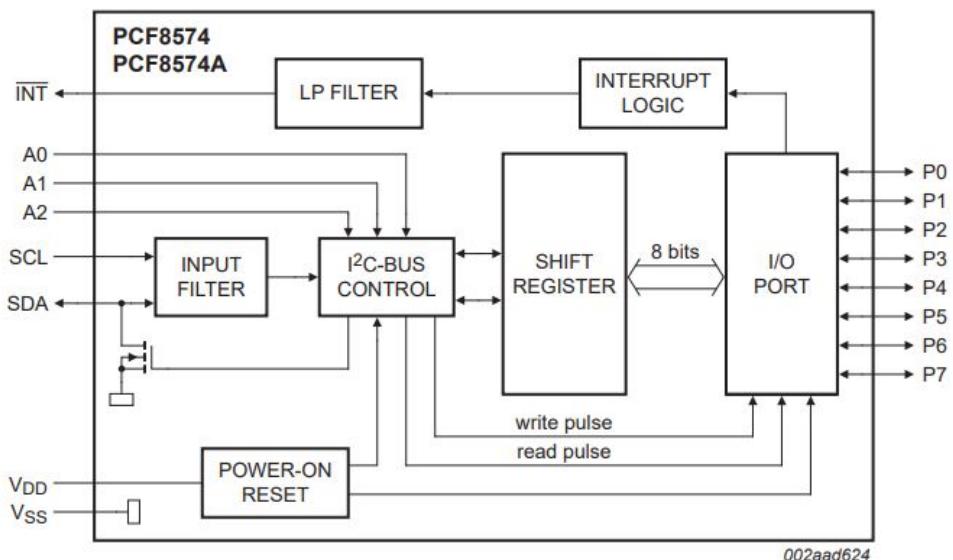
- Desenvolvimento de portas de entrada e de saída multiplexadas;

- Criação do layout da plataforma hardware;
- Criação dos softwares para embarcar no microcontrolador;
- Reprodução do modelo de uma planta industrial em Python.

#### 4.0.1 Expansão de portas TWI

Devido aos poucos pinos disponíveis para entradas e saídas digitais, no ARduinuno Uno R3 (módulo 12), propõe-se um circuito expensor. O CI PCF8574 é um expensor de portas I2C de 8 bits, utilizado para acionar até oito pinos do chip utilizando apenas duas portas do microcontrolador, liberando assim as demais portas para outras tarefas. O PCF8574 é um CI com baixo consumo de corrente e pinos com potência suficiente para controlar dispositivos como leds mas que também pode-se usar como saídas (e entradas) de uso geral, acionando relés, atuadores ou recebendo dados de sensores, figura 28. Os pinos Vdd e Vss são os pinos de alimentação (Vdd

**Figura 28 – Remote 8-bit I/O expander for I2C-bus with interrupt**



**Fonte:** Datasheet PC8574, NXP

é o positivo, 2,5 à 6VDC, e Vss é o GND). Tem-se também um pino INT, usado para trabalhar com interrupções do microcontrolador. Os pinos SDA SCL são a interface I2C e os pinos P0 à P7 são saídas/entradas do PCF8574. Enfim, os pinos A0, A1 e A2 são os de endereçamento do PCF8574, endereço esse que será usado para comunicação I2C identificando o dispositivo no barramento. O PCF8574 pode ter até 8 endereços diferentes, o que significa que no EDUARD poderia ter ao mesmo tempo até 8 desses chips, totalizando 64 portas adicionais. Então para

selecionar o endereço I2C que se deseja usar com o chip, basta seguir a tabela da figura 29, atribuindo o valor 0 ou 1 aos pinos A0, A1 e A2.

**Figura 29 – Remote 8-bit I/O expander for I2C-bus with interrupt**

Pinos endereço			PCF8574P
A0	A1	A2	Endereço I2C
0	0	0	32 (decimal) 20 (hex)
1	0	0	33 (decimal) 21 (hex)
0	1	0	34 (decimal) 22 (hex)
1	1	0	35 (decimal) 23 (hex)
0	0	1	36 (decimal) 24 (hex)
1	0	1	37 (decimal) 25 (hex)
0	1	1	38 (decimal) 26 (hex)
1	1	1	39 (decimal) 27 (hex)

**Fonte:** Datasheet PC8574, NXP

#### 4.0.2 Multiplexador CD4051

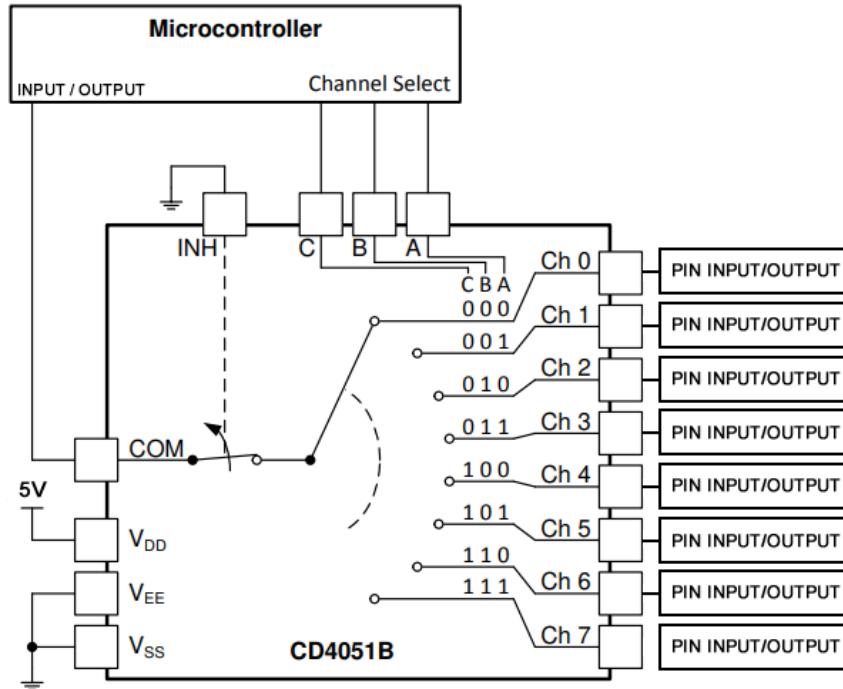
Conhecido por Multiplexador, Multiplexer ou Mux, o CD4051, a princípio, é uma chave seletora digital, pois a partir do código BCD que é enviado às entradas de seleção são ativados seus respectivos canais, que são os canais de 0 a 7, totalizando 8 canais. Uma de suas maiores vantagens na prática comparado a outros CI's da categoria é que seus canais podem ser utilizados a partir dos sinais analógicos também, ou seja, trabalha ainda com as tensões de 0 a 5 Vcc, o que o torna apropriado para esta aplicação onde há necessidade de leitura ou emissão de sinais analógicos.

Pela figura 30, pode-se notar como as combinações dos canais C, B e A correspondem ao número binário da porta, por exemplo quando a saída C-B-A é 0-0-0 obtém-se a ativação do pino Ch 0, já quando a saída C-B-A é 1-1-1 obtém-se a ativação do pino Ch 07. Esta arquitetura permite uma fácil programação de ativação de pinos como sugerido no algoritmo 1.

No EDUARD proposto nesse trabalho usam-se dois circuitos integrados CD4051, o primeiro gerenciará os módulos vistos na figura 27:

- 2 saídas analógicas 4 a 20mA (módulo 14);
- 2 saídas analógicas 0 a 10V (módulo 13);
- 4 saídas digitais 0-24V (módulo 15);
- 6 saídas digitais relé (módulo 16);

**Figura 30 – Single 8-Channel Analog Multiplexer/Demultiplexer**



**Fonte:** Datasheet CD4051, Texas Instruments

#### **Algoritmo 1 – Código de exemplo para configurar as saídas do multiplexador**

---

```

1: # Vetor A,B e C
2: const int canal[3];
3: #Canal saida entre 0 e 7
4: int saida;
5: void configuraMux(saida)
6: {
7:   for (int i = 0; i < 3; i++)
8:   {
9:     int estado = bitRead(saida, i);
10:    # Configura combinação A-B-C
11:    digitalWrite(canal[i], estado);
12:  }
13: }
```

---

**Fonte:** Autoria própria

- 2 transistores do sistema aquecedor ou com ativação por Jumpers (módulo 17).

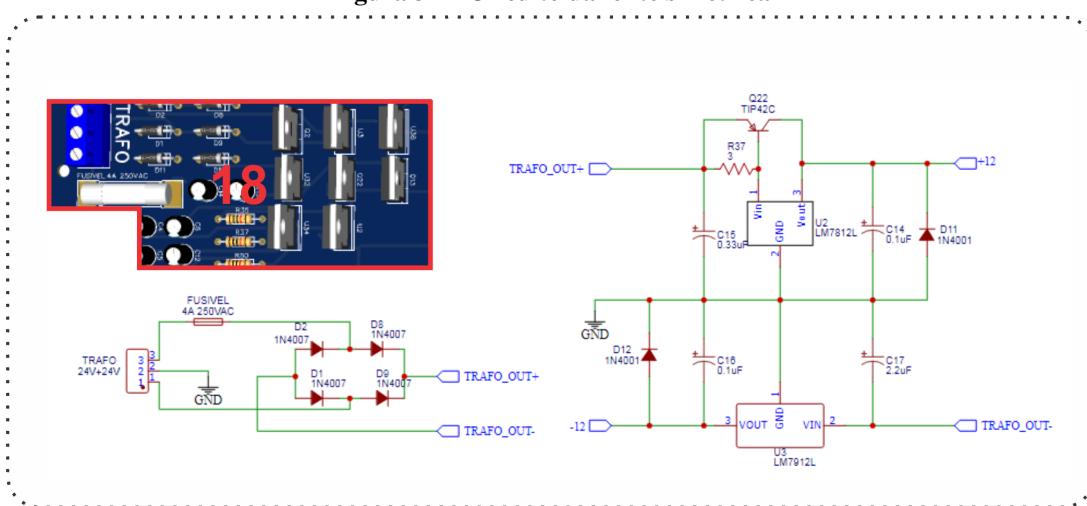
o segundo gerenciará:

- 2 entradas analógicas 4 a 20 mA (módulo 6);
- 2 entradas analógicas 0 a 10V (módulo 7);
- 2 entradas 0-12V (módulo 5);
- 2 entradas 0-24V (módulo 4).

### 4.0.3 Fonte simétrica

Antes de entrar no contexto de entradas e saídas E/S é necessário esclarecer alguns detalhes para o funcionamento delas. Amplificadores operacionais são usados para esse escopo, os quais são muito úteis quando se trata de saídas e entradas analógicas ou buffers. Por isso, salienta-se que para utilizar tais componentes é preciso de tensão negativa, a qual pode ser gerada por uma fonte simétrica (módulo 18). Como mostrado na figura 31, uma fonte simétrica é uma fonte de corrente contínua que possui duas saídas de tensão de valores iguais, porém, com polaridades invertidas em relação ao terra, ou seja, uma saída positiva e uma negativa.

**Figura 31 – Circuito da fonte simétrica**



**Fonte:** Autoria própria

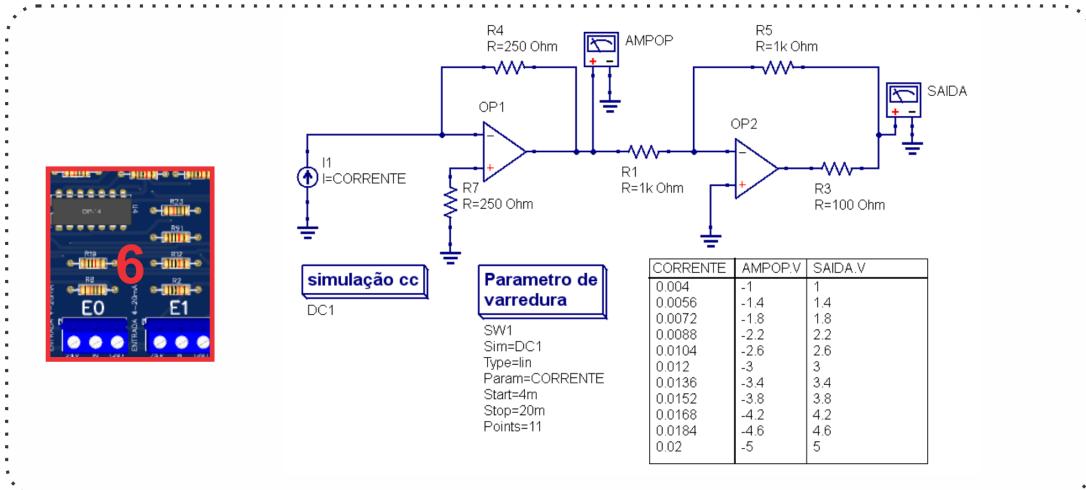
#### 4.0.4 Entrada e saídas E/S analógicas

Finalmente, pode-se entrar no assunto de entradas e saídas I/O começando pelas entradas e saídas analógicas de 4 a 20 mA e 0 a 10 V. Como mostrado na figura 35, usam-se dois amplificadores operacionais para a porta de entrada 4 a 20 mA. O primeiro amplificador configurado em modo inversor é responsável para receber uma corrente de entrada, que varia entre 4 e 20 mA, e convertê-la em um valor de tensão, que nesse caso varia de -1 até -5 V. O segundo amplificador, configurado em modo inversor, será responsável para adequar esse range de -1 até -5 V para um valor compatível do microcontrolador de 1 até 5V.

Pelos resultados mostrados na figura 32, observa-se que uma entrada de  $4mA$  corresponde a uma conversão de  $1V$ , enquanto para uma entrada de  $20mA$  corresponde uma conversão de  $5V$ . Enfim, no projeto final, um diodo Zener será colocado no fim desse subcírcuito como

proteção, evitando assim que por algum problema, como, por exemplo, um curto no amplificador, possa-se elevar a tensão acima de 5V, provocando a queima do microcontrolador.

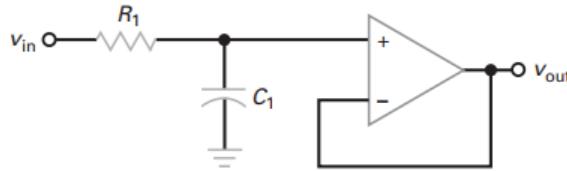
**Figura 32 – Entrada 4 a 20mA simulada com Qucs**



**Fonte:** Autoria própria

No que diz respeito às saídas 4 a 20 mA é preciso salientar que essa família de microcontroladores não possui periféricos de saída *DAC*, mas somente saídas PWM (Pulse Width Modulation) ou Modulação por Largura de Pulso. Um *DAC* (*digital to analog converter*) é um conversor digital-analógico em microcontroladores com esses tipos de periféricos; apenas escreve-se em um registrador o valor binário correspondente à tensão desejada na saída e o conversor trata de colocar a tensão correta no pino de saída (BONI, 2018). Portanto, entende-se que o primeiro passo é converter a onda quadrada do PWM em uma tensão analógica aproximadamente proporcional ao duty-cycle do PWM e para tal fim usa-se um filtro ativo de primeira ordem. Um filtro ativo é um tipo de filtro eletrônico analógico, distinguido dos outros pelo uso de um ou mais componentes ativos associados a elemento passivo. Como mostrado na figura 33, o filtro passa-baixa de primeira ordem de ganho unitário, constituído por um resistor  $R_1$ , um capacitor  $C_1$  e um amplificador operacional em modo buffer, permite a passagem de todas as frequências, desde zero até a frequência de corte, bloqueando todas as outras frequências acima disso (PERTENCE, 2015).

Para calcular esse tipo de filtro ativo  $RC$  deve-se entender a importância de considerar um compromisso entre a ondulação e o tempo de acomodação, pois dois requisitos exigem ações opostas, já que um filtro com pouca ondulação tem uma resposta lenta e um filtro com tempo de acomodação rápido pode ter muita ondulação. Portanto, para atender a esse compromisso assume-se o valor de  $10k\Omega$  para  $R_1$ ,  $1\mu F$  para  $C_1$  e a frequência do *PWM* do microcontrolador

**Figura 33 – Filtro passa baixa de primeira ordem****Fonte:** (PERTENCE, 2015)

é definida na construção deste e pode ser vista na tabela 3.

**Tabela 3 – Frequências do pino PWM**

BOARD	PWM PINS	PWM FREQUENCY
Uno, Nano, Mini	3, 5, 6, 9, 10, 11	490 Hz (pins 5 and 6: 980 Hz)
Mega	2 - 13, 44 - 46	490 Hz (pins 4 and 13: 980 Hz)
Leonardo, Micro, Yún	3, 5, 6, 9, 10, 11, 13	490 Hz (pins 3 and 11: 980 Hz)
Uno WiFi Rev2, Nano Every	3, 5, 6, 9, 10	976 Hz
MKR boards	0 - 8, 10, A3, A4	732 Hz
Zero	3 - 13, A0, A1	732 Hz
Nano 33 IoT	2, 3, 5, 6, 9 - 12, A2, A3, A5	732 Hz
Nano 33 BLE/BLE Sense	1 - 13, A0 - A7	500 Hz
Due	2-13	1000 Hz
101	3, 5, 6, 9	pins 3 and 9: 490 Hz, pins 5 and 6: 980 Hz

**Fonte:** (ARDUINO, 2018)

Com esses parâmetros pode-se calcular a frequência de corte  $f_c$  por meio de equação 2 e o tempo considerando a acomodação de 90% do filtro pela equação 3.

$$f_c = \frac{1}{2\pi R_1 C_1} \quad (2)$$

$$f_c = 15,92 \text{ Hz}$$

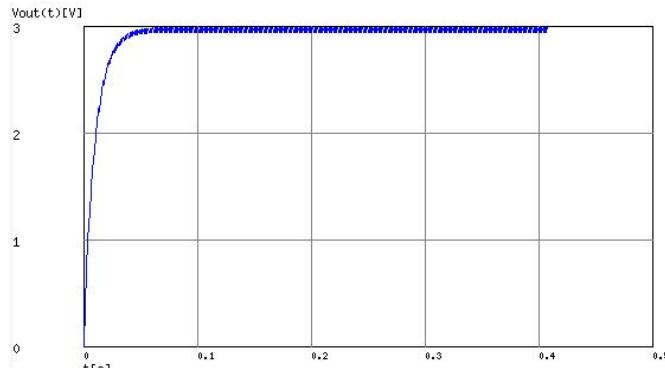
$$t_{RC} = -\ln(1 - 0.9) R_1 C_1 \quad (3)$$

$$t_{RC} = 0.023 \text{ s}$$

Com uma resposta do filtro suficientemente rápida e com uma ondulação tolerável, como mostrado na figura 34, pode-se elaborar a saída analógica 4 a 20 mA. Nesse tipo de saída tem-se um sinal PWM convertido por filtro para uma tensão analógica de 0 a 5V e um amplificador operacional será responsável por converter proporcionalmente a uma corrente de 4 a 20 mA.

Portanto, considerando essas condições de proporção em que  $I_L = 4mA$  com  $V_1 = 1V$ ,  $I_{L+} = 20mA$  com  $V_1 = 5V$  e assumido  $R_1 = R_2 = R_3 = R_4$  pode-se calcular o subcircuito

**Figura 34 – Resposta do filtro passa baixa**



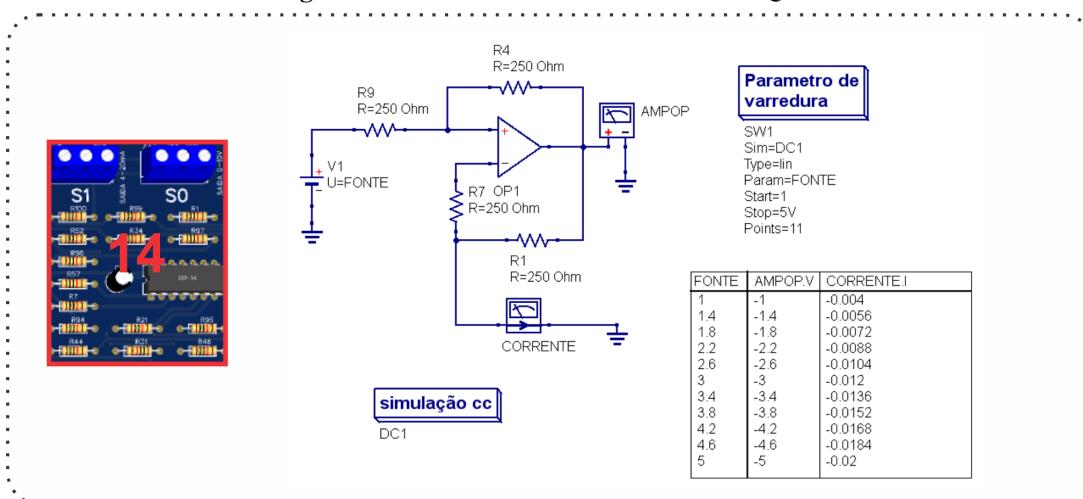
Fonte: autoria própria

mostrado na figura 35 por meio da equação 4,

$$\begin{aligned}
 I_L &= \frac{(V_1)}{R_1} \\
 I_L R_1 &= V_1 - V_2 \\
 4e - 3 * R_1 &= 1 \\
 20e - 3 * R_1 &= 5
 \end{aligned} \tag{4}$$

obtendo  $R_1 = R_2 = R_3 = R_4 = 250\Omega$ .

**Figura 35 – Saída 4 a 20mA simulada com Qucs.**



Fonte: Autoria própria

Nas saídas analógicas de 0 a 10 V tem-se uma variação de tensão que varia entre 0 e 10 V, portanto, desenvolveu-se um circuito que tem uma variação diretamente proporcional entre a saída definida e a saída do pino do microcontrolador, a qual varia entre 0V e 5V. Assim sendo, cria-se um circuito contendo um amplificador operacional, como mostrado na figura 36, o qual

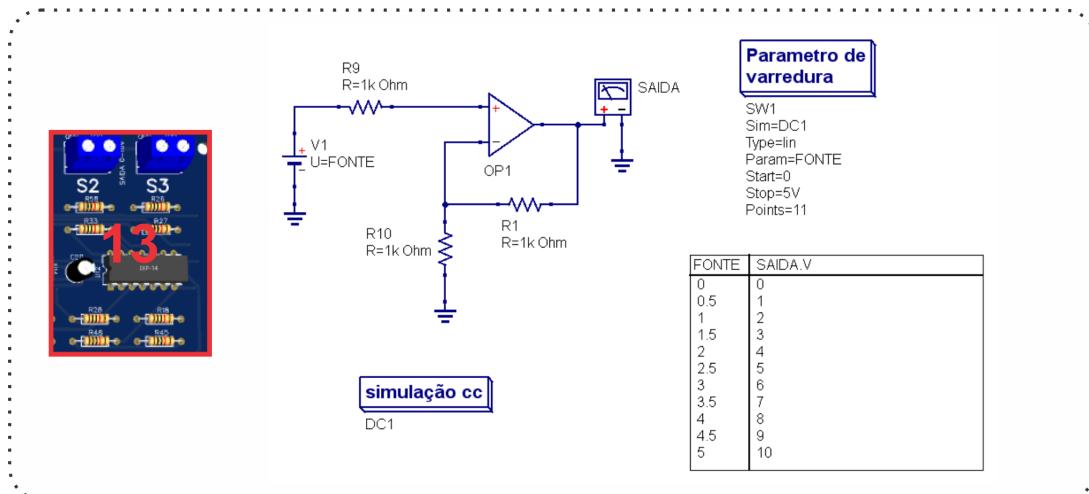
será configurado como filtro passa-baixa ativo não inversor. Esse amplificador será responsável por converter o sinal PWM do microcontrolador em sinal de tensão analógico e por manter a proporção da tensão entre a saída do dispositivo e a saída do microcontrolador. Logo, por meio da equação 5 e assumindo-se  $R_1 = 1k$ , calcula-se  $R_2$ ,

$$V_{out} = \left(1 + \frac{R_2}{R_1}\right) V_{in}$$

$$R_2 = \left(\frac{V_{out}}{V_{in}} - 1\right) R_1 \quad (5)$$

sendo que se obtém o valor  $R_2 = R_1$ .

**Figura 36 – Saída 0 a 10V simulada com Qucs**



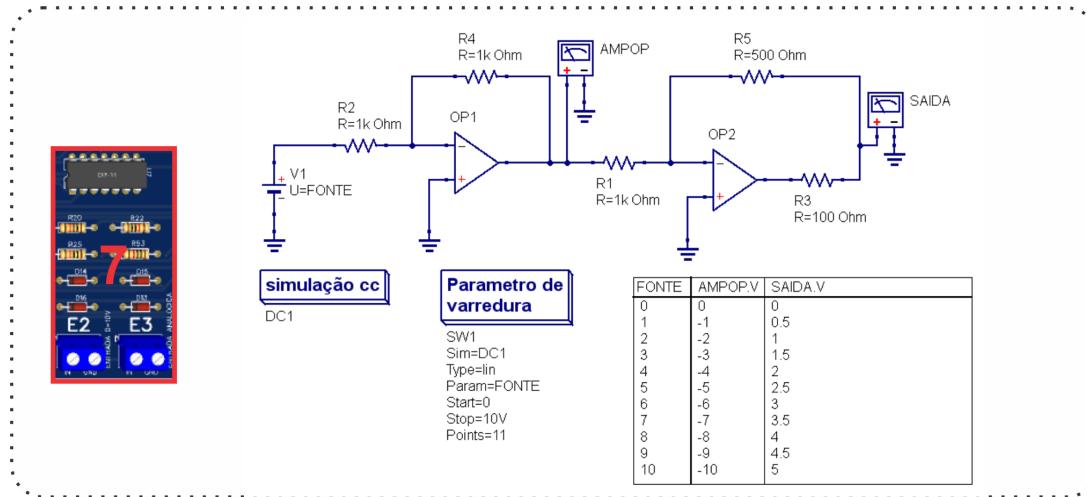
**Fonte:** Autoria própria

Na entrada analógica 0 a 10 V tem-se uma variação de tensão que vai de 0 e 10 V, desenvolveu-se um circuito que tem uma variação diretamente proporcional entre a entrada definida e a entrada do pino do microcontrolador, a qual varia entre 0 e 5V. Assim sendo, cria-se um circuito contendo dois amplificadores operacionais configurados em modo inverso, como mostrado na figura 37. O primeiro amplificador operacional será responsável por fazer a isolação e inverter a tensão, enquanto o segundo será responsável pela proporção, que será calculada por meio da equação 6, e assumindo-se os valores  $R_1 = R_2 = R_3 = 1k$ , calcula-se  $R_4$ . Por fim, um diodo Zener será colocado para proteger o microcontrolador de sobretensão.

$$V_{out} = -\frac{R_4}{R_3} \cdot V_{in}$$

$$R_4 = -\frac{5.1000}{10} \quad (6)$$

**Figura 37 – Entrada 0 a 10V simulada com Qucs**



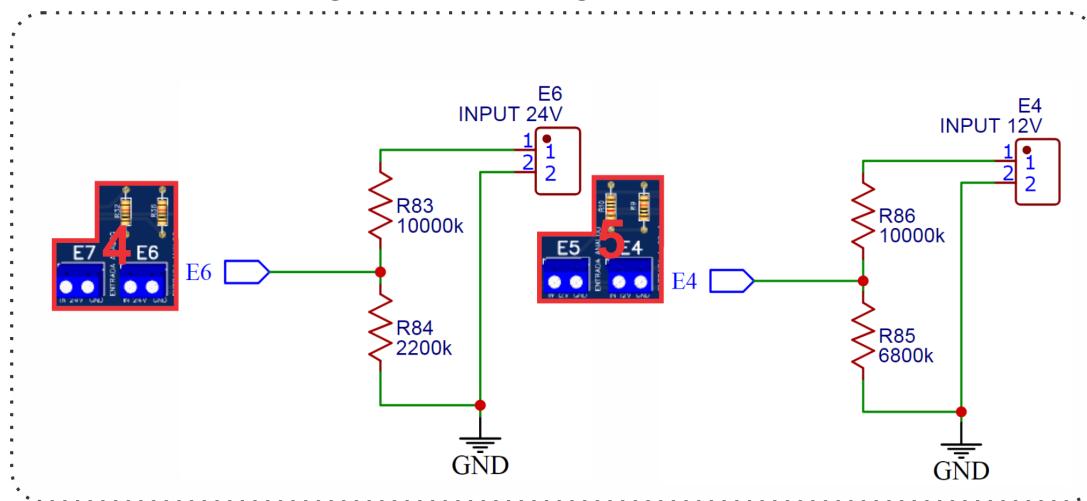
**Fonte: Autoria própria**

Obtem-se  $R_4 = 500\Omega$ , assim, a saída será de 5V quando teremos na entrada 10V.

#### 4.0.5 Entrada e saídas E/S digitais

Uma entrada digital consiste tipicamente numa fonte de tensão, isto é um interruptor que dependendo do estado aberto/fechado, onde o microcontrolador detecta um estado de tensão ou tensão 0, interpretando-o como um estado lógico de "1"ou "0". Neste protótipo propõe-se entradas de 0-24V e 0-12V. Estes circuitos, tem a tarefa de baixar a tensão em nível do microcontrolador, neste caso em 5V.

**Figura 38 – Entradas digitais 0-12V e 0-24V**

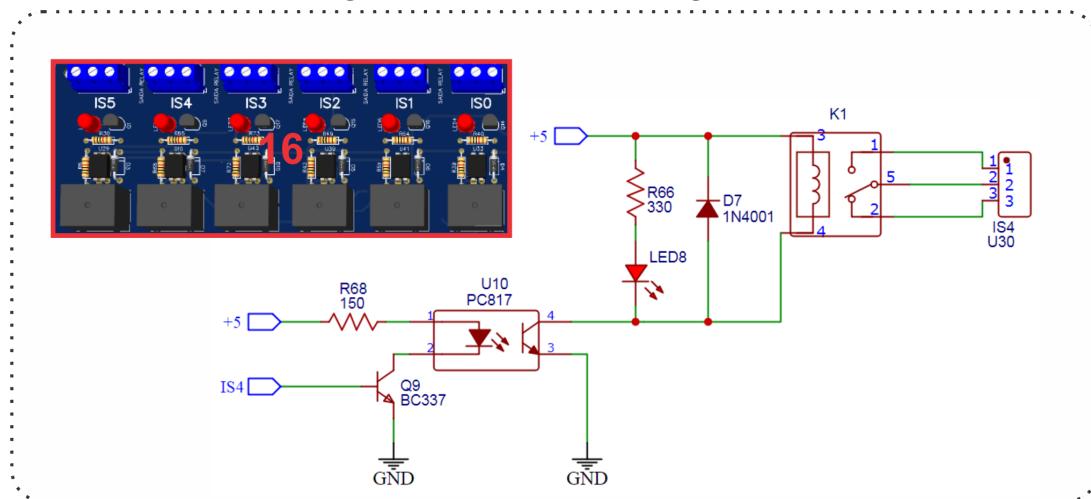


**Fonte: Autoria própria**

No EDUARD tem-se saídas com relé e transistores. Os relés, por serem elementos

eletromecânicos, estão sujeitos a limitações como desgaste dos contatos e velocidade de comutação, mas podem trabalhar em uma ampla faixa de tensão e corrente. É comum encontrar saídas digitais de 250 V AC/DC de 2 a 10 A, por exemplo, o que diminui a necessidade de circuitos auxiliares. Como mostrado na figura 39, o pino do microcontrolador da saída digital relé é isolada por um optoacoplador e um diodo em antiparalelo é incluído no circuito para evitar que um pico de tensão possa causar um arco nos contatos da chave ou destruir os transistores da chave quando a bobina é desligada.

**Figura 39 – Interface da saída digital relé**



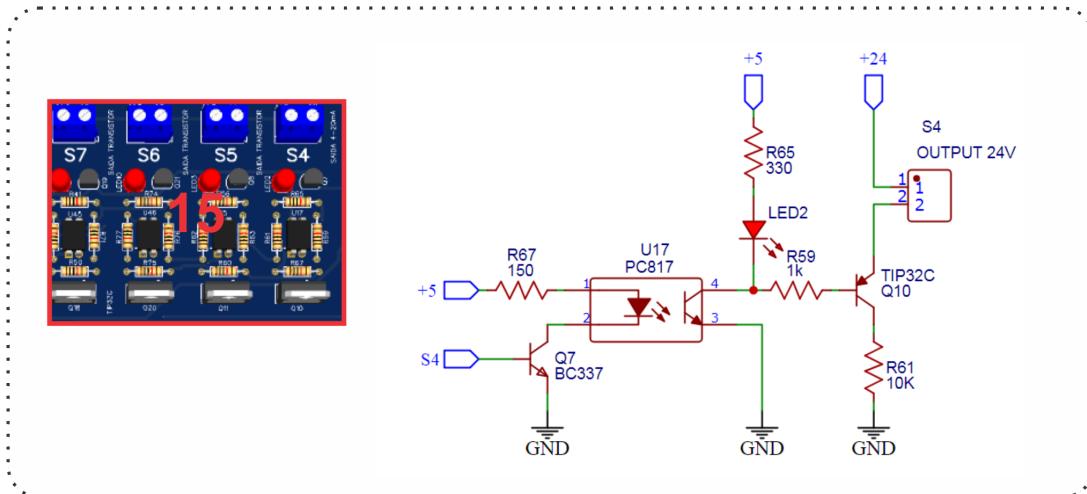
**Fonte: Autoria própria**

A saída transistor, mostrada na figura 40, utiliza esse componente estático porque possui maior vida útil que os relés. Essa saída trabalha apenas com corrente contínua, geralmente com tensão de 24 VDC, e comuta correntes de baixa amplitude como, por exemplo, 500 mA na maioria dos casos. Em alguns casos é possível a utilização de mais de um canal para uma mesma carga para aumentar a corrente máxima. Por exemplo, é possível utilizar dois canais de 0,5 A simultaneamente para acionar uma carga de 1 A. Nesse caso, o acionamento dos canais deve ser simultâneo para evitar sobrecargas em um dos canais.

#### 4.1 AQUECEDORES CONTROLADOS

Dentro o protótipo proposto propõe-se um circuito didático elaborado pela APMonitor.com: o BYU Arduino Temperature Control Lab, figura 41. Esta shield para Arduino simples e econômica, foi projetada para atender cursos práticos de controle de processo. A saída de calor produzida por um transistor é ajustada modulando a tensão na base deste. O calor produzido

**Figura 40 – Interface saída Transistor 24V**



**Fonte:** Autoria própria

é transferido por convecção para um sensor de temperatura. O calor deste sensor é lido pela plataforma Arduino que regula uma saída PWM, a qual fornece tensão na base do transistor. Este pequeno circuito permite criar várias experiências de controle de sintonia por PID, MPC, entre outras.

**Figura 41 – BYU Arduino Temperature Control Lab**

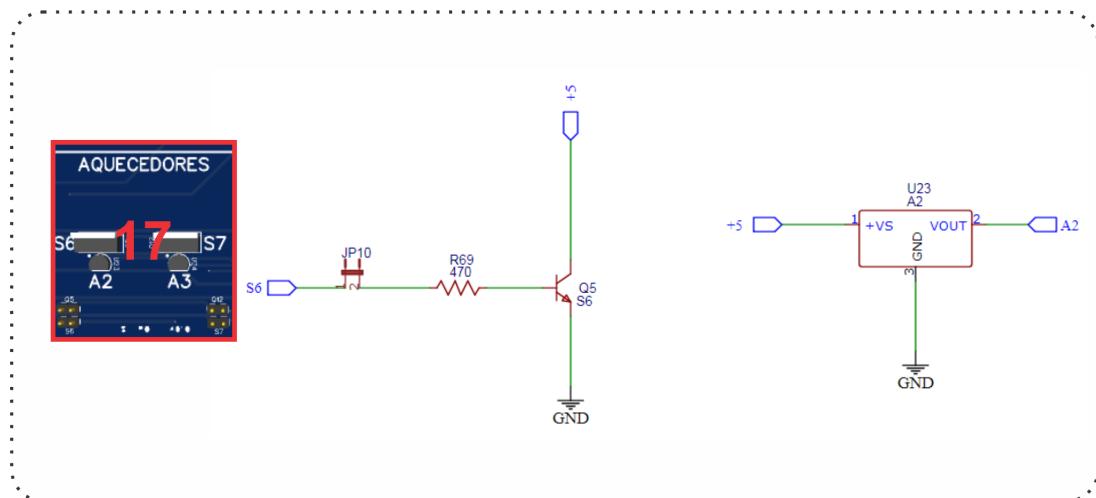


**Fonte:** APMonitor.com

Na figura 42 é mostrado o circuito a ser integrado no protótipo, qual faz uso do transistor TIP31C e do sensor de temperatura LM35:

#### 4.2 CONCLUSÕES DO CAPÍTULO 4

Neste capítulo foi apresentado o protótipo EDUARD, uma plataforma que recebe, por encaixe, o Arduino Uno R3. EDUARD foi concebido por arquitetura de módulos. Esquemático

**Figura 42 – Circuito dos aquecedores**

**Fonte: Autoria Própria**

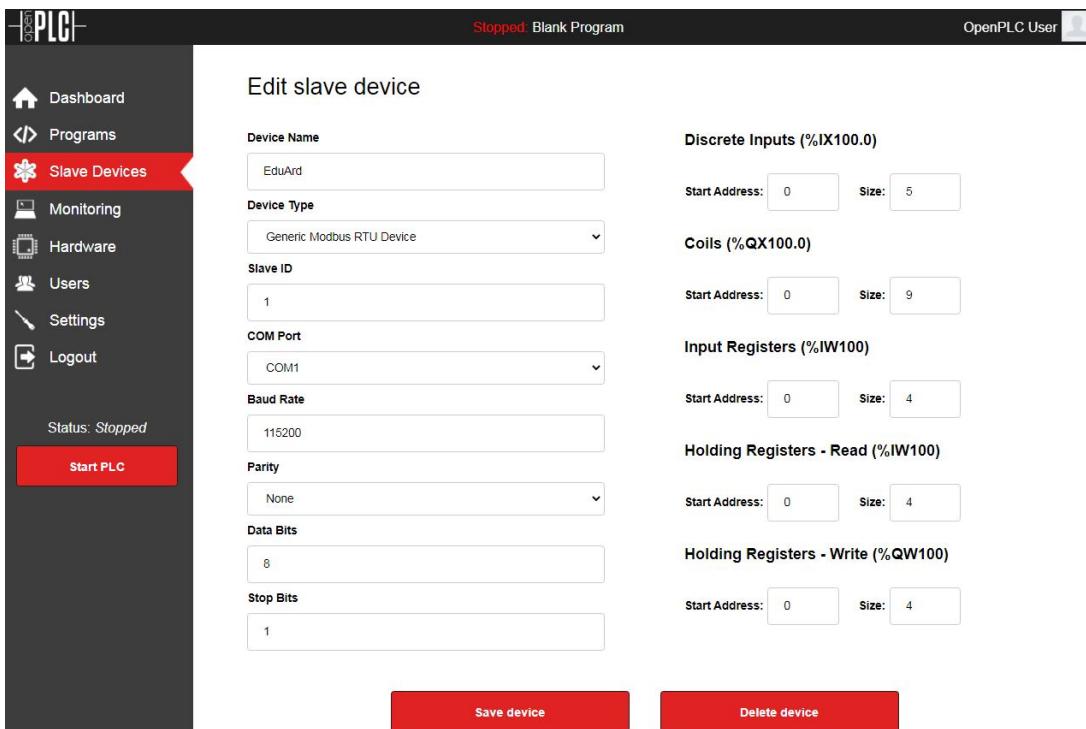
eletrônicos de módulos, reproduzindo portas de entradas e saídas, foram explanado conforme explicado na seção 3.1.3, que trata de módulos E/S.

## 5 FERRAMENTA SOFTWARE DO SISTEMA

### 5.1 OPENPLC

O OpenPLC é um projeto de controlador em software que utiliza um servidor web local baseado em NodeJs, o qual é estável em plataformas linux ou windows. Além do servidor web, faz parte do projeto um editor, o OpenPLC Editor, que atende as principais linguagens definidas conforme a norma IEC 61131-3, e que estabelece a arquitetura básica de software e as linguagens de programação para CLPs. Dentre as linguagens suportadas pelo OpenPLC, estão: LD (Ladder Diagram ou Diagrama Ladder), FBD (Function Block Diagram ou Diagramas de Blocos Funcionais), ST (Structured Text ou Texto Estruturado), IL (Instruction List ou Lista de Instruções) e SFC (Sequential Function Chart ou Sequenciamento Gráfico de Funções). Esse software exporta o código produzido em um arquivo com extensão .st, o qual será instalado no servidor web. De acordo com as indicações do site do desenvolvedor ([openplcproject.com](http://openplcproject.com)) faz-se

**Figura 43 – Interface configuração slave device OpenPLC**



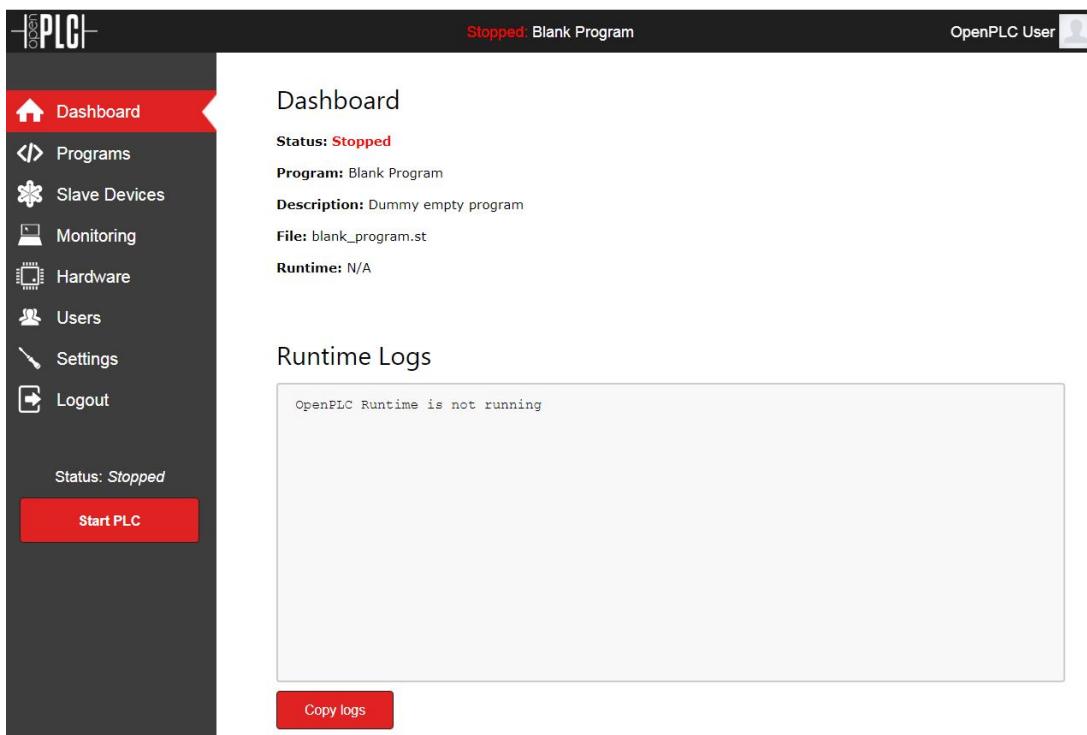
**Fonte:** Autoria própria

a instalação o servidor. Uma vez instalado procede-se a configuração por meio da interface web. O sistema dispõe de várias páginas, sendo as mais importantes a Slave Device onde pode-se

configurar o dispositivo a ser controlado, como mostrado na figura 43 e a Programs onde pode ser feito o upload do arquivo .st para gerenciamento do controlador, como mostrado na figura 43.

Enfim, uma vez completadas essas configurações, pode-se ativar o sistema por meio do botão Start PLC presente na página Dashboard, figura 44, iniciando assim o controle do PLC por comunicação Modbus.

**Figura 44 – Interface Dashboard OpenPLC**



**Fonte:** Autoria própria

## 5.2 SCADABR

Assim como o software anterior, o ScadaBR oferece uma interface web para a visualização das variáveis, gráficos, estatísticas, configuração dos protocolos, alarmes, construção de telas tipo IHM e uma série de opções de configuração. Após configurar os protocolos de comunicação com os equipamentos e definir as variáveis (entradas e saídas, ou "tags") de uma aplicação automatizada é possível montar interfaces de operador web utilizando o próprio navegador. Também é possível criar aplicativos personalizados, em qualquer linguagem de programação, a partir do código-fonte disponibilizado ou de sua API "web-services"(GOMES, 2015). Algumas funcionalidades do sistema:

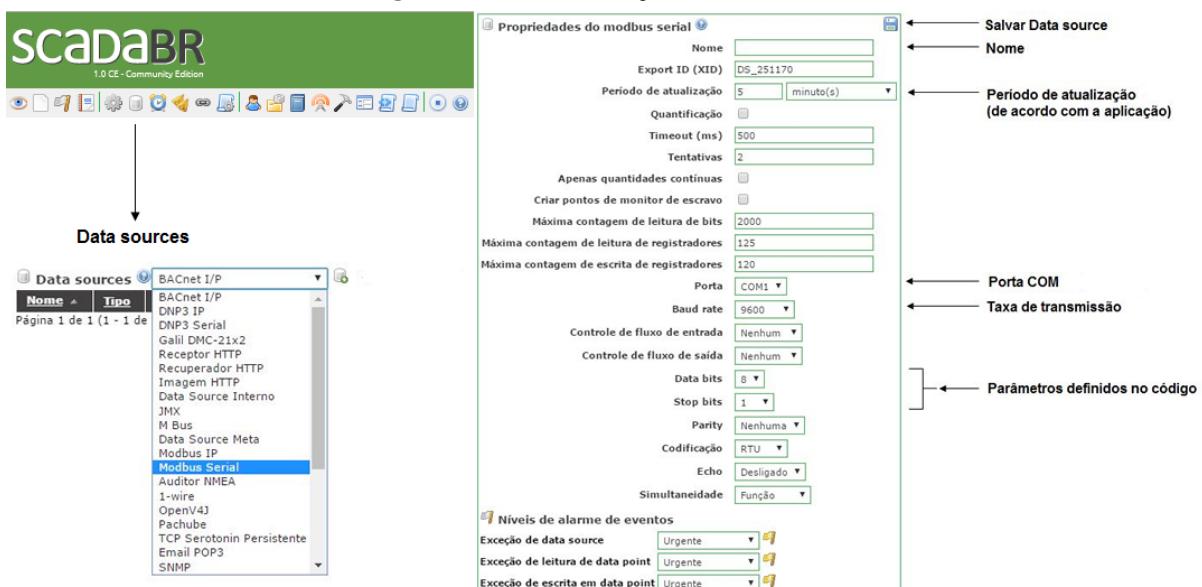
- Interface do usuário baseada na web para monitoramento remoto do sistema por meio de

displays numéricicos e gráficos;

- Registro histórico das variáveis medidas com geração de relatórios personalizáveis;
- Detecção de ocorrências de alarmes, com registro em base de dados e emissão de alertas visuais ou sonoros, com possibilidade de envio de mensagens via e-mail ou celular;
- Drivers de aquisição de dados compatíveis com Linux e Windows para utilização com os protocolos mais difundidos atualmente na indústria (por exemplo, Modbus e OPC);
- Wrappers (interfaces de programação) para linguagens de alto nível como PHP e C, permitindo desenvolvimento rápido de aplicações personalizadas.

Nesta seção aborda-se a configuração da estrutura de comunicação entre o ScadaBR e o protótipo. Primeiramente, para que o ScadaBR identifique o elemento com o qual irá ser conectado, deve-se criar uma instância chamada data source, cuja função é definir onde está o dispositivo externo e os parâmetros do tipo de comunicação utilizada, figura 45. Nessa tela deve-se configurar pelo menos os parâmetros básicos, tais como o período de atualização dos dados, a porta de comunicação COM utilizada na conexão entre o supervisório e o dispositivo junto à relativa taxa de transmissão Baudrate, a quantidade de bits de dados e de parada e o tipo de codificação.

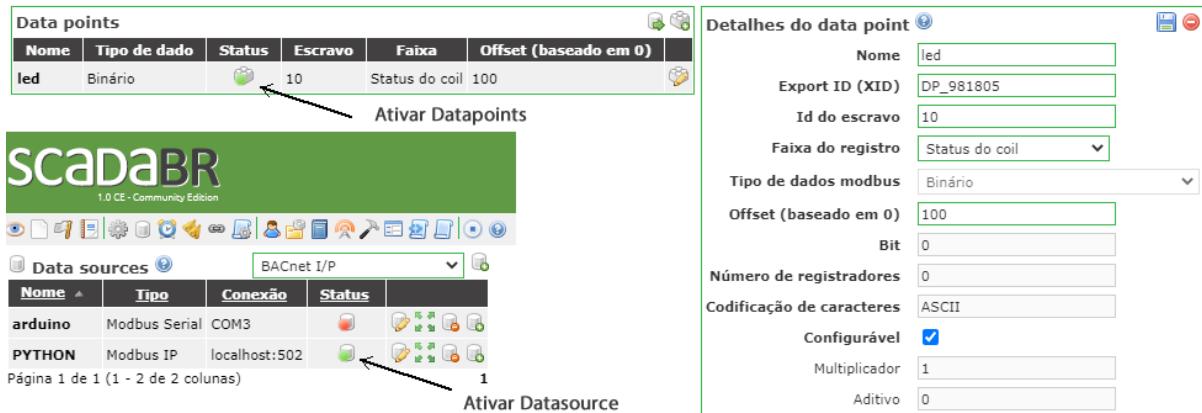
**Figura 45 – Tela de criação do Data Source**



**Fonte:** Autoria própria

Após a criação e configuração do Data source, deve-se criar a denominada Data point. Os Data points são os elementos que irão conter os valores relacionados com as entradas e

**Figura 46 – Tela configuração do Data Source**



**Fonte:** Autoria própria

saídas da plataforma conectada. Como mostrado na figura 46 deve-se nomear o Data point e define-se os 2 campos fundamentais: o primeiro é o ID do escravo e o segundo é a Faixa do registro. Sucessivamente, os data points devem ser criados inserindo-os registradores holding e definindo três campos fundamentais: o primeiro é o faixa do registro, em que a opção a ser selecionada deve ser registrador holding; em seguida, o campo tipo de dados MODBUS deve ser determinado como binário; por último, o campo offset – este campo informa ao ScadaBR qual dos registradores holding será escrito ou lido, de modo que o número a ser utilizado vai de acordo com a ordem de escrita dos registradores holding no código do dispositivo, como mostrado na figura 47.

### 5.3 SOFTWARES AUXILIARES

Ambientes virtuais são de suma importância no meio acadêmico, pois permitem o melhor entendimento dos problemas por meio das simulações. Para isso, ferramentas computacionais de cálculo e modelagem são indispensáveis, auxiliando e facilitando as tarefas (RIOS, 2006). Porém, muitos softwares possuem um alto custo de aquisição e por isso deixam de ser utilizados devido à falta de recursos das universidades ou são utilizados ilegalmente. Nesse contexto, este trabalho visa a contornar o problema adotando a utilização de algumas ferramentas open-source, mostradas na tabela 4.

VS Code (*Visual Studio Code*) é um editor de código-fonte desenvolvido pela Microsoft para Windows, Linux e macOS. Neste trabalho foi usado com a extensão Python. A linguagem de programação Python é uma multiplataforma e uma linguagem interpretada, ou seja, o código-fonte é interpretado por uma máquina virtual durante a execução do programa. É uma linguagem

**Figura 47 – Tela configuração do Data Points**

The figure shows two windows side-by-side. On the left is a table titled 'Data points' with columns: Nome, Tipo de dado, Status, Escravo, Faixa, and Offset (baseado em 0). The table lists 22 entries, mostly 'Numérico' type, with values ranging from F0 to X2. On the right is a detailed configuration dialog for a specific point named 'V1'. The dialog fields include: Nome (V1), Export ID (XID) (DP\_208151), Id do escravo (1), Faixa do registro (Registrador holding), Tipo de dados modbus (Inteiro de 2 bytes sem sinal), Offset (baseado em 0) (1000), Bit (0), Número de registradores (0), Codificação de caracteres (ASCII), Configurável (unchecked), Multiplicador (0.01), and Aditivo (0).

**Fonte:** Autoria própria

**Tabela 4 – Ferramentas open-source neste trabalho**

Nome do software	Descrição
VScode	ambiente de desenvolvimento multi-linguagem
PlatformIO	extensão para VSCode para microcontroladores
Python	extensão para VSCode para ambiente Python
Qucs	ambiente para simulação de circuitos eletrônicos

**Fonte:** Autoria própria

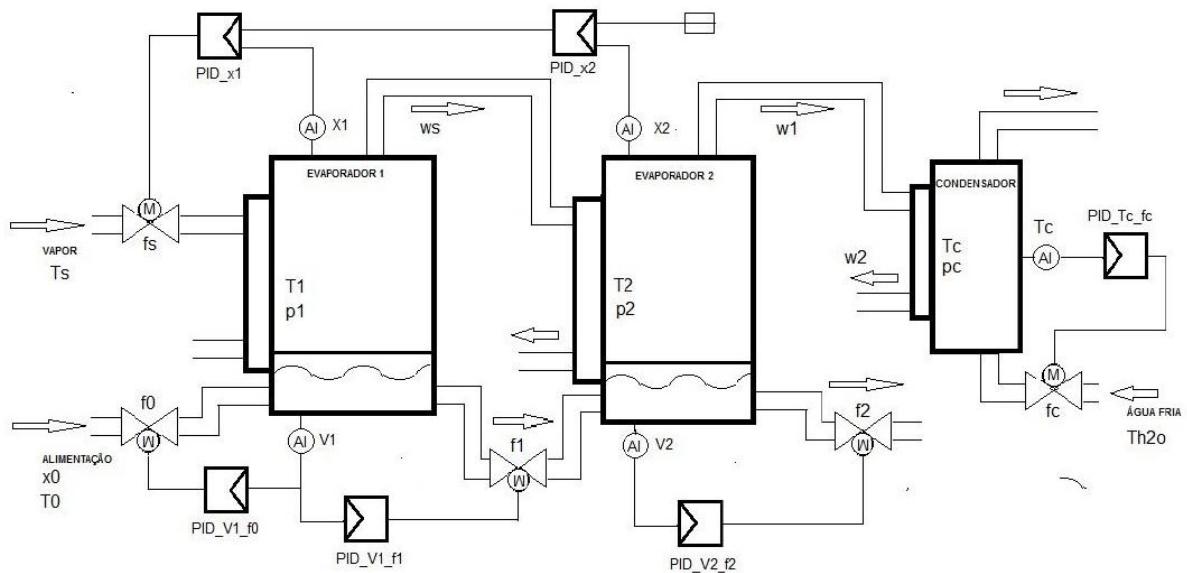
de programação orientada a objetos, o que significa que ela possui recursos, classes e objetos, por exemplo, que dão suporte ao paradigma de programação orientada a objetos (PYTHON, 2001). Também foi instalada a extensão PlatformIO, que é um framework baseado na classe de Arduino que permite escrever software em microcontroladores para controlar dispositivos conectados ao módulo Arduino (PLATFORMIO, 2014). Nesta configuração o VSCode permite escrever e executar duas linguagens diferentes contemporaneamente como C++ e Python.

O Qucs (*Quite Universal Circuit Simulator*) consiste na junção de vários softwares de simulação de circuitos eletrônicos. O software é capaz de fazer análises que incluem parâmetro S (incluindo ruído), CA (incluindo ruído), CC, análise transiente, equilíbrio harmônico (ainda não concluído), simulação digital (VHDL e Verilog-HDL) e varreduras de parâmetro (MARGRAF; JAHN, 2003).

## 5.4 PLANTA DIDÁTICA

Neste trabalho usa-se um modelo de planta industrial para fazer testes com o protótipo hardware, programado em forma de classe com o propósito de poder instanciar mais de um objetos, mantendo uma estrutura simples para refatoração. O modelo em questão simula um processo por evaporação em múltiplos estágios que conjuga em série dois evaporadores de um estágio, com o objetivo de concentrar uma solução pela ebulação do solvente (FOUST; WENZEL, 1982). Nesse tipo de processo existe uma ligação entre os evaporadores de tal modo que o vapor produzido em um estágio do evaporador serve como fluido de aquecimento para o seguinte estágio e assim sucessivamente até o último estágio. Como mostrado na figura 48, o processo é de tipo alimentação direta, quando a alimentação é feita no estágio mais quente e percorre os estágios no mesmo sentido do fluxo de vapor, sendo utilizada quando a temperatura de alimentação é alta. Nesse processo, com a entrada de uma alimentação busca-se produzir uma solução mais concentrada. A alimentação, aqui, trata-se de uma solução (licor negro e água) com concentração  $x_0$  de 20%, temperatura  $T_0$  de 90 °C e vazão  $f_0$ , cujo valor inicial é de 0,11 m<sup>3</sup>/s, que alimenta o evaporador 1. A evaporação é realizada dentro do evaporador 1, que funciona como um trocador de calor aquecendo a solução até a ebulação, passando em seguida para um segundo evaporador que repete o mesmo processo.

**Figura 48 – Sistema de evaporação com 2 estágios**



**Fonte:** (SOUZA, 2016)

O evaporador 1 possui temperatura  $T_1$ , pressão  $p_1$ , fluxo de energia ( $q_1$ ), volume  $V_1$  de aproximadamente 5 m<sup>3</sup>, com área de transferência de 2.900 m<sup>2</sup> e coeficiente de calor de 4

kW/m<sup>2</sup> °C. É aquecido com vapor através da válvula (fs) a uma temperatura saturada do vapor (Ts) de 140 °C. A saída de vapor do evaporador 1 é responsável pelo aquecimento do evaporador 2 através de ws. Dentro desse primeiro evaporador, a solução passa a ter concentração x1 e é liberada para o evaporador 2 através da válvula de saída (f1). O evaporador 2 recebe aquecimento do evaporador 1 (ws), possui temperatura T2, pressão p2, fluxo de energia (q2), volume V2 aproximado de 5 m<sup>3</sup>, com área de transferência de 3.000 m<sup>2</sup> e coeficiente de calor de 4 kW/m<sup>2</sup> °C.

Dentro do segundo evaporador, com valor inicial de vazão da válvula de saída (f2) de 0,057 m<sup>3</sup>/s, a concentração passa a ser x2. Após aquecimento dos dois evaporadores, o vapor de saída do evaporador 2 (w1) é transferido para um condensador que possui temperatura Tc, fluxo de energia (qc), área de transferência de calor igual a 3.000 m<sup>2</sup> e coeficiente de transferência de calor igual a 5 kW/m<sup>2</sup> °C. A temperatura de água fria (Th2o) na entrada do condensador é de aproximadamente 20 °C; a entrada dessa água é controlada pela válvula de entrada do condensador (fc) (SOUZA, 2016). As equações 7 a 20 representam o modelo matemático do processo industrial descrito acima. O modelo foi baseado no trabalho de Granfors e Nilsson (1999).

$$\frac{d(\rho V)}{dt} = \rho_{in}F_{in} - \rho_{out}F_{out} - W \quad (7)$$

$$\frac{d(X_w\rho V)}{dt} = X_{win}\rho_{in}F_{in} - X_{wout}\rho_{out}F_{out} - W \quad (8)$$

$$\frac{d(X_s\rho V)}{dt} = X_{sin}\rho_{in}F_{in} - X_{sout}\rho_{out}F_{out} \quad (9)$$

$$\frac{d(\rho VH)}{dt} = \rho_{in}F_{in}H_{in} + Q - \rho_{out}F_{out}H_{out} - WH_{out}^V \quad (10)$$

$$\frac{d(X_w\rho V)}{dt} = X_w \frac{d(\rho V)}{dt} + \rho V \frac{d(X_w)}{dt} \quad (11)$$

$$\frac{d(X_w)}{dt} = \frac{1}{\rho V} [\rho_{in}F_{in}(X_{win} - X_{wout}) - W(1 - X_{wout})] \quad (12)$$

$$\frac{d(X_s\rho V)}{dt} = X_s \frac{d(\rho V)}{dt} + \rho V \frac{d(X_s)}{dt} \quad (13)$$

$$\frac{d(X_s)}{dt} = \frac{\rho_{in} F_{in}}{\rho V (X_{sin} - X_{sout})} \quad (14)$$

$$\frac{d(\rho V H)}{dt} = H \frac{d(\rho V)}{dt} + \rho V \frac{d(H)}{dt} \quad (15)$$

$$\frac{d(H)}{dt} = \frac{1}{\rho V} [\rho_{in} F_{in} (H_{in} - H_{out}) - W (H_{out}^V - H_{out}) + Q] \quad (16)$$

$$Q = kA(T_{out} - T_{cold}) \quad (17)$$

$$Q_c = W \Delta H_{steam} \quad (18)$$

$$Q_c = \rho_c C_{pc} (T_{cin} - T_{cout}) \quad (19)$$

$$Q_c = k_c A_c (T - T_{cout}) \quad (20)$$

De acordo com Granfors e Nilsson (1999), para a simulação do processo, além das equações vistas, precisa-se considerar também outras rotinas de cálculo representadas as seguir como funções:

- dnslblq (T,X): Densidade como uma função de temperatura e concentração do licor negro;
- entlblq (T,X): Entalpia como uma função de temperatura e concentração do licor negro;
- tlblq (X,H): Temperatura como uma função de concentração e entalpia do licor negro;
- eqpblk (T,X): Equilíbrio da pressão como uma função de temperatura e concentração do licor negro;
- eqth2o (P): Equilíbrio de temperatura como uma função de pressão;
- eqph2o (T): Equilíbrio da pressão como uma função de temperatura;
- epvh2o (P,T): Entalpia do vapor puro como uma função de pressão e temperatura;
- eplh2o (P,T): Entalpia do líquido puro como uma função de pressão e temperatura,

onde:

$\rho$ : densidade ( $\text{kg/m}^3$ );

$V$ : volume ( $\text{m}^3$ );

$F$ : vazão volumétrica ( $\text{m}^3/\text{s}$ );

$W$ : massa de vazão da saída do evaporador 2 ( $\text{kg/s}$ )

$X$ : concentração ( $\text{kgH}_2\text{O/kg}$ );

$H$ : entalpia ( $\text{kJ/kg}$ );

$Q$ : fluxo de energia ( $\text{kJ/s}$ );

$A$ : área de transferência de calor ( $\text{m}^2$ );

$k$ : coeficiente de transferência de calor ( $\text{kJ/m}^2\text{°C}$ );

$T$ : temperatura ( $^\circ\text{C}$ );

$C$ : condensador;

$\Delta$ : diferença;

$C_p$ : é Calor específico ( $\text{kJ/kg}^\circ\text{C}$ );

$e$ : erro de controle;

$K_c$ : ganho do controlador;

$m$ : massa de vazão da água fria ( $\text{kg/s}$ );

$P$ : pressão ( $\text{kPa}$ );

$u$ : é o sinal de controle ( $\text{m}^3/\text{s}$ );

$LN$ : média logarítmica da temperatura;

$L$ : líquido denominado licor negro;

$V$ : vapor;

$IN$  dentro;

*OUT* para;

*w*: água;

*s*: licor;

*hot*: quente;

*cold*: frio;

Tendo o modelo matemático da planta definido, pode-se elaborar as estratégias de controle para os ensaios:

1. A variável de processo  $x1$  representa a concentração no evaporador 1 e a variável manipulada  $f_s$  representa a válvula de vapor. Essa estratégia, por meio de um controle cascata, visa controlar a abertura da válvula  $f_s$  para manter o valor de concentração no evaporador 2 ( $x2$ ); essa variável  $x2$  tem setpoint próprio, enquanto a saída do PID de  $x2$  será o setpoint do  $x1$ ;
2. Variável de processo  $V1$  representa o volume no evaporador 1 e variável manipulada  $f_0$  representa a válvula de alimentação1. Essa estratégia visa controlar a abertura da válvula  $f_0$ , para manter o volume do evaporador 1 ( $V1$ );
3. Variável de processo  $V1$  e variável manipulada  $f_1$ , que representa a válvula de vazão/alimentação entre os evaporadores 1 e 2. Essa estratégia visa controlar a abertura da válvula  $f_1$  para manter o volume do evaporador 1 ( $V1$ );
4. A variável de processo  $V2$  representa o volume no evaporador 2 e a variável manipulada  $f_2$  representa a válvula de vazão do evaporador 2. Essa estratégia visa controlar a abertura da válvula  $f_2$  para manter o volume do evaporador 2 ( $V2$ );
5. A variável de processo  $Tc$  representa a temperatura do condensador e a variável manipulada  $f_c$  representa a válvula de alimentação da água fria. Essa estratégia visa controlar a abertura da válvula  $f_c$  para manter a temperatura do  $Tc$ .

## 5.5 O SOFTWARE DE GERENCIAMENTO DO SISTEMA

Neste contexto utiliza-se a linguagem de programação Python para elaborar o código que será responsável pelo gerenciamento da planta, pelo gerenciamento do controlador, pela

comunicação com os dispositivos e pela impressão dos gráficos. Para que esse código se torne de fácil manipulação ou refatoração foram criadas duas classes (Evaporadores.py e Controlador.py), contendo as funções necessárias. Com essas classes prontas, além de permitir uma redução drástica de tempo para criar o código de gerenciamento pode-se instanciar objetos da planta ou do controlador quantas vezes for necessário, permitindo ensaios paralelos e comparações de resultados. No caso da classe do controlador, quando o dispositivo desenvolvido for conectado pode-se mudar os parâmetros de controle sem precisar reprogramar o microcontrolador, permitindo mais dinâmica na execução das experiências acadêmicas. Por fim, no código central, responsável para as chamadas das classes, os objetos serão instanciados respeitando os parâmetros mostrados na tabelas 5, 6 e 7

**Tabela 5 – Receita dos sensores da planta**

<b>DESCRIÇÃO</b>	<b>VARIABEL</b>	<b>REGIME INICIAL</b>	<b>SETPOINTS</b>
Volume evaporador 1	V1	5	3
Volume evaporador 2	V2	5	3
Concentração evaporador 1	X1	0,25	
Concentração evaporador 2	X2	0,34	0,44
Temperatura condensador	TC		70

**Fonte:** Autoria própria

**Tabela 6 – Receita dos atuadores da planta**

<b>DESCRIÇÃO</b>	<b>VARIABEL</b>	<b>REGIME INICIAL</b>	<b>RANGE</b>
Válvula alimentação evaporador 1	F0	0,11	0-5
Válvula vazão/alimentação evaporador 1 e 2	F1	0,822	0-5
Válvula alimentação evaporador 2	F2	0,577	0-5
Válvula vazão condensador	Fc	0,2139	0-5
Válvula vapor	Fs	5	0-5

**Fonte:** Autoria própria

**Tabela 7 – Receita das perturbações da planta**

<b>DESCRIÇÃO</b>	<b>VARIABEL</b>	<b>REGIME INICIAL</b>
Fluxo entrada liquor negro	X0	0,2
Temperatura liquor negro	T0	60
Temperatura da agua fria no condensador	Tc	20
Temperatura vapor	Ts	140

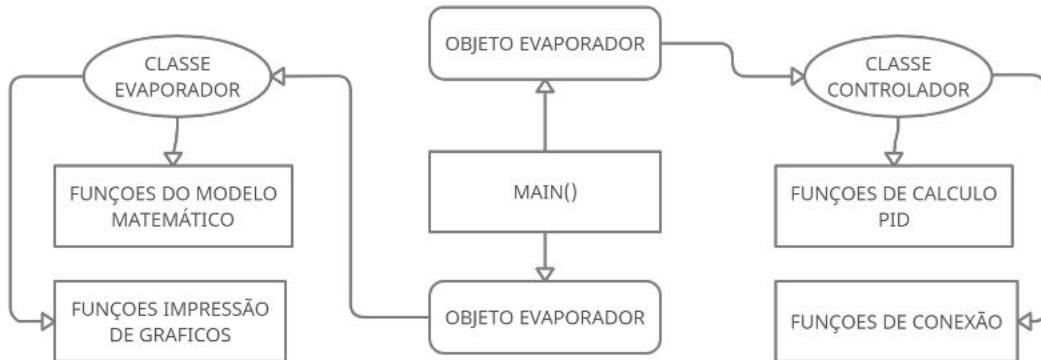
**Fonte:** Autoria própria

## 6 RESULTADOS E DISCUSSÃO

### 6.1 ENSAIO VIRTUAL

Neste capítulo explanam-se três ensaios com o objetivo de validar os resultados obtidos com o protótipo hardware. Na primeira experiência é feito um controle inteiramente virtual, em que o programa em Python resolve todo o cálculo da modelagem da planta e faz o controle PID, sendo o código desenvolvido por meio de orientação a objeto, isto é, um código instanciando o objeto da classe evaporador, que contém todas as funções de cálculo do modelo matemático da planta, e o objeto da classe Controlador.py, que contém todas as funções de cálculo do controlador PID, como mostrado na figura 49.

**Figura 49 – Diagrama de blocos do código Python**



**Fonte:** Autoria própria

Utilizam-se os resultados gráficos desse primeiro ensaio como comparação aos resultados obtidos no segundo ensaio. Observa-se que no diagrama de blocos da figura 49 existem funções de conexão que permitem que o Python converse com o protótipo hardware desenvolvido. Os gráficos a seguir mostram os resultados do controle de volume, a concentração do licor negro, temperatura do vapor nos 2 evaporadores e temperatura do fluido no condensador. Para simulação do comportamento dinâmico do processo foi elaborado uma receita, representada na tabela 8.

Pode-se observar pelo gráfico 1 que o volume V1 no evaporador 1 do licor negro se manteve constante até a etapa 5000, quando o valor do setpoint foi mudado para  $4\text{ m}^3$ . Também é evidenciada uma oscilação no volume neste evaporador na etapa 8000 devido a mudança do setpoint do volume, V2, do evaporador 2 para  $3,8\text{ m}^3$ .

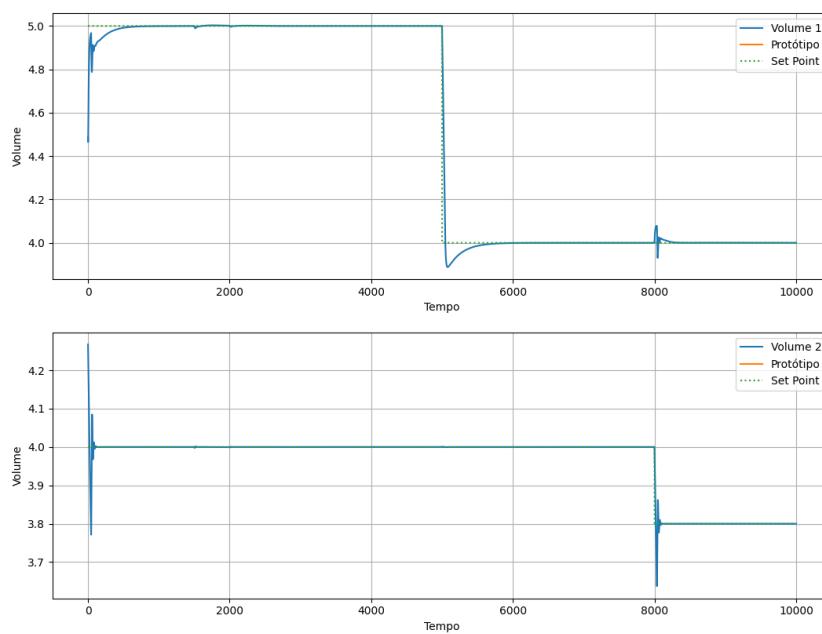
Pelo gráfico 2 pode-se observar que as temperaturas dos evaporadores seguem as

**Tabela 8 – Setpoints do processo**

<b>ETAPAS</b>	<b>SETPOINT</b>
0	V1 = 5 V2 = 5 TC = 70 X2 = 0,44
1500	V1 = 5 V2 = 4 TC = 70 X2 = 0,48
2000	V1 = 5 V2 = 4 TC = 70 X2 = 0,50
5000	V1 = 4 V2 = 4 TC = 65 X2 = 0,48
8000	V1 = 4 V2 = 3,8 TC = 75 X2 = 0,44

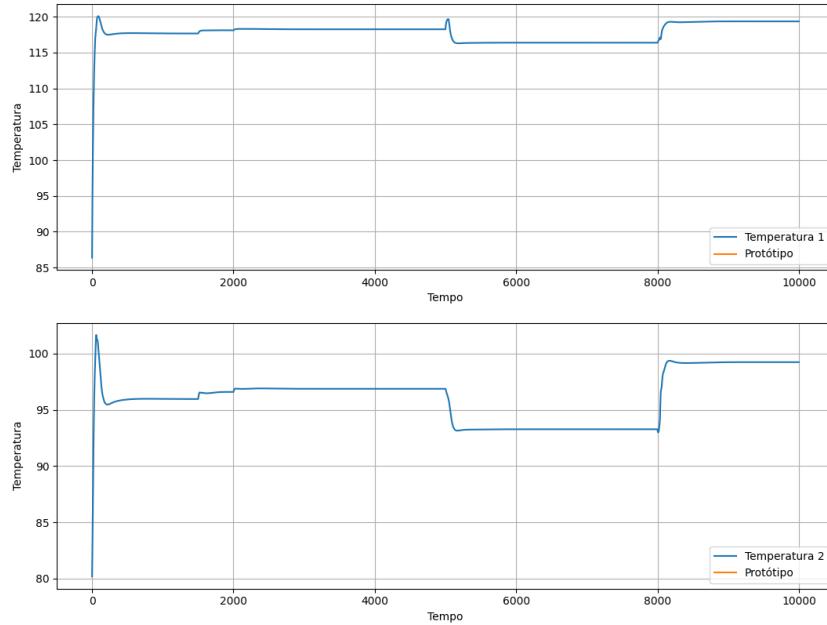
mudanças impostas no processo pelo setpoints da concentração X2 e dos volumes, V1 e V2.

No gráfico 3 são mostradas as concentrações do licor negro dos 2 evaporadores. O setpoint do X1 é influenciado pelo setpoint do X2, e a concentração de X1 é controlada pela abertura da válvula  $f_s$ . Se o setpoint de X2 for maior que X1, o setpoint de X1 tenderá a aumentar.

**Gráfico 1 – Volumes dos evaporadores**

**Fonte:** Autoria própria

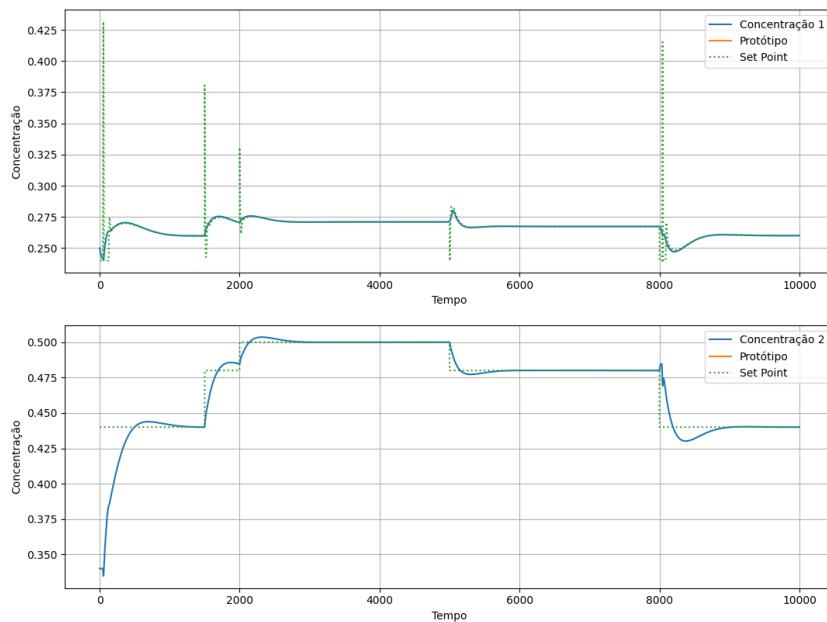
**Gráfico 2 – Temperatura dos evaporadores**



**Fonte: Autoria própria**

Caso contrário, o setpoint de X1 diminui. Se X1 for maior que seu setpoint a abertura da válvula fs aumenta, caso contrário ela diminui. Portanto pode-se observar pelo gráfico, e acompanhar pela receita da tabela 8, o aumento da concentração do X1 nas etapas 0, 1500 e 2000, e uma diminuição nas etapas 5000 e 8000.

**Gráfico 3 – Concentração do licor negro**

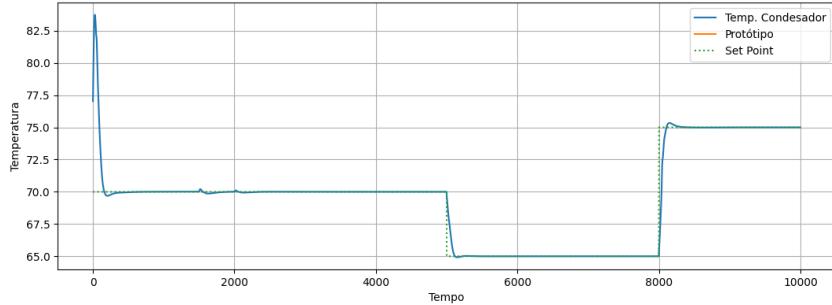


**Fonte: Autoria própria**

O gráfico 4 reapresenta a temperatura no condensador, TC, a qual foi configurada para

mudar para  $65^C$  na etapa 5000 e para  $75^C$  na etapa 8000. Pode-se observar que o aumento do setpoint dessa temperatura TC, cria um pico instantâneo no setpoint X1.

**Gráfico 4 – Temperatura no condensador**

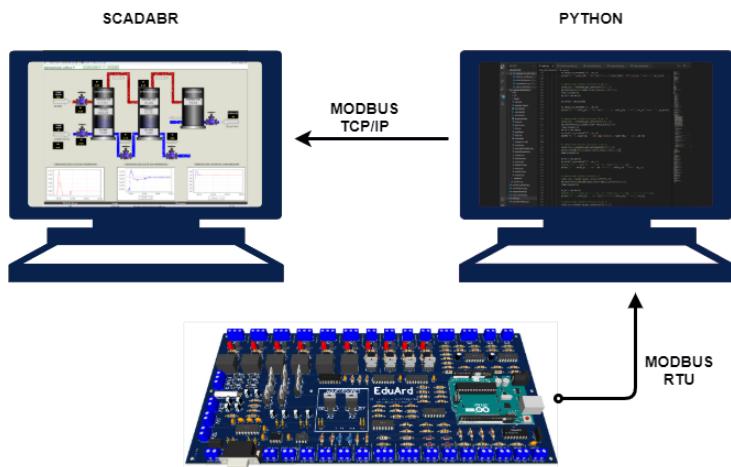


**Fonte:** Autoria própria

## 6.2 ENSAIO HÍBRIDO

Foi adicionado no protótipo a realização da sintonia do PID por meio de uma comunicação serial. O sistema em Python será incrementado de uma classe servidormodbus.py, a qual será responsável para criar um servidor modbus TCP/IP na porta local 502. Esse servidor se comunicará como o ScadaBR. O diagrama é mostrado na figura 50. Com o supervisório

**Figura 50 – Diagrama da comunicação serial**



**Fonte:** Autoria própria

conectado, pode-se criar, configurar e ativar o datasource e os datapoints como mostrado na figura 51.

Enfim, depois de confeccionar o sinótico da interface gráfica do ScadaBR, pode-se acionar o sistema e observar os resultados. Na figura 52, é mostrado o sinótico do supervisório,

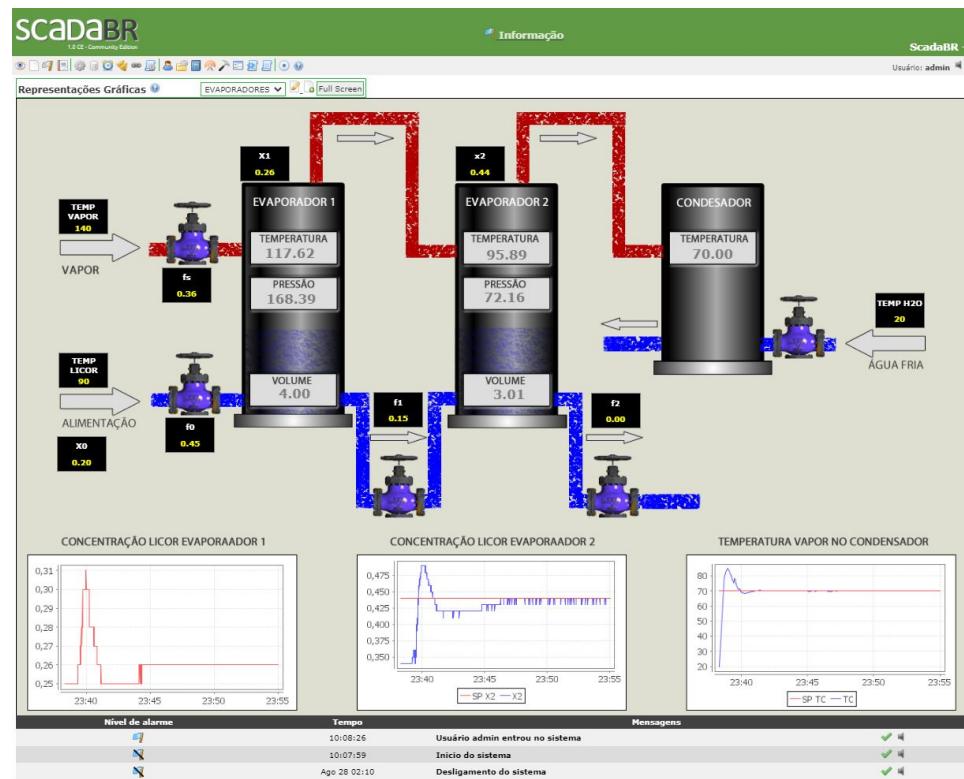
Figura 51 – Configuração parâmetros ScadaBR



Fonte: Autoria própria

o qual mostra graficamente a evolução das variáveis em cada evaporador como temperatura (T1 e T2), pressão, volume (V1 e V2), válvula de alimentação evaporador 1 (f0), válvula de vazão/alimentação entre os evaporadores (f1), válvula vazão do evaporador 2(f2), concentração do licor negro no 2 evaporadores X1 e X2.

Figura 52 – Sinótico do ScadaBR



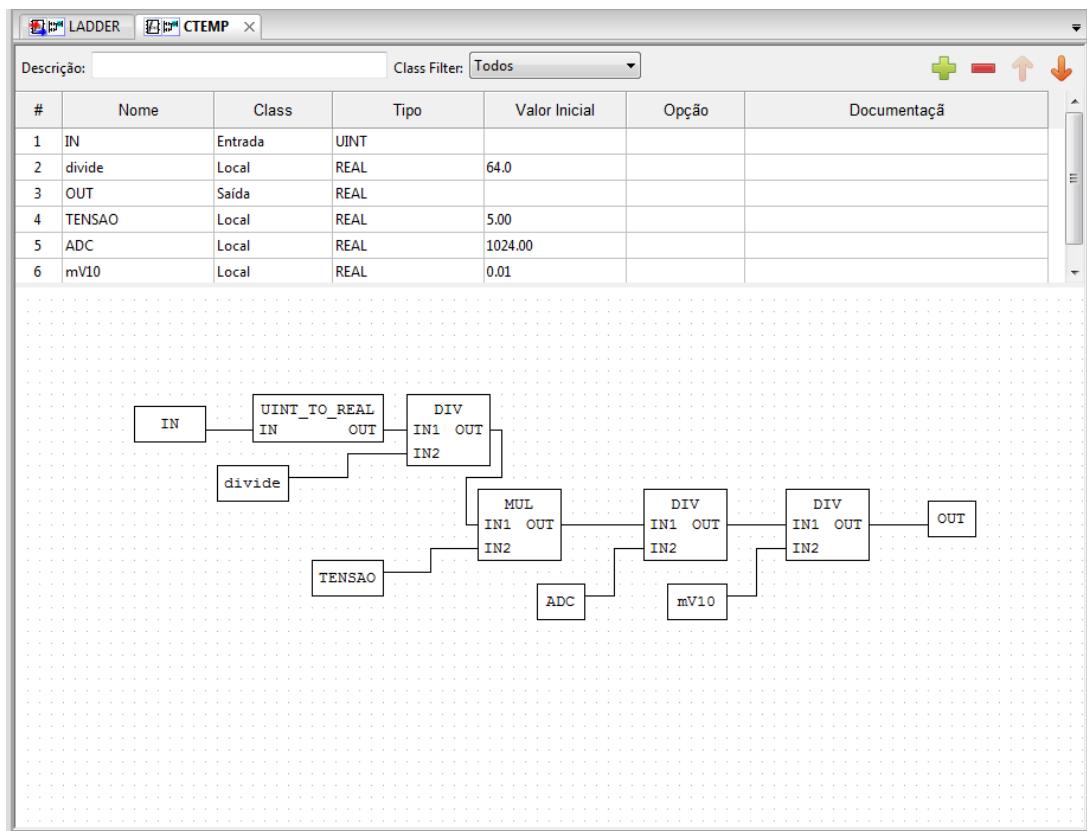
Fonte: Autoria própria

Pode-se observar também os gráficos que acompanha em tempo real o desempenho das concentrações do licor negro nos 2 evaporadores e a temperatura do condensador.

### 6.3 ENSAIO COM AQUECEDOR

Neste ensaio será testado o funcionamento dos aquecedores instalados no protótipo hardware. O Transistor TIP31C será controlado na base por um sinal interno PWM. Como não existe resistências neste circuito aquecedor, a temperatura interna do transistor irá subir rapidamente, sendo que o sensor de temperatura encostado nele absorverá o calor por convecção. Este sensor será responsável por transmitir um valor em tensão para o microcontrolador que o converterá em um valor de temperatura em grau Celsius. Neste ensaio pretende-se manter a

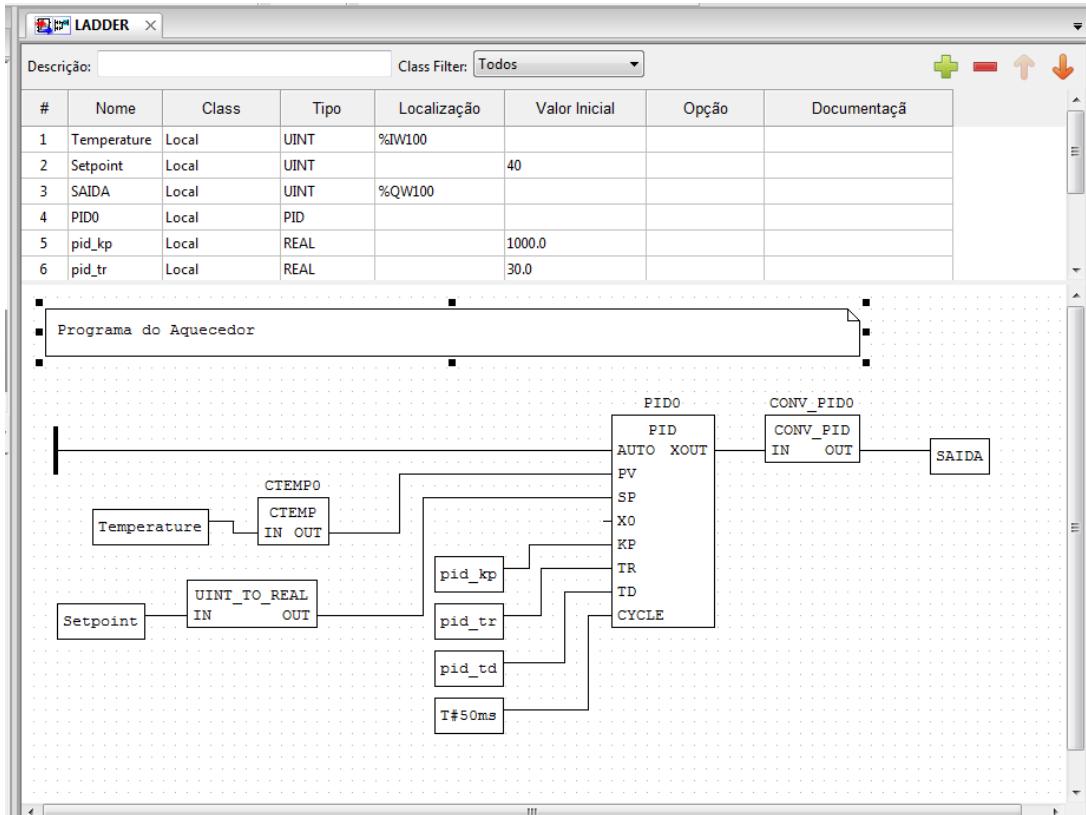
**Figura 53 – Bloco Ladder para conversão de temperatura**



**Fonte: Autoria própria**

temperatura constante em 40°C. Portanto elabora-se um programa de sintonia PID no software OpenPLC Editor, o qual será importado no servidor OpenPLC. Este servidor realizará a conexão Modbus RTU com o protótipo e será responsável pela sintonia. Além disso, o servidor manterá uma segunda conexão Modbus TCP/IP com o supervisório ScadaBR, o qual será responsável por acompanhar em tempo real a evolução do processo.

**Figura 54 – Sintonia PID**



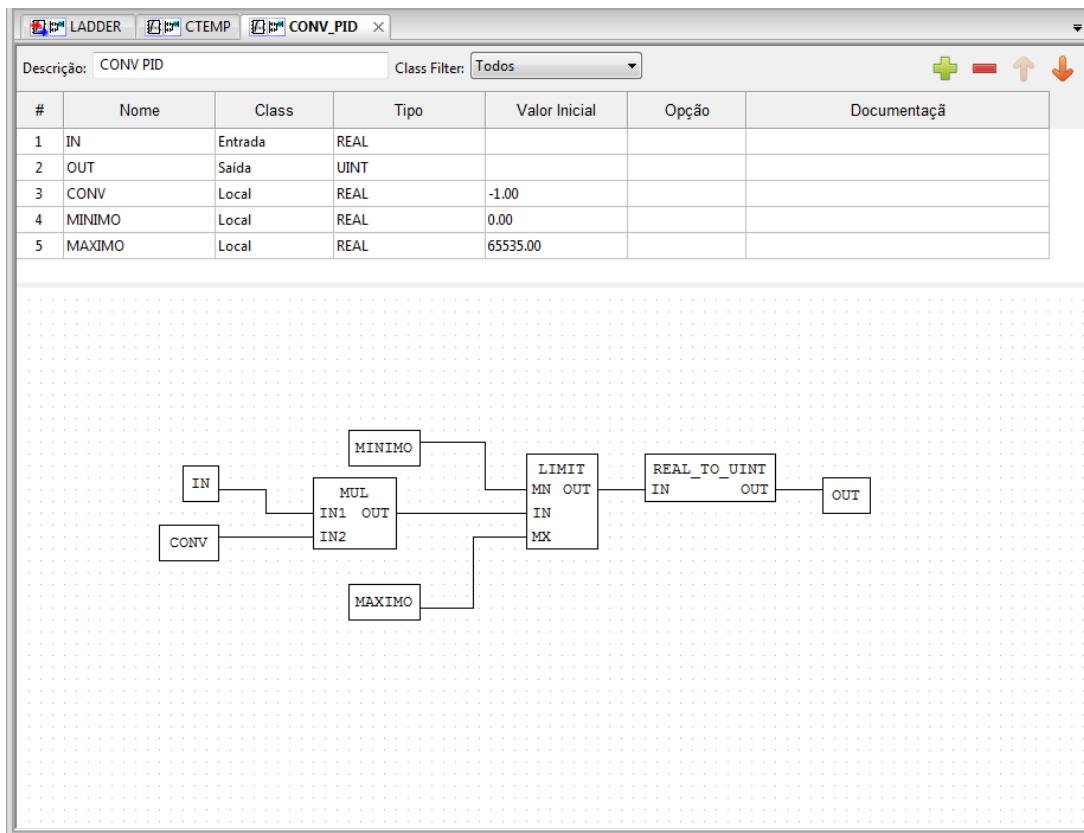
**Fonte:** Autoria própria

O OpenPLC Editor não oferece blocos Ladder complexos, portanto foi necessário desenvolver 2 blocos extras para complementar o código necessário. O bloco CTEMP mostrado na figura 53 realiza a conversão do número inteiro recebido por Modbus para o valor de tipo REAL que representa a tensão recebida do sensor. De acordo com o datasheet do sensor de temperatura LM35 10mV representa 1°C, portanto divide-se o valor obtido por 0.01. O programa central mostrado na figura 54, realiza a sintonia por meio do bloco PID, o qual recebe o valor de temperatura convertido pelo bloco CTEMP, o valor do setpoint, as constantes de sintonia, e o valor do ciclo de varredura.

Este bloco PID trabalha em uma única direção (variável-setpoint) e não é configurável, portanto, por meio do bloco CONV\_PID, figura 55, realiza-se as conversões necessárias. Primeiro inverte-se a direção do valor de saída do PID multiplicando-o por -1, em seguida são estipulados os limites: o mínimo como 0 e o máximo como 65535. Enfim, o valor de tipo REAL é convertido em inteiro para ser enviado para o protótipo. O arquivo desenvolvido é salvo em formato .st e importado no servidor Openplc como mostrado na figura 56.

Finalmente pode-se observar em tempo real o funcionamento do sistema por meio do sinótico do ScadaBR, figura 57

**Figura 55 – Conversão do valor PID**



**Fonte:** Autoria própria

**Figura 56 – Conversão do valor PID**

**Programs**

Here you can upload a new program to OpenPLC or revert back to a previous uploaded program shown on the table.

Program Name	File	Date Uploaded
Aquecedores	917949.st	Sep 01, 2021 - 10:22PM
Blank Program	blank_program.st	May 24, 2018 - 03:02PM

[List all programs](#)

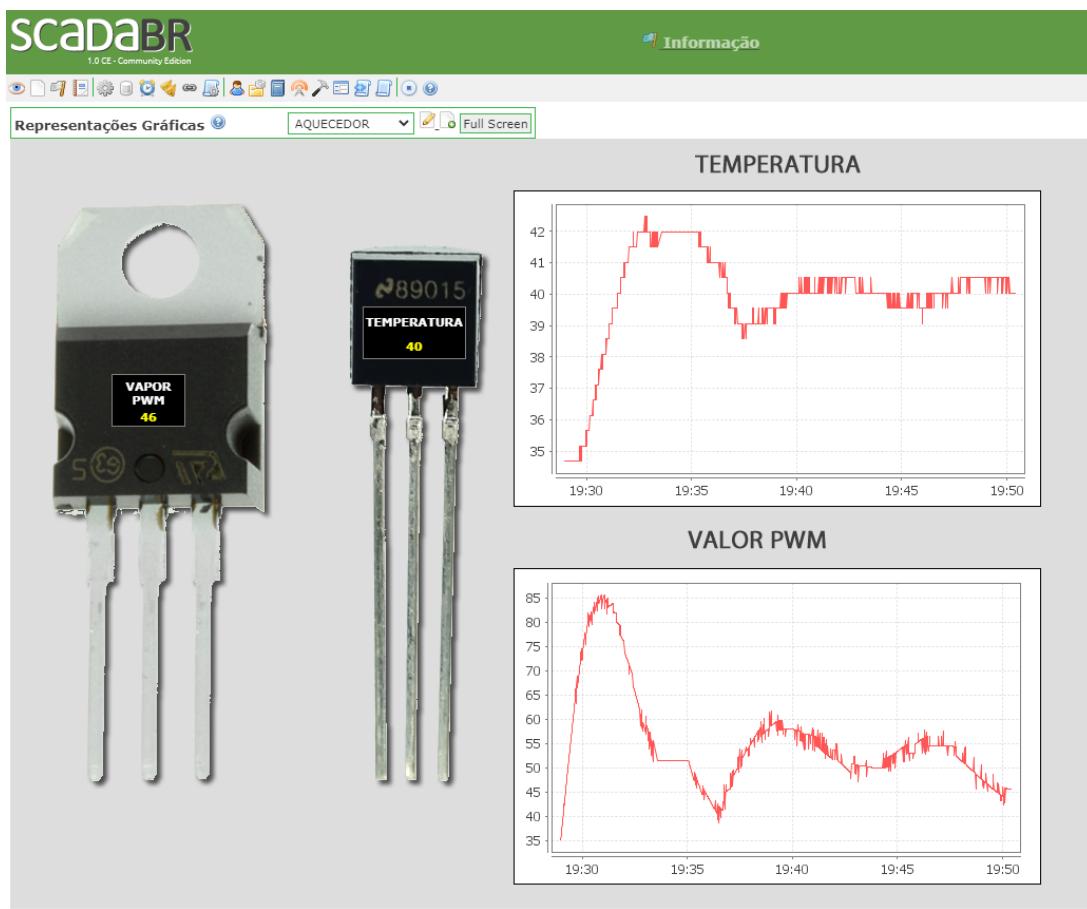
**Monitoring**

Refresh Rate (ms): 100

Point Name	Type	Location	Forced	Value
Temperature	UINT	%IW100	No	3008
SAIDA	UINT	%QW100	No	40355

**Fonte:** Autoria própria

**Figura 57 – Sinoptico de ScadaBR**



©2009-2011 Fundação Certi, MCA Sistemas, Unis Sistemas, Conetec. Todos os direitos reservados.

**Fonte:** Autoria própria

#### 6.4 CONCLUSÕES DO CAPÍTULO 6

Neste capítulo foram apresentados três ensaios: virtual, híbrido e com aquecedores. No ensaio virtual, onde todos os cálculos foram realizados pelo Python, constatou-se um bom funcionamento e obteve-se o resultados esperados. No ensaio híbrido, onde foi adicionado o EDUARD e o ScadaBR, a comunicação Modbus RTU e Modbus TCP/IP funcionou satisfatoriamente atendendo os resultados esperados. Enfim, no ensaio com os aquecedores, onde a sintonia do controlador foi desenvolvida no OpenPLC Editor e gravada no servidor OpenPLC, mostrou-se confiável no controle da temperatura do transistor, mantendo essa temperatura estável em 40°C.

## 7 CONCLUSÕES E PERSPECTIVAS

Esta dissertação apresentou o desenvolvimento de um protótipo de hardware aliado ao uso de softwares open-source para ser utilizado dentro um laboratório acadêmico na área de controle e automação. Esse tipo de ambiente possibilita melhorias na prática de ensino enquanto proporciona aos alunos uma experiência completa, desde construir a própria ferramenta de hardware, programar de modo simples e rápido o sistema até conseguir realizar ensaios resolvendo problemas mais elaborados de controle. Analisou-se o bom funcionamento dos softwares open-source utilizados, demonstrando que uma ferramenta de baixo custo pode ser útil para ser utilizado no ensino de automação de processos.

Ficou demonstrado que um processo qualquer modelado por equações diferenciais e algébricas pode ser integrado ao ambiente, permitindo durante a simulação a que estratégia de controle sejam testadas neste sistema programável com software e hardware open-source.

O projeto hardware foi desenvolvido por meio do software-web EasyEda (EASYEDA, 2019), que apesar de poder ser usado por meio do navegador, já que o software disponibiliza o auto-roteamento web, mas preferiu-se instalar no computador o EasyEda-router que torna a tarefa mais rápida. Para simulação e testes de circuitos análgicos usou-se o Qucs (MARGRAF; JAHN, 2003). Para programação em Python da planta didática e o software embarcado no Arduino Uno R3, utilizou-se o VSCode com extensão para código Python e a extensão Platformio para código C++ (MICROSOFT, 2015). Por ser um software brasileiro e com um grande comunidade de suporte escolhou-se ScadaBR como supervisório. Enfim o servidor OpenPlc para programação em Ladder foi escolhido por ser o mais simples e eficaz entre os analisados nessa pesquisa. Completa o projeto a plataforma hardware Arduino Uno R3, que pela gigantesca comunidade de suporte, se torna a mais adaptada para uso em laboratório de alunos iniciante de engenharia.

Alguns trabalhos futuros serão estudados para este ambiente, como implementação de outras tecnologias, utilização do ambiente de forma remota ou um ambiente via web, implementação de sistemas wireless, simulação de outros processos industriais, estudo mais aprofundado sobre tecnologias disponíveis no laboratório e implementações de outras topologias de comunicação das redes ou protocolos industriais e expansão do hardware com a utilização de microcontroladores mais avançado ou FPGA, além da utilização destas tecnologias com turma de alunos reais.

## REFERÊNCIAS

- AGUIRRE, L. **Controle de Sistemas Amostrados**. Belo Horizonte: Independently Published, 2019.
- ALMEIDA, R.M.A. de; MORAES, C.H.V. de; SERAPHIM, T. de Faria Piola. **Desenvolvendo Software para Microcontroladores em Linguagem C**. Rio de Janeiro: Elsevier Brasil, 2017.
- ALVES, T. R.; BURATTO, M.; SOUZA, F. M. de; RODRIGUES, T. V. Openplc: An open source alternative to automation. **IEEE Global Humanitarian Technology Conference**, 2014.
- ARDUINO, Team. **The MKR family gets bigger with two new IoT boards!** 2018. Disponível em: <https://blog.arduino.cc/2018/05/12/the-mkr-family-gets-bigger-with-two-new-iot-boards>.
- BEREMIZ, Team. **Free software for automation**. 2017. Disponível em: <https://beremiz.org/>. Acesso em: 15 jan. 2021.
- BOLTON, W. **Programmable Logic Controllers**. London: Elsevier Science, 2006.
- BONI, Djones. **DAC com sinal PWM**. 2018. Disponível em: <https://professorelettrico.com/posts/hardware/dac-com-sinal-pwm/>. Acesso em: 19 jan. 2021.
- BRYAN, L.A.; BRYAN, E.A. **Programmable Controllers: Theory and Implementation**. Atlanta: American Technical Publishers, Incorporated, 2002.
- CARDOSO, Bruno Matheus. **Desenvolvimento de um robô móvel com trajetória programável utilizando a plataforma Arduíno**. 2016. Monografia (Bacharelado em Ciência da Computação) — Universidade Tecnológica Federal do Paraná, 2016.
- EASYEDA, Team. **An easier and powerful online PCB design tool**. 2019. Disponível em: <https://easyeda.com/>. Acesso em: 23 jan. 2021.
- ESPRESSIF, Team. **A feature-rich MCU with integrated Wi-Fi and Bluetooth connectivity for a wide-range of applications**. 2018. Disponível em: <https://www.espressif.com/en/products/socs/esp32>. Acesso em: 19 jan. 2021.
- FOULIS, Christos. A portable low-cost arduino-based laboratory kit for control education. **UKACC 12th International Conference on Control (CONTROL)**, 2018.

FOUST, A. S.; WENZEL, L. A. **Principios das operações unitárias**. Rio de Janeiro: Editora Guanabara, 1982.

FRANCHI, C.M.; CAMARGO, V.L.A. de. **Controladores lógicos programáveis**. São Paulo: Érica, 2008.

FREITAS, Carlos Márcio. **Protocolo Modbus: Fundamentos e Aplicações**. 2014. Disponível em: <https://www.embarcados.com.br/protocolo-modbus/>. Acesso em: 27 set. 2020.

FRIZZARIN, Fernando Bryan. **Guia para colocar suas ideias em prática**. São Paulo: Casa do código, 2016.

GALADIMA, Ahmad. Arduino as a learning tool. **2014 11th International Conference on Electronics, Computer and Computation (ICECCO)**, 2014.

GARRIGOS; MARROQUI. Designing arduino electronic shields: Experiences from secondary and university courses. **2017 IEEE Global Engineering Education Conference (EDUCON)**, 2017.

GOMES, João Paulo de Toledo. **Protótipo para monitoramento e aquisição de dados de pressão através de sensores sem fio em sistemas de abastecimento de água com utilização da plataforma Radiuino e ScadaBR**. 2015. Dissertação (Mestrado) — Universidade de Ribeirão Preto, Ribeirão Preto, 2015.

GRANFORS, A.; NILSSON, B. Modelling of single evaporators. **Computers & chemical engineering**, Elsevier, 1999.

GUEDES, Danila Branco. **Linguagem de programação Python e Arduino como ferramenta para motivar estudantes iniciantes em programação**. Monografia (Tecnologia em Sistemas de Telecomunicações) — Universidade Tecnológica Federal do Paraná.

HUGHES, J.M. **Arduino per tecnici, ingegneri e maker**. Milano: Tecniche Nuove, 2016.

HURTUK, Ján; CHOVARNEC, Martin; ÁDAM, Norbert. The arduino platform connected to education process. **2017 IEEE 21st International Conference on Intelligent Engineering Systems (INES)**, 2017.

KOBAYASHI, Tiago Hiroshi. **Uma Ferramenta de Manipulação de Pacotes para Análise de Protocolos de Redes Industriais Baseados em TCP/IP**. 2009. Dissertação (Mestrado) — Universidade Federal Do Rio Grande Do Norte, Natal, 2009.

LI, Jen-Hsing. Control system laboratory with arduino. **2018 International Symposium on Computer, Consumer and Control (IS3C)**, 2018.

LIMA, Marco De. **Avr E Arduino: Tecnicas De Projeto**. Florianópolis: Clube de Autores, 2012.

MARGRAF, Michael; JAHN, Stefan. **Quite Universal Circuit Simulator**. 2003. Disponível em: <http://qucs.sourceforge.net/>. Acesso em: 1 jan. 2021.

MARTINS, Luis Miguel. **Projecto dum Laboratório Remoto para Automação de Processos Industriais**. 2013. Dissertação (Mestrado) — Instituto Superior de Engenharia de Lisboa, Lisboa, 2013.

MBLOGIC, Team. **MBLogic for an open world in automation**. 2010. Disponível em: <http://mblogic.sourceforge.net/mbapps/apps.html>. Acesso em: 15 jan. 2021.

MCA Sistemas. **Scadabr - Supervisory control and data acquisition software**. 2013. Disponível em: <http://www.scadabr.com.br/>. Acesso em: 30 set. 2020.

MICROBERTS, M. **Arduino Básico**. São Paulo: Novatec Editora, 2018.

MELLO, Danilo Augusto. **Solução para monitoramento ambiente utilizando Arduino**. 2017. Monografia (Tecnologia em Sistemas para Internet) — Universidade Tecnológica Federal do Paraná, 2017.

MENESTRINA, Tatiana C.; BAZZO, W. A. Ciência, tecnologia e sociedade e formação do engenheiro: análise da legislação vigente. **Revista Brasileira de Ensino de Ciência e Tecnologia**, Ponta Grossa, 2008.

MICROSOFT, Team. **Code editing, redefined**. 2015. Disponível em: <https://code.visualstudio.com/>. Acesso em: 23 jan. 2021.

MODBUS-IDA. **MODBUS Messaging on TCP/IP Implementation Guide**. 2004. Disponível em: [https://modbus.org/docs/Modbus\\_Messaging\\_Implementation\\_Guide\\_V1\\_0a.pdf](https://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0a.pdf). Acesso em: 27 set. 2020.

MONK, S. **Programação com Arduino**. Porto Alegre: Bookman Editora, 2017.

MONTEIRO, Luis Henrique. **O uso do Arduino e do Processing no ensino de física**. 2016. Dissertação (Mestrado) — UNIRIO, Rio de Janeiro, 2016.

MORAES, C.C. de; CASTRUCCI, P. de Lauro. **Engenharia de automação industrial**. Rio de Janeiro: LTC, 2007.

MULLER, Christa. **PVBrowser, simple process visualization**. 2020. Disponível em: <https://pvbrowser.de/>. Acesso em: 18 jan. 2021.

NASCIMENTO, João Maria Araújo do; LUCENA, Pedro Berretta de. **PROTÓCOLO MODBUS**. 2003. Disponível em: [https://www.dca.ufrn.br/~affonso/FTP/DCA447/trabalho3/trabalho3\\_13.pdf](https://www.dca.ufrn.br/~affonso/FTP/DCA447/trabalho3/trabalho3_13.pdf). Acesso em: 27 set. 2020.

NUNES, Heberval Moreira. **Desenvolvimento e aplicação de um kit experimental com arduino para o ensino do eletromagnetismo**. 2018. Dissertação (Mestrado) — Universidade Federal do Maranhão, São Luís, 2018.

OSHLWA. **Open Source Hardware Association**. 2012. Disponível em: <https://www.oshwa.org/>. Acesso em: 10 jan. 2021.

PACHECO, Atilano Fernandez; MARTIN, Sergio; CASTRO, Manuel. Implementation of an arduino remote laboratory with raspberry pi. **2019 IEEE Global Engineering Education Conference (EDUCON)**, 2019.

PASCALSCADA, Team. **PascalSCADA, HMI/SCADA for developers**. 2018. Disponível em: <https://www.pascalscada.com/>. Acesso em: 18 jan. 2021.

PEREIRA, Janinha Aparecida. **Um recurso didático baseado na plataforma Arduino para o ensino de energia**. 2018. Dissertação (Mestrado) — Universidade Tecnológica Federal do Paraná, Ponta Grossa, 2018.

PERTENCE, A. **Amplificadores Operacionais e Filtros Ativos**. Porto Alegre: Bookman Editora, 2015.

PETRUZELLA, F.D. **Controladores Lógicos Programáveis**. New York: Bookman Companhia ED, 2014.

PJRC. **Teensy 4.0 Development Board**. 2019. Disponível em: <https://www.pjrc.com/store/teensy40.html>. Acesso em: 19 jan. 2021.

PLATFORMIO, labs. **Professional collaborative platform for embedded development**. 2014. Disponível em: <https://platformio.org/>. Acesso em: 1 jan. 2021.

PLCOPEN, Team. **IEC 61131-3: a norma para programação.** 1993. Disponível em: [https://plcopen.org/sites/default/files/downloads/intro\\_iec\\_march04\\_portuguese.pdf](https://plcopen.org/sites/default/files/downloads/intro_iec_march04_portuguese.pdf). Acesso em: 16 jan. 2021.

PROVIEW, Team. **Proview, open-source process control.** 2020. Disponível em: <http://www.proview.se/v3/>. Acesso em: 18 jan. 2021.

PRUDENTE, F. **Automação Industrial - Plc : Teoria E Aplicações.** Rio de Janeiro: LTC, 2008.

PYTHON. **Programming Language.** 2001. Disponível em: <https://www.python.org/>. Acesso em: 1 jan. 2021.

RIOS, Lucas. Dal. Softwares livres no ensino de engenharia: uma atitude socialmente justa, economicamente viável e tecnologicamente sustentável. **Congresso Brasileiro de ensino de Engenharia,** 2006.

RUBIM, Roberto. **Microcontrolador Arduino no ensino de Física: Proposta e aplicação de uma situação de aprendizagem sobre o tema Luz e Cor.** 2014. Dissertação (Mestrado) — Universidade Federal de São Carlos, São Carlos, 2014.

SAVOCHEJKO, Roman. **OpenSCADA, open supervisory control and data acquisition project.** 2018. Disponível em: <http://oscada.org/>. Acesso em: 16 jan. 2021.

SILVA, Daniel Filipe. **Sistema didático de uma unidade industrial de processos discretos.** 2018. Dissertação (Mestrado) — Universidade do Porto, Porto, 2018.

SOUZA, Jacqueline De. **Ambiente didático para controle avançado de processos industriais.** 2016. Dissertação (Mestrado) — Universidade Tecnológica Federal do Paraná, Curitiba, 2016.

STMicroelectronics. **Medium-density performance line ARM®-based 32-bit MCU with 64 or 128 KB Flash, USB, CAN, 7 timers, 2 ADCs, 9 com. interfaces.** 2018. Disponível em: <https://s3-sa-east-1.amazonaws.com/robocore-lojavirtual/1067/STM32F103C8T6.pdf>. Acesso em: 18 jan. 2021.

TECGRAF PUC-RIO. **Instituto Tecgraf de Desenvolvimento de Software Técnico-Científico da PUC-Rio (Tecgraf/PUC-Rio) é um modelo brasileiro de sucesso em PDI autossustentável.** 2021. Disponível em: <https://www.tecgraf.puc-rio.br/>. Acesso em: 19 jan. 2021.

WESTHUES, Jonathan. **LDmicro: Ladder Logic for PIC and AVR.** 2016. Disponível em: <http://mblogic.sourceforge.net/mbapps/apps.html>. Acesso em: 15 jan. 2021.

ZANCAN, M. D. **Controladores programáveis**. Santa Maria: CTISM/UFSM, 2011.

ZELENOVSKY, Ricardo; MENDONÇA, Alexandre. **Arduino, guia avançado de projetos**. Rio de Janeiro: Interciência, 2019.

## **APÊNDICES**

## APÊNDICE A - ARQUIVOS DE PROGRAMAÇÃO

```

1 Main.py
3
5   from arduino_modbus import ard_mod
6   from controladores import *
7   from evaporadores import *
8   import numpy as np
9   import serial
10  import time
11  from servidormodbus import ServidorModbus
12  from matplotlib import pyplot as plt
13  from arduino_modbus import ard_mod
14
15 flag=1
16 flag_arduino=0
17 flag_modbus=0
18
19 if flag_modbus==1:
20     modbus = ServidorModbus('localhost', 502)
21     modbus.run()
22     #lista MODBUS
23     entradas_analogicas = [0,0,0,0,0,0,0,0,0]
24     endereco_entrada = 1000
25     saidas_analogicas = [0,0,0,0,0]
26     endereco_saida = 2000
27     set_point = [0,0,0,0]
28     endereco_setpoint = 5000
29     perturbacoes = [0,0,0,0]
30     endereco_perturbacoes = 6000
31
32 if flag_arduino==1:
33     # Modebus RTU: Python-Arduino
34     am = ard_mod()
35
36
37 if flag_arduino==2:
38     ser = serial.Serial('COM3', 115200)
39     # ser1 = serial.Serial('COM7', 9600)
40     time.sleep(3)
41
42 spx1=[]
43 spx2=[]
44 sptc=[]

```

```

45 spv1=[]
46 spv2=[]
47
48
49 # Instancia o modelo do sistema
50 e=Evaporadores()
51
52 ea=Evaporadores()
53
54 # Instancia os PIDs do controlador
55 pid1=Controlador()
56 pid2=Controlador()
57 pid3=Controlador()
58 pid4=Controlador()
59 pid5=Controlador()
60 pid6=Controlador()
61
62 # Configura o regime inicial
63 e.set_regime(5, 0.2500, 312.8, 5, 0.3400, 274.1)
64 e.atualiza_sensores()
65
66 ea.set_regime(5, 0.2500, 312.8, 5, 0.3400, 274.1)
67 ea.atualiza_sensores()
68 # contador=0
69
70 #SET POINTS
71 aspv1=5
72 aspv2=4
73 asptc=70
74 aspx2=0.44
75 #t=np.linspace(0,110,110)
76 stp=0.44
77 t=np.linspace(0,10000,10000)
78
79 for i in t:
80
81     print("PROCESSANDO: %d"%i,"passos")
82
83     #setpoints mudando
84     if(i>=1500):
85         pid5.set_setpoint(x2=0.48)
86         aspx2=0.48
87     if (i >= 2000):
88         pid5.set_setpoint(x2=0.50)
89         aspx2=0.50
90     if (i >= 5000):
91         pid5.set_setpoint(x2=0.48)

```

```

    aspx2=0.48
93    pid1.set_setpoint(v1=4)
    aspv1=4
95    pid4.set_setpoint(Tc=65)
    asptc=65
97    if (i >= 8000):
        pid5.set_setpoint(x2=0.44)
99        aspx2=0.44
        pid2.set_setpoint(v2=3.8)
101       aspv2=3.8
        pid4.set_setpoint(Tc=75)
103       asptc=75

105    # Controla V1 por f0
    valor_V1=e.get_sensores("V1")
107    p1=pid1.pid("V1", 1, valor_V1,K=3,Ti=200, Td=2)
    w1=pid1.limites_atuador(p1,0,5)
109    e.atualiza_atuador("f0",w1)

111    # Controla V2 por f1
    # if i==5000:
113    #     pid2.atualiza_setpoint("V2",3)
    valor_V2=e.get_sensores("V2")
115    p2 = pid2.pid("V2", 1, valor_V2, K=3,Ti=2, Td=2)
    w2 = pid2.limites_atuador(p2, 0, 5)
117    e.atualiza_atuador("f1", w2)

119
121    # Controla V2 por f2
    valor_V3 = e.get_sensores("V2")
123    p3 = pid3.pid("V2", -1, valor_V3,K=0.3,Ti=20, Td=10)
    w3 = pid3.limites_atuador(p3, 0, 5)
125    e.atualiza_atuador("f2", w3)

127

129    # Controla Tc por fc
    valor_Tc = e.get_pertubacoes("Tc")
131    p4 = pid4.pid("Tc", -1, valor_Tc, K=0.003, Ti=2, Td=2)
    w4 = pid4.limites_atuador(p4, 0, 5)
133    e.atualiza_atuador("fc", w4)

135
137    # Controla x2
    valor_x2 = e.get_sensores("X2")
    p5 = pid5.pid("X2", 1, valor_x2, K=3, Ti=200, Td=100)

```

```

139     w5 = pid5.limites_atuador(p5, 0.25, 1)
140     pid5.set_variavel(w5)
141     wx2=pid5.get_variavel()

143
144     # Controla x1 por fs
145     pid6.atualiza_setpoint("X1",wx2)
146     # pid6.atualiza_setpoint("X1", valor_x2)
147     valor_x1 = e.get_sensores("X1")
148     p6 = pid6.pid("X1", 1, valor_x1, K=3, Ti=3, Td=10)
149     w6 = pid6.limites_atuador(p6, 0, 5)
150     e.atualiza_atuador("fs", w6)

151
152     if flag_modbus==1:
153         entradas_analogicas[0]=int(pid1.get_sensores("V1")*100)
154         set_point[0]=int(pid1.get_setpoint("V1")*100)
155         saidas_analogicas[0]=int(e.get_atuadores("f0")*100)
156         entradas_analogicas[1]=int(pid2.get_sensores("V2")*100)
157         set_point[1]=int(pid2.get_setpoint("V2")*100)
158         saidas_analogicas[1]=int(e.get_atuadores("f1")*100)
159         saidas_analogicas[2]=int(e.get_atuadores("f2")*100)
160         entradas_analogicas[2]=int(e.get_pertubacoes("Tc")*100)
161         set_point[2]=int(pid3.get_setpoint("Tc")*100)
162         saidas_analogicas[3]=int(e.get_atuadores("fc")*100)
163         entradas_analogicas[3]=int(pid5.get_sensores("X2")*100)
164         set_point[3]=int(pid5.get_setpoint("X2")*100)
165         entradas_analogicas[4]=int(pid6.get_sensores("X1")*100)
166         saidas_analogicas[4]=int(e.get_atuadores("fs")*100)

167
168         entradas_analogicas[5]=int(e.get_sensores("T1")*100)
169         entradas_analogicas[6]=int(e.get_sensores("T2")*100)
170         entradas_analogicas[7]=int(e.get_sensores("P1")*100)
171         entradas_analogicas[8]=int(e.get_sensores("P2")*100)

172
173         perturbacoes[0]=int(e.get_pertubacoes("x0")*100)
174         perturbacoes[1]=int(e.get_pertubacoes("T0")*100)
175         perturbacoes[2]=int(e.get_pertubacoes("Ts")*100)
176         perturbacoes[3]=int(e.get_pertubacoes("Tcin")*100)

177
178         modbus.set_entradas(endereco_entrada,entradas_analogicas)
179         modbus.set_saidas(endereco_saida,saidas_analogicas)
180         modbus.set_setpoints(endereco_setpoint,set_point)
181         modbus.set_perturbacoes(endereco_perturbacoes,perturbacoes)

182
183         # print(" entradas IW1000.%s \r" % entradas_analogicas)
184         # print("holding Register \r\n R1000: %s" % modbus._db.get_words(1000))
185         # print(" saidas QW2000.%s \r\n" % saidas_analogicas)

```

```

# print(" setpoints QW5000.%s \r" % set_point)
187
# print("===== ")
188
time.sleep(0.05)

189

191 if flag_arduino==1:
192     try:
193         # comunica o arduino Controla V1 por f0
194         valor_V1a = round(ea.get_sensores("V1"), 2)
195         am.escrever(3,200,2,aspv1,valor_V1a,1,1,0,5,True)
196         time.sleep(0.1)

197         at_v1 = am.ler(9)
198
199         at_erro = am.ler(10)
200
201         ea.atualiza_atuador("f0", at_v1)
202         print("V1: ",valor_V1,"- f0=",w1,"=====",valor_V1a,"- f1=",at_v1, " "
203               "erro = ", at_erro)

205
206         # comunica o arduino Controla V2 por f1
207         valor_V2a = round(ea.get_sensores("V2"), 2)
208         am.escrever(3,2,2,aspv2,valor_V2a,2,1,-1,5,True)
209         time.sleep(0.1)
210         at_v2 = am.ler(9)
211
212         at_erro2 = am.ler(10)
213
214         ea.atualiza_atuador("f1", at_v2)
215         print("V2: ", valor_V2, "- f1=",w2,"=====", valor_V2a, "- f1=", at_v2,
216               " erro = ", at_erro2)

217
218         # comunica o arduino Controla V2 por f2
219         valor_V3a = round(ea.get_sensores("V2"), 2)
220         am.escrever(3,2,2,aspv2,valor_V3a,3,0,0,5,True)
221         time.sleep(0.1)
222         at_v3 = am.ler(9)
223         ea.atualiza_atuador("f2", at_v3)
224         print("V3: ", valor_V3, "- f1=",w3,"=====", valor_V3a, "- f2=", at_v3)

225

226         # comunica o arduino Controla Tc por fc
227         valor_Tca = round(ea.get_pertubacoes("Tc"), 2)
228         am.escrever(0.03,2,2,asptc,valor_Tca,4,-1,0,5,True)
229         time.sleep(0.1)

```

```

231     at_tc = am.ler(9)
233     ea.atualiza_atuador("fc", at_tc)
234     #print("Tc: ", valor_Tc, "-", w4, "=", valor_Tca, "-", pa4)
235     print("Tc ", valor_Tc, "- fc=",w4,"=====", valor_Tca, "- f2=", at_tc)

237
238     # comunica o arduino Controla x2
239     valor_x2a =round( ea.get_sensores("X2"), 2)
240     am.escrever(3,200,100,aspx2,valor_x2a,5,1,0.24,0.5,True)
241     time.sleep(0.1)

243     at_x2 = am.ler(9)
244     #print("x2: ", valor_x2, "-", w5, "=", valor_x2a, "-", pa5, "- i =",i
245     , "| ",stp)
246     print("x2: ", valor_x2, "- st=",w5,"=====", valor_x2a, "- st=", at_x2)

247     # comunica o arduino Controla x1 por fs
248     valor_x1a =round( ea.get_sensores("X1"), 2)
249     am.escrever(3,3,10,at_x2,valor_x1a,6,1,0,5,True)
250     time.sleep(0.1)
251     at_x1 = am.ler(9)
252     ea.atualiza_atuador("fs", at_x1)
253     #print("x1: ", valor_x1, "-", w6, "=", valor_x1a, "-", pa6)
254     print("x1: ", valor_x1, "- fs=",w6,"=====", valor_x1a, "- fs=", at_x1)

255
256     except:
257         print('opa')
258         time.sleep(1)

259
260
261     # Calcula o modelo
262     e.modelo_evaporador()
263     if flag_arduino==1:
264         ea.modelo_evaporador()

265
266     # Selecion as refer n as para impressao
267     spx1.append(pid6.get_setpoint("X1"))
268     spx2.append(pid5.get_setpoint("X2"))
269     sptc.append(pid4.get_setpoint("Tc"))
270     spv1.append(pid1.get_setpoint("V1"))
271     spv2.append(pid2.get_setpoint("V2"))

272     #ser.flushOutput()
273     #ser.flushInput()
274     # if flag_arduino==1:
275     #     ser.close()

```

```

277     # ser1.close()

279 if flag==1:
    # impressao graficos
281 fig, (ax1, ax2) = plt.subplots(2, 1)
    # fig.suptitle('Volumes')
283
    ax1.set_xlabel('Tempo')
285     ax1.set_ylabel('Volume')
    ax1.plot(e.impressao("V1"), label="Volume 1")
287     ax1.plot(ea.impressao("V1"), label="Prot tipo")
    ax1.plot(spv1,':', label="Set Point")
289     ax1.legend()
    ax1.grid(True)
291
    ax2.set_xlabel('Tempo')
293     ax2.set_ylabel('Volume')
    ax2.plot(e.impressao("V2"), label="Volume 2")
295     ax2.plot(ea.impressao("V2"), label="Prot tipo")
    ax2.plot(spv2,':', label="Set Point")
297     ax2.legend()
    ax2.grid(True)
299
    fig, (ax3, ax4) = plt.subplots(2, 1)
    # fig.suptitle('Temperaturas')

303     ax3.set_xlabel('Tempo')
    ax3.set_ylabel('Temperatura')
305     ax3.plot(e.impressao("T1"), label="Temperatura 1")
    ax3.plot(ea.impressao("T1"), label="Prot tipo")
307     ax3.legend()
    ax3.grid(True)
309
    ax4.set_xlabel('Tempo')
311     ax4.set_ylabel('Temperatura')
    ax4.plot(e.impressao("T2"), label="Temperatura 2")
313     ax4.plot(ea.impressao("T2"), label="Prot tipo")
    ax4.legend()
315     ax4.grid(True)

317     fig, (ax5, ax6) = plt.subplots(2, 1)
    # fig.suptitle('Volumes')
319
    ax5.set_xlabel('Tempo')
321     ax5.set_ylabel('Concentra o')
    ax5.plot(e.impressao("X1"), label="Concentra o 1")
323     ax5.plot(ea.impressao("X1"), label="Prot tipo")

```

```

325     ax5.plot(spx1,':', label="Set Point")
326     ax5.legend()
327     ax5.grid(True)
328
329     ax6.set_xlabel('Tempo')
330     ax6.set_ylabel('Concentra o')
331     ax6.plot(e.impressao("X2"), label="Concentra o 2")
332     ax6.plot(ea.impressao("X2"), label="Prot tipo")
333     ax6.plot(spx2,':', label="Set Point")
334     ax6.legend()
335     ax6.grid(True)
336
337     fig, (ax7,ax8)=plt.subplots(2, 1)
338     ax7.set_xlabel('Tempo')
339     ax7.set_ylabel('Temperatura')
340     ax7.plot(e.impressao("TC"), label="Temp. Condesador")
341     ax7.plot(ea.impressao("TC"), label="Prot tipo")
342     ax7.plot(sptc,':', label="Set Point")
343     ax7.legend()
344     ax7.grid(True)
345
346     ax8.set_xlabel('Tempo')
347     ax8.set_ylabel('Transferencia do calor')
348     ax8.plot(e.impressao("Q1"), label="Q1")
349     ax8.plot(e.impressao("Q2"), label="Q2")
350     ax8.plot(e.impressao("QC"), label="QC")
351     # ax8.plot(ea.impressao("Q1"), label="Arduino Q1")
352     # ax8.plot(sptc, ':', label="Set Point")
353     ax8.legend()
354     ax8.grid(True)
355
356     fig, (ax9, ax10) = plt.subplots(2, 1)
357     ax9.set_xlabel('Tempo')
358     ax9.set_ylabel('enthalpy ')
359     ax9.plot(e.impressao("H1"), label="H1")
360     ax9.plot(ea.impressao("H1"), label="Prot tipo")
361     ax9.legend()
362     ax9.grid(True)
363
364     ax10.set_xlabel('Tempo')
365     ax10.set_ylabel('enthalpy ')
366     ax10.plot(e.impressao("H2"), label=" H2")
367     ax10.plot(ea.impressao("H2"), label="Prot tipo")
368     # ax9.plot(sptc, ':', label="Set Point")
369     ax10.legend()
370     ax10.grid(True)

```

```
371
373     plt.show()
375 \end{listing}
Controladores.py
377 \begin{lstlisting}
from typing import Dict, Any
379
class Controlador:
381
    __setpoint = 0
383    __direta = 1
384    __K = 3
385    __Ti = 2
386    __Td = 2
387    __min = 0
388    __max = 0
389    __sensor_max = 0
390
391 def __init__(self):
392     self.set_setpoint()
393     self.set_sensores()
394     self.set_tempo()
395     self.set_partes()
396     self.set_calculos()
397     self.set_atuadores()
398
399 def set_partes(self, i=0, d=0, sys=0):
400     self.__parte = {
401         "integral": i,
402         "derivativa": d,
403         "sistema": sys
404     }
405
406 def set_calculos(self, Tt=0.5, N=10):
407     self.__calculo = {
408         "Tt": Tt,
409         "N": N
410     }
411
412 def set_concentracoes(self, x0, x1, x2):
413     self.__concentracoes = {
414         "x0": x0,
415         "x1": x1,
416         "x2": x2
417     }
```

```

419     def get_concentracoess(self):
420         return self.__concentracoess
421
422     def set_temperturas(self, t0, ts, th2o):
423         self.__temperaturas = {
424             "t0": t0,
425             "ts": ts,
426             "th2o": th2o
427         }
428
429     def get_temperaturas(self):
430         return self.__temperaturas
431
432     def set_tempo(self, int=1):
433
434         self.__tempo = {
435             "int": int,
436             "ctl": int * 100
437         }
438
439     def get_tempo(self):
440
441         return self.__tempo["ctl"]
442
443     def set_setpoint(self, v1=3, v2=3, Tc=70, x2=0.44, x1=0):
444
445         self.__setpoint = {
446             "v1": v1,
447             "v2": v2,
448             "Tc": Tc,
449             "x2": x2,
450             "x1": x1
451         }
452
453     def atualiza_setpoint(self, chave, valor):
454
455         self.__setpoint[chave] = valor
456
457     def get_setpoint(self, chave):
458
459         return self.__setpoint[chave]
460
461     def set_sensores(self, v1=0, v2=0, Tc=0, x1=0, x2=0):
462
463         self.__sensores = {
464             "v1": v1,
465             "v2": v2,
466             "Tc": Tc,
467             "x1": x1,
468             "x2": x2
469         }

```

```

465     "V2": v2,
466     "TC": Tc,
467     "X1": X1,
468     "X2": X2
469 }
470
471 def get_sensores(self, chave):
472
473     return self.__sensores[chave]
474
475 def set_variavel(self, valor):
476
477     self.__variavel=valor
478
479 def get_variavel(self):
480
481     return self.__variavel
482
483 def set_atuadores(self, f0=0, f1=0, f2=0, fc=0, fs=0):
484
485     self.__atuador = {
486         "f0": f0,
487         "f1": f1,
488         "f2": f2,
489         "fc": fc,
490         "fs": fs
491     }
492
493 def get_atuadores(self, chave):
494
495     return self.__atuador[chave]
496
497 def limites_atuador(self, valor, min, max):
498
499     # min = limite inferior
500     # max = limite superior
501
502     if valor < min:
503         valor = min
504
505     elif valor > max:
506         valor = max
507
508     return valor
509
510 def get_erro(self, chave, direta):
511
512     if direta == 1:
513         erro = self.get_setpoint(chave) - self.get_sensores(chave)
514     else:

```

```

        erro = self.get_sensores(chave) - self.get_setpoint(chave)
513    return erro

515 def pid(self, chave, direta, valor, K=3, Ti=2, Td=0.2):

517    # Variaveis utilizadas
518    # ** Entrada
519    # --- uc - setpoint
520    # --- y   - resposta do sistema
521    # --- K   - ganho do controlador
522    # --- Ti  - tempo de integracao
523    # --- Td  - tempo de derivacao
524    # --- h   - periodo de amostragem
525    # --- yOld - ultimo valor da resposta do sistema
526    # --- IOld - ultimo valor da parte integral
527    # --- DOld - ultimo valor da parte derivativa

528    # ** Saída
529    # --- w   - saída do controlador PID limitada
530    # --- v   - saída do controlador PID
531    # --- INew - ultimo valor da parte integral
532    # --- DNew - ultimo valor da parte derivativa

533    # ** Calculo
534    # --- Tt  - tempo de 'tracking'
535    # --- N   - máximo ganho derivativo
536    # --- ulow - limite de saída inferior
537    # --- uhigh - limite de saída superior

538    # Variaveis de calculo
539    # Tt = self.__calculo["Tt"]*Ti*0.5
540    N = self.__calculo["N"]

541    # Periodo de amostragem
542    h=self.__tempo["int"]

543    # leitura sensor
544    self.__sensores[chave] = valor
545    y = self.get_sensores(chave)
546    yold = self.__parte["sistema"]
547    self.__parte["sistema"] = y

548    # Calculo do erro
549    erro = self.get_erro(chave, direta)

550    # Calculo da parte proporcional
551    P = K*erro

```

```

559     # Atualiza os dados referentes a parte integral
561     Iold = self.__parte["integral"]
562     I=Iold + (K*h/Ti)*erro
563     self.__parte["integral"] = I

565     # Calculo da parte derivativa
566     Dold = self.__parte["derivativa"]
567     D = (Td/(Td + N*h))*Dold - K*N*(Td/(Td + N*h))*(y - yold)
568     self.__parte["derivativa"] = D
569

571     pid = P + I + D;
572
573     return pid

```

### Evaporadores.py

```

import math
2
class Evaporadores:
4
    def __init__(self):
6        self.set_regime()
7        self.set_perturbacoes()
8        self.set_atuadores()
9        self.set_parametros()
10       self.set_sensores()
11       self.set_tempo_amostragem()
12       self.set_impressao()

14
15       def set_tempo_amostragem(self, valor=0.03):
16
17           self.__Temp_int = valor
18
19       def set_regime(self, M1=0, x1=0, h1=0, M2=0, x2=0, h2=0):
20
21           self.__regime = {
22               "M1": M1,      # total mass in evap 1 [ton] (m1 in kg (m1 = 1000*x(1)))
23               "x1": x1,      # TS composition in evap 1 [kg/kg]
24               "h1": h1,      # enthalpy in evap 1 [kJ/kg]
25               "M2": M2,      # total mass in evap 2 [ton] (m2 in kg (m2 = 1000*x(4)))
26               "x2": x2,      # TS composition in evap 2 [kg/kg]
27               "h2": h2       # enthalpy in evap 2 [kJ/kg]
28           }

```

```

30     def get_regime(self, chave):
31
32         if not chave:
33             return self.__regime
34         else:
35             return self.__regime[chave]
36
37     def set_perturbacoes(self, x0=0.2, T0=60, Ts=140, Tc=20):
38         self.__perturbacao = {
39             "x0": x0,           # FLUXO ENTRADA LICOR NEGRO
40             "T0": T0,           # TEMPERATURA ENTRADA LICOR NEGRO
41             "Ts": Ts,           # TEMPERATURA VAPOR
42             "Tc": Tc            # TEMPERATURA AGUA CONDENSADOR
43         }
44     def atualiza_perturbacoes(self, chave, valor):
45
46         self.__perturbacao[chave] = valor
47
48     def get_perturbacoes(self, chave):
49
50         return self.__perturbacao[chave]
51
52     def set_atuadores(self, f0=0.11, f1=0.0822, f2=0.0577, fc=0.2139, fs=5):
53         self.__atuador = {
54             "f0": f0,
55             "f1": f1,
56             "f2": f2,
57             "fc": fc,
58             "fs": fs
59         }
60     def atualiza_atuador(self, chave, valor):
61
62         self.__atuador[chave] = valor
63
64     def get_atuadores(self, chave):
65
66         return self.__atuador[chave]
67
68     def set_parametros(self, k1=4, k2=4, kc=5, A1=2900, A2=3300, Ac=3000, Cpc=4.18,
69                     rhoc=1000):
70
71         self.__parametro = {
72             "k1": k1,
73             "k2": k2,
74             "kc": kc,
75             "A1": A1,
76             "A2": A2,
77             "Ac": Ac,
78             "Cpc": Cpc,
79             "rhoc": rhoc
80         }
81
82     def atualiza_parametros(self, chave, valor):
83
84         self.__parametro[chave] = valor
85
86     def get_parametros(self, chave):
87
88         return self.__parametro[chave]
89
90     def set_tamano(self, tamano):
91
92         self.tamano = tamano
93
94     def get_tamano(self):
95
96         return self.tamano
97
98     def set_ciclo(self, ciclo):
99
100        self.ciclo = ciclo
101
102    def get_ciclo(self):
103
104        return self.ciclo
105
106    def set_ciclos(self, ciclos):
107
108        self.ciclos = ciclos
109
110    def get_ciclos(self):
111
112        return self.ciclos
113
114    def set_ciclo_minimo(self, ciclo_minimo):
115
116        self.ciclo_minimo = ciclo_minimo
117
118    def get_ciclo_minimo(self):
119
120        return self.ciclo_minimo
121
122    def set_ciclo_maximo(self, ciclo_maximo):
123
124        self.ciclo_maximo = ciclo_maximo
125
126    def get_ciclo_maximo(self):
127
128        return self.ciclo_maximo
129
130    def set_ciclo_inicial(self, ciclo_inicial):
131
132        self.ciclo_inicial = ciclo_inicial
133
134    def get_ciclo_inicial(self):
135
136        return self.ciclo_inicial
137
138    def set_ciclo_final(self, ciclo_final):
139
140        self.ciclo_final = ciclo_final
141
142    def get_ciclo_final(self):
143
144        return self.ciclo_final
145
146    def set_ciclo_ultimo(self, ciclo_ultimo):
147
148        self.ciclo_ultimo = ciclo_ultimo
149
150    def get_ciclo_ultimo(self):
151
152        return self.ciclo_ultimo
153
154    def set_ciclo_anterior(self, ciclo_anterior):
155
156        self.ciclo_anterior = ciclo_anterior
157
158    def get_ciclo_anterior(self):
159
160        return self.ciclo_anterior
161
162    def set_ciclo_proximo(self, ciclo_proximo):
163
164        self.ciclo_proximo = ciclo_proximo
165
166    def get_ciclo_proximo(self):
167
168        return self.ciclo_proximo
169
170    def set_ciclo_inicial(self, ciclo_inicial):
171
172        self.ciclo_inicial = ciclo_inicial
173
174    def get_ciclo_inicial(self):
175
176        return self.ciclo_inicial
177
178    def set_ciclo_final(self, ciclo_final):
179
180        self.ciclo_final = ciclo_final
181
182    def get_ciclo_final(self):
183
184        return self.ciclo_final
185
186    def set_ciclo_anterior(self, ciclo_anterior):
187
188        self.ciclo_anterior = ciclo_anterior
189
190    def get_ciclo_anterior(self):
191
192        return self.ciclo_anterior
193
194    def set_ciclo_proximo(self, ciclo_proximo):
195
196        self.ciclo_proximo = ciclo_proximo
197
198    def get_ciclo_proximo(self):
199
200        return self.ciclo_proximo
201
202    def set_ciclo_inicial(self, ciclo_inicial):
203
204        self.ciclo_inicial = ciclo_inicial
205
206    def get_ciclo_inicial(self):
207
208        return self.ciclo_inicial
209
210    def set_ciclo_final(self, ciclo_final):
211
212        self.ciclo_final = ciclo_final
213
214    def get_ciclo_final(self):
215
216        return self.ciclo_final
217
218    def set_ciclo_anterior(self, ciclo_anterior):
219
220        self.ciclo_anterior = ciclo_anterior
221
222    def get_ciclo_anterior(self):
223
224        return self.ciclo_anterior
225
226    def set_ciclo_proximo(self, ciclo_proximo):
227
228        self.ciclo_proximo = ciclo_proximo
229
230    def get_ciclo_proximo(self):
231
232        return self.ciclo_proximo
233
234    def set_ciclo_inicial(self, ciclo_inicial):
235
236        self.ciclo_inicial = ciclo_inicial
237
238    def get_ciclo_inicial(self):
239
240        return self.ciclo_inicial
241
242    def set_ciclo_final(self, ciclo_final):
243
244        self.ciclo_final = ciclo_final
245
246    def get_ciclo_final(self):
247
248        return self.ciclo_final
249
250    def set_ciclo_anterior(self, ciclo_anterior):
251
252        self.ciclo_anterior = ciclo_anterior
253
254    def get_ciclo_anterior(self):
255
256        return self.ciclo_anterior
257
258    def set_ciclo_proximo(self, ciclo_proximo):
259
260        self.ciclo_proximo = ciclo_proximo
261
262    def get_ciclo_proximo(self):
263
264        return self.ciclo_proximo
265
266    def set_ciclo_inicial(self, ciclo_inicial):
267
268        self.ciclo_inicial = ciclo_inicial
269
270    def get_ciclo_inicial(self):
271
272        return self.ciclo_inicial
273
274    def set_ciclo_final(self, ciclo_final):
275
276        self.ciclo_final = ciclo_final
277
278    def get_ciclo_final(self):
279
280        return self.ciclo_final
281
282    def set_ciclo_anterior(self, ciclo_anterior):
283
284        self.ciclo_anterior = ciclo_anterior
285
286    def get_ciclo_anterior(self):
287
288        return self.ciclo_anterior
289
290    def set_ciclo_proximo(self, ciclo_proximo):
291
292        self.ciclo_proximo = ciclo_proximo
293
294    def get_ciclo_proximo(self):
295
296        return self.ciclo_proximo
297
298    def set_ciclo_inicial(self, ciclo_inicial):
299
300        self.ciclo_inicial = ciclo_inicial
301
302    def get_ciclo_inicial(self):
303
304        return self.ciclo_inicial
305
306    def set_ciclo_final(self, ciclo_final):
307
308        self.ciclo_final = ciclo_final
309
310    def get_ciclo_final(self):
311
312        return self.ciclo_final
313
314    def set_ciclo_anterior(self, ciclo_anterior):
315
316        self.ciclo_anterior = ciclo_anterior
317
318    def get_ciclo_anterior(self):
319
320        return self.ciclo_anterior
321
322    def set_ciclo_proximo(self, ciclo_proximo):
323
324        self.ciclo_proximo = ciclo_proximo
325
326    def get_ciclo_proximo(self):
327
328        return self.ciclo_proximo
329
330    def set_ciclo_inicial(self, ciclo_inicial):
331
332        self.ciclo_inicial = ciclo_inicial
333
334    def get_ciclo_inicial(self):
335
336        return self.ciclo_inicial
337
338    def set_ciclo_final(self, ciclo_final):
339
340        self.ciclo_final = ciclo_final
341
342    def get_ciclo_final(self):
343
344        return self.ciclo_final
345
346    def set_ciclo_anterior(self, ciclo_anterior):
347
348        self.ciclo_anterior = ciclo_anterior
349
350    def get_ciclo_anterior(self):
351
352        return self.ciclo_anterior
353
354    def set_ciclo_proximo(self, ciclo_proximo):
355
356        self.ciclo_proximo = ciclo_proximo
357
358    def get_ciclo_proximo(self):
359
360        return self.ciclo_proximo
361
362    def set_ciclo_inicial(self, ciclo_inicial):
363
364        self.ciclo_inicial = ciclo_inicial
365
366    def get_ciclo_inicial(self):
367
368        return self.ciclo_inicial
369
370    def set_ciclo_final(self, ciclo_final):
371
372        self.ciclo_final = ciclo_final
373
374    def get_ciclo_final(self):
375
376        return self.ciclo_final
377
378    def set_ciclo_anterior(self, ciclo_anterior):
379
380        self.ciclo_anterior = ciclo_anterior
381
382    def get_ciclo_anterior(self):
383
384        return self.ciclo_anterior
385
386    def set_ciclo_proximo(self, ciclo_proximo):
387
388        self.ciclo_proximo = ciclo_proximo
389
390    def get_ciclo_proximo(self):
391
392        return self.ciclo_proximo
393
394    def set_ciclo_inicial(self, ciclo_inicial):
395
396        self.ciclo_inicial = ciclo_inicial
397
398    def get_ciclo_inicial(self):
399
400        return self.ciclo_inicial
401
402    def set_ciclo_final(self, ciclo_final):
403
404        self.ciclo_final = ciclo_final
405
406    def get_ciclo_final(self):
407
408        return self.ciclo_final
409
410    def set_ciclo_anterior(self, ciclo_anterior):
411
412        self.ciclo_anterior = ciclo_anterior
413
414    def get_ciclo_anterior(self):
415
416        return self.ciclo_anterior
417
418    def set_ciclo_proximo(self, ciclo_proximo):
419
420        self.ciclo_proximo = ciclo_proximo
421
422    def get_ciclo_proximo(self):
423
424        return self.ciclo_proximo
425
426    def set_ciclo_inicial(self, ciclo_inicial):
427
428        self.ciclo_inicial = ciclo_inicial
429
430    def get_ciclo_inicial(self):
431
432        return self.ciclo_inicial
433
434    def set_ciclo_final(self, ciclo_final):
435
436        self.ciclo_final = ciclo_final
437
438    def get_ciclo_final(self):
439
440        return self.ciclo_final
441
442    def set_ciclo_anterior(self, ciclo_anterior):
443
444        self.ciclo_anterior = ciclo_anterior
445
446    def get_ciclo_anterior(self):
447
448        return self.ciclo_anterior
449
450    def set_ciclo_proximo(self, ciclo_proximo):
451
452        self.ciclo_proximo = ciclo_proximo
453
454    def get_ciclo_proximo(self):
455
456        return self.ciclo_proximo
457
458    def set_ciclo_inicial(self, ciclo_inicial):
459
460        self.ciclo_inicial = ciclo_inicial
461
462    def get_ciclo_inicial(self):
463
464        return self.ciclo_inicial
465
466    def set_ciclo_final(self, ciclo_final):
467
468        self.ciclo_final = ciclo_final
469
470    def get_ciclo_final(self):
471
472        return self.ciclo_final
473
474    def set_ciclo_anterior(self, ciclo_anterior):
475
476        self.ciclo_anterior = ciclo_anterior
477
478    def get_ciclo_anterior(self):
479
480        return self.ciclo_anterior
481
482    def set_ciclo_proximo(self, ciclo_proximo):
483
484        self.ciclo_proximo = ciclo_proximo
485
486    def get_ciclo_proximo(self):
487
488        return self.ciclo_proximo
489
490    def set_ciclo_inicial(self, ciclo_inicial):
491
492        self.ciclo_inicial = ciclo_inicial
493
494    def get_ciclo_inicial(self):
495
496        return self.ciclo_inicial
497
498    def set_ciclo_final(self, ciclo_final):
499
500        self.ciclo_final = ciclo_final
501
502    def get_ciclo_final(self):
503
504        return self.ciclo_final
505
506    def set_ciclo_anterior(self, ciclo_anterior):
507
508        self.ciclo_anterior = ciclo_anterior
509
510    def get_ciclo_anterior(self):
511
512        return self.ciclo_anterior
513
514    def set_ciclo_proximo(self, ciclo_proximo):
515
516        self.ciclo_proximo = ciclo_proximo
517
518    def get_ciclo_proximo(self):
519
520        return self.ciclo_proximo
521
522    def set_ciclo_inicial(self, ciclo_inicial):
523
524        self.ciclo_inicial = ciclo_inicial
525
526    def get_ciclo_inicial(self):
527
528        return self.ciclo_inicial
529
530    def set_ciclo_final(self, ciclo_final):
531
532        self.ciclo_final = ciclo_final
533
534    def get_ciclo_final(self):
535
536        return self.ciclo_final
537
538    def set_ciclo_anterior(self, ciclo_anterior):
539
540        self.ciclo_anterior = ciclo_anterior
541
542    def get_ciclo_anterior(self):
543
544        return self.ciclo_anterior
545
546    def set_ciclo_proximo(self, ciclo_proximo):
547
548        self.ciclo_proximo = ciclo_proximo
549
550    def get_ciclo_proximo(self):
551
552        return self.ciclo_proximo
553
554    def set_ciclo_inicial(self, ciclo_inicial):
555
556        self.ciclo_inicial = ciclo_inicial
557
558    def get_ciclo_inicial(self):
559
560        return self.ciclo_inicial
561
562    def set_ciclo_final(self, ciclo_final):
563
564        self.ciclo_final = ciclo_final
565
566    def get_ciclo_final(self):
567
568        return self.ciclo_final
569
570    def set_ciclo_anterior(self, ciclo_anterior):
571
572        self.ciclo_anterior = ciclo_anterior
573
574    def get_ciclo_anterior(self):
575
576        return self.ciclo_anterior
577
578    def set_ciclo_proximo(self, ciclo_proximo):
579
580        self.ciclo_proximo = ciclo_proximo
581
582    def get_ciclo_proximo(self):
583
584        return self.ciclo_proximo
585
586    def set_ciclo_inicial(self, ciclo_inicial):
587
588        self.ciclo_inicial = ciclo_inicial
589
590    def get_ciclo_inicial(self):
591
592        return self.ciclo_inicial
593
594    def set_ciclo_final(self, ciclo_final):
595
596        self.ciclo_final = ciclo_final
597
598    def get_ciclo_final(self):
599
600        return self.ciclo_final
601
602    def set_ciclo_anterior(self, ciclo_anterior):
603
604        self.ciclo_anterior = ciclo_anterior
605
606    def get_ciclo_anterior(self):
607
608        return self.ciclo_anterior
609
610    def set_ciclo_proximo(self, ciclo_proximo):
611
612        self.ciclo_proximo = ciclo_proximo
613
614    def get_ciclo_proximo(self):
615
616        return self.ciclo_proximo
617
618    def set_ciclo_inicial(self, ciclo_inicial):
619
620        self.ciclo_inicial = ciclo_inicial
621
622    def get_ciclo_inicial(self):
623
624        return self.ciclo_inicial
625
626    def set_ciclo_final(self, ciclo_final):
627
628        self.ciclo_final = ciclo_final
629
630    def get_ciclo_final(self):
631
632        return self.ciclo_final
633
634    def set_ciclo_anterior(self, ciclo_anterior):
635
636        self.ciclo_anterior = ciclo_anterior
637
638    def get_ciclo_anterior(self):
639
640        return self.ciclo_anterior
641
642    def set_ciclo_proximo(self, ciclo_proximo):
643
644        self.ciclo_proximo = ciclo_proximo
645
646    def get_ciclo_proximo(self):
647
648        return self.ciclo_proximo
649
650    def set_ciclo_inicial(self, ciclo_inicial):
651
652        self.ciclo_inicial = ciclo_inicial
653
654    def get_ciclo_inicial(self):
655
656        return self.ciclo_inicial
657
658    def set_ciclo_final(self, ciclo_final):
659
660        self.ciclo_final = ciclo_final
661
662    def get_ciclo_final(self):
663
664        return self.ciclo_final
665
666    def set_ciclo_anterior(self, ciclo_anterior):
667
668        self.ciclo_anterior = ciclo_anterior
669
670    def get_ciclo_anterior(self):
671
672        return self.ciclo_anterior
673
674    def set_ciclo_proximo(self, ciclo_proximo):
675
676        self.ciclo_proximo = ciclo_proximo
677
678    def get_ciclo_proximo(self):
679
680        return self.ciclo_proximo
681
682    def set_ciclo_inicial(self, ciclo_inicial):
683
684        self.ciclo_inicial = ciclo_inicial
685
686    def get_ciclo_inicial(self):
687
688        return self.ciclo_inicial
689
690    def set_ciclo_final(self, ciclo_final):
691
692        self.ciclo_final = ciclo_final
693
694    def get_ciclo_final(self):
695
696        return self.ciclo_final
697
698    def set_ciclo_anterior(self, ciclo_anterior):
699
700        self.ciclo_anterior = ciclo_anterior
701
702    def get_ciclo_anterior(self):
703
704        return self.ciclo_anterior
705
706    def set_ciclo_proximo(self, ciclo_proximo):
707
708        self.ciclo_proximo = ciclo_proximo
709
710    def get_ciclo_proximo(self):
711
712        return self.ciclo_proximo
713
714    def set_ciclo_inicial(self, ciclo_inicial):
715
716        self.ciclo_inicial = ciclo_inicial
717
718    def get_ciclo_inicial(self):
719
720        return self.ciclo_inicial
721
722    def set_ciclo_final(self, ciclo_final):
723
724        self.ciclo_final = ciclo_final
725
726    def get_ciclo_final(self):
727
728        return self.ciclo_final
729
730    def set_ciclo_anterior(self, ciclo_anterior):
731
732        self.ciclo_anterior = ciclo_anterior
733
734    def get_ciclo_anterior(self):
735
736        return self.ciclo_anterior
737
738    def set_ciclo_proximo(self, ciclo_proximo):
739
740        self.ciclo_proximo = ciclo_proximo
741
742    def get_ciclo_proximo(self):
743
744        return self.ciclo_proximo
745
746    def set_ciclo_inicial(self, ciclo_inicial):
747
748        self.ciclo_inicial = ciclo_inicial
749
750    def get_ciclo_inicial(self):
751
752        return self.ciclo_inicial
753
754    def set_ciclo_final(self, ciclo_final):
755
756        self.ciclo_final = ciclo_final
757
758    def get_ciclo_final(self):
759
760        return self.ciclo_final
761
762    def set_ciclo_anterior(self, ciclo_anterior):
763
764        self.ciclo_anterior = ciclo_anterior
765
766    def get_ciclo_anterior(self):
767
768        return self.ciclo_anterior
769
770    def set_ciclo_proximo(self, ciclo_proximo):
771
772        self.ciclo_proximo = ciclo_proximo
773
774    def get_ciclo_proximo(self):
775
776        return self.ciclo_proximo
777
778    def set_ciclo_inicial(self, ciclo_inicial):
779
780        self.ciclo_inicial = ciclo_inicial
781
782    def get_ciclo_inicial(self):
783
784        return self.ciclo_inicial
785
786    def set_ciclo_final(self, ciclo_final):
787
788        self.ciclo_final = ciclo_final
789
790    def get_ciclo_final(self):
791
792        return self.ciclo_final
793
794    def set_ciclo_anterior(self, ciclo_anterior):
795
796        self.ciclo_anterior = ciclo_anterior
797
798    def get_ciclo_anterior(self):
799
800        return self.ciclo_anterior
801
802    def set_ciclo_proximo(self, ciclo_proximo):
803
804        self.ciclo_proximo = ciclo_proximo
805
806    def get_ciclo_proximo(self):
807
808        return self.ciclo_proximo
809
810    def set_ciclo_inicial(self, ciclo_inicial):
811
812        self.ciclo_inicial = ciclo_inicial
813
814    def get_ciclo_inicial(self):
815
816        return self.ciclo_inicial
817
818    def set_ciclo_final(self, ciclo_final):
819
820        self.ciclo_final = ciclo_final
821
822    def get_ciclo_final(self):
823
824        return self.ciclo_final
825
826    def set_ciclo_anterior(self, ciclo_anterior):
827
828        self.ciclo_anterior = ciclo_anterior
829
830    def get_ciclo_anterior(self):
831
832        return self.ciclo_anterior
833
834    def set_ciclo_proximo(self, ciclo_proximo):
835
836        self.ciclo_proximo = ciclo_proximo
837
838    def get_ciclo_proximo(self):
839
840        return self.ciclo_proximo
841
842    def set_ciclo_inicial(self, ciclo_inicial):
843
844        self.ciclo_inicial = ciclo_inicial
845
846    def get_ciclo_inicial(self):
847
848        return self.ciclo_inicial
849
850    def set_ciclo_final(self, ciclo_final):
851
852        self.ciclo_final = ciclo_final
853
854    def get_ciclo_final(self):
855
856        return self.ciclo_final
857
858    def set_ciclo_anterior(self, ciclo_anterior):
859
860        self.ciclo_anterior = ciclo_anterior
861
862    def get_ciclo_anterior(self):
863
864        return self.ciclo_anterior
865
866    def set_ciclo_proximo(self, ciclo_proximo):
867
868        self.ciclo_proximo = ciclo_proximo
869
870    def get_ciclo_proximo(self):
871
872        return self.ciclo_proximo
873
874    def set_ciclo_inicial(self, ciclo_inicial):
875
876        self.ciclo_inicial = ciclo_inicial
877
878    def get_ciclo_inicial(self):
879
880        return self.ciclo_inicial
881
882    def set_ciclo_final(self, ciclo_final):
883
884        self.ciclo_final = ciclo_final
885
886    def get_ciclo_final(self):
887
888        return self.ciclo_final
889
890    def set_ciclo_anterior(self, ciclo_anterior):
891
892        self.ciclo_anterior = ciclo_anterior
893
894    def get_ciclo_anterior(self):
895
896        return self.ciclo_anterior
897
898    def set_ciclo_proximo(self, ciclo_proximo):
899
900        self.ciclo_proximo = ciclo_proximo
901
902    def get_ciclo_proximo(self):
903
904        return self.ciclo_proximo
905
906    def set_ciclo_inicial(self, ciclo_inicial):
907
908        self.ciclo_inicial = ciclo_inicial
909
910    def get_ciclo_inicial(self):
911
912        return self.ciclo_inicial
913
914    def set_ciclo_final(self, ciclo_final):
915
916        self.ciclo_final = ciclo_final
917
918    def get_ciclo_final(self):
919
920        return self.ciclo_final
921
922    def set_ciclo_anterior(self, ciclo_anterior):
923
924        self.ciclo_anterior = ciclo_anterior
925
926    def get_ciclo_anterior(self):
927
928        return self.ciclo_anterior
929
930    def set_ciclo_proximo(self, ciclo_proximo):
931
932        self.ciclo_proximo = ciclo_proximo
933
934    def get_ciclo_proximo(self):
935
936        return self.ciclo_proximo
937
938    def set_ciclo_inicial(self, ciclo_inicial):
939
940        self.ciclo_inicial = ciclo_inicial
941
942    def get_ciclo_inicial(self):
943
944        return self.ciclo_inicial
945
946    def set_ciclo_final(self, ciclo_final):
947
948        self.ciclo_final = ciclo_final
949
950    def get_ciclo_final(self):
951
952        return self.ciclo_final
953
954    def set_ciclo_anterior(self, ciclo_anterior):
955
956        self.ciclo_anterior = ciclo_anterior
957
958    def get_ciclo_anterior(self):
959
960        return self.ciclo_anterior
961
962    def set_ciclo_proximo(self, ciclo_proximo):
963
964        self.ciclo_proximo = ciclo_proximo
965
966    def get_ciclo_proximo(self):
967
968        return self.ciclo_proximo
969
970    def set_ciclo_inicial(self, ciclo_inicial):
971
972        self.ciclo_inicial = ciclo_inicial
973
974    def get_ciclo_inicial(self):
975
976        return self.ciclo_inicial
977
978    def set_ciclo_final(self, ciclo_final):
979
980        self.ciclo_final = ciclo_final
981
982    def get_ciclo_final(self):
983
984        return self.ciclo_final
985
986    def set_ciclo_anterior(self, ciclo_anterior):
987
988        self.ciclo_anterior = ciclo_anterior
989
990    def get_ciclo_anterior(self):
991
992        return self.ciclo_anterior
993
994    def set_ciclo_proximo(self, ciclo_proximo):
995
996        self.ciclo_proximo = ciclo_proximo
997
998    def get_ciclo_proximo(self):
999
1000       return self.ciclo_proximo

```

```

76     "A2": A2,
77     "Ac": Ac,
78     "Cpc": Cpc,
79     "rhoc": rhoc
80 }
81
82 def get_parametros(self, chave):
83
84     return self.__parametro[chave]
85
86 def set_sensores(self, V1=0, X1=0, T1=0, V2=0, X2=0, T2=0):
87
88     self.__sensor = {
89         "V1": V1,
90         "X1": X1,
91         "T1": T1,
92         "V2": V2,
93         "X2": X2,
94         "T2": T2,
95     }
96
97 def get_sensores(self, chave):
98
99     if not chave:
100        return self.__sensor
101    else:
102        return self.__sensor[chave]
103
104 def atualiza_sensores(self):
105
106     # Atualiza sensor V1
107     m1 = self.__regime["M1"]*1000
108     x1 = self.__regime["x1"]
109     h1 = self.__regime["h1"]
110     t1 = self.tlbql(x1, h1)
111     self.__sensor["V1"] = m1/ self.dnslblq(t1,x1 )
112
113     # Atualiza sensor T1
114     self.__sensor["T1"] = t1
115
116     # # Atualiza sensor X1
117     self.__sensor["X1"] = x1
118
119     # Atualiza sensor V2
120     m2 = self.__regime["M2"]*1000
121     x2 = self.__regime["x2"]
122     h2 = self.__regime["h2"]

```

```

122     t2 = self.tlbql(x2,h2)
123     self.__sensor["V2"] = m2 / self.dnslblq(t2, x2)
124
125     # Atualiza sensor T2
126     self.__sensor["T2"] = t2
127
128     # # Atualiza sensor X2
129     self.__sensor["X2"] = x2
130
131     #Dados para impresso
132     self.__impressao["X1"].append(x1)
133     self.__impressao["X2"].append(x2)
134     self.__impressao["V1"].append(self.__sensor["V1"])
135     self.__impressao["V2"].append(self.__sensor["V2"])
136     self.__impressao["T1"].append(t1)
137     self.__impressao["T2"].append(t2)
138
139
140 def dnslblq(self, t, x):
141
142     # Density (m3/kg) for liquid black liqour as a function of
143     # temperature ( C ) and concentration (kg water / kg solution).
144     #
145     # dnslblq takes scalars as argument and givs an argument back.
146     #
147     # Constructed by Anders Gr nfors november 1997.
148
149     dnslblq = 1007.4 - 0.495 * t + 600 * x
150
151     return dnslblq
152
153 def entlbql(self, t, x):
154
155     # Enthalpy (kJ/kg) for liquid black liqour as a function of
156     # temperature ( C ) and concentration (kg water / kg solution).
157     #
158     # entlbql takes scalars as argument and givs an argument back.
159     #
160     # Constructed by Anders Gr nfors november 1997.
161
162     entlbql = (4.1868 - 2.261 * x) * t
163
164     return entlbql
165
166 def tlbql(self, x, h):
167
168     # Enthalpy (kJ/kg) for liquid black liqour as a function of

```

```

#
# entlblq takes scalars as argument and givs an argument back.
#
# Constructed by Anders Gr nfors november 1997.

174    tlbql = h / (4.1868 - 2.261 * x)

176    return tlbql

178 def eplh2o(self, P, T):
# EPLH2O Specific entalpy of liquid water, E = EPLH2O(P,T) (kJ/kg), as a function
180 # of pressure, P (kPa) and temperature T ( C ).

#
182 # Source: Ernst Schmidt, 1969, Properties of water and steam in SIunits,
# Springer-Verlag Berlin Heidelberg New York, R. Oldenbourg Munchen.
184 #

# Pressure range: 0.6 - 19000 kPa. Temperature range: 0 - 362 C.
186 # Max. error relative the International Skeleton Tables: 0.15 #
#
188 # The program uses the subprogram EQTH2O.

#
190 # Author : C. Engman , Dept. Chem. Eng. I, Lund, February 1983
# Revised: R. Colschen, Dept. Chem. Eng. I, Lund, August 1985
192 # Revised: R. Colschen, Dept. Chem. Eng. I, Lund, May 1987
# Revised: Stefan T rn, Dept. Chem. Eng. I, Lund, March 1988
194 # Revised: R. Colschen, Dept. Chem. Eng. I, Lund, May 1990
# Converted to MATLAB by:
196 # Anders Gr nfors, Dept. Chem. Eng. I, Lund 5 Mars 1995
# Revised: Anders Gr nfors, Dept. Chem. Eng. I, Lund, 19 December , 1995
198 # TooLowOrHigh = find((T < 0) | (T > 374) | (P < 0.61)...

200     t1=0.00154487872701993*T+0.421983624285494
201     t2=t1*t1
202     t3=t2*t1
203     t4=t2*t2
204     t6=t4*t2
205     t7=t4*t3
206     t8=t4*t4
207     t10=1/t6
208     t11=1.0-0.8438375405*t2-0.0005362162162*t10
209     t12=t11*t11
210     t14=math.sqrt(1.72*t12-0.00022685859691024*T-0.0619664257460328+0.44989682368897
#           e-5*P)
211     t15=1.0-0.8438375405*t2-0.0005362162162*t10+t14
212     t17=-0.00260725333075854*T-0.712171247296694+0.00321729729720/t7
213     t24=t15** (1/17)
214     t25=t24*t24

```

```

t26=t25*t25
216 t32=abs (0.231731805714506-0.00154487872701993*T)
t33=t32*t32
218 t34=t33*t33
t35=t34*t34
220 t38=t8*t8
t39=t38*t3
222 t43=abs (0.115e-5+t39) **2
t48=t8*t3
224 t51=(0.000015108+t48) **2
t54=P**2
226 t55=t54*P
t61=7.002753165+0.0000452079566003617*P
228 t62=t61*t61
t69=t54*t54
230 t71=739.301458788841*T+163920.466110549-0.276364606019155e7*t2+0.944280256739551
    e7*t3-0.208307675590731e8\
*t4+0.306822124630358e8*t4*t1
232 t72=-0.301196627013353e8*t6+0.189794973993922e8*t7-0.696083281226833e7*t8
    +0.113161485881604e7*t8*t1
t73=559.749606393127*(

234 t15*(-0.830459770114945+0.700773129897994*t2+0.000445305995637357*t10
    +0.58620689551725\
*t14+0.41666666667*t1*t17)+0.000113429298455121*T+0.0309832128730164-0.72*t1*
    t11*t17)/(

236 t26*t24)
t74=0.00317*(-0.02616571843-0.02284279054*t2+242.1647003*(

238 0.0139039085431794*T+4.45156804856945)*t35*t32\
+0.1269716088e-9*(20*t39+0.115e-5)/t43)*P
240 t75=-70.1204*((12*t48+0.000015108)*(

242 0.937992010849910e-11*P+0.444317439561950e-16*t54+0.102161218698085e-21\
*t55))/t51

244 t76=906.966657888536*t38*t2*(2.41196+19*t2)*(

246 1/t62/t61+0.135410710940326e-7*P)+0.172888641578764e-16*t55\
+0.37197019193547e-27*t69/t38/t4

248 epl=t71+t72+t73+t74+t75+t76
return epl
250

252 def epvh2o(self, P, T):
254
# EPVH2O Specific enthalpy of steam E = EPVH2O(P,T) H(kJ/kg), as a function
# of pressure, P(kPa), and temperature, T ( C ).

256 # Source: Ernst Schmidt, 1969, Properties of water and steam in SIunits,
# Springer-Verlag Berlin Heidelberg New York, R. Oldenbourg Munchen.

```

```

258 #
259 # Pressure range: 0.6 - 18000 kPa. Temperature range: 0 - 800 C.
260 # Max. error relative the International Skeleton Tables: 0.20 #
261 #
262 # The program uses the subprogram EQTH2O.
263 #
264 # Author : C Engman , Dept. Chem. Eng I, Lund, September 1982
265 # Revised: R Colschen , Dept. Chem. Eng I, Lund, August 1985
266 # Revised: R Colschen , Dept. Chem. Eng I, Lund, May 1987
267 # Revised: Stefan T rn , Dept. Chem. Eng I, Lund, March 1988
268 # Revised: R Colschen , Dept. Chem. Eng I, Lund, May 1990
269 # Converted to MATLAB by:
270 # Anders Gr nfors, Dept, Chem, Eng. I, Lund, 8 Mars 1995
271 # Revised: Anders Gr nfors, Dept. Chem. Eng. I, Lund, 21 December ,1995
272 # TSAT = self.eqth2o(P)
273 # TooLowOrHigh = find((T.*1.00010 < TSAT) | (P < 0.60)...
274 # | (P > 18000.0) | (T < 0.0) | (T >800))

275
276 t1=0.1544878727019929e-2*T+0.4219836242854936e0
277 t2=t1**2
278 t4=t2**2140
279 t11=math.exp(0.1323657500328418e1-0.3537772284721149e-2*T)
280 t15=P**2
281 t18=math.exp(0.7941945001970508e1-0.212266337083269e-1*T)
282 t19=(0.6798054997429492e1+0.212266337083269e-1*T)*t18
283 t22=math.exp(0.8824383335522787e0-0.2358514856480766e-2*T)
284 t26=math.exp(0.4412191667761393e0-0.1179257428240383e-2*T)
285 t30=t15*P
286 t37=t15**2
287 t44=math.exp(0.6177068334865951e1-0.1650960399536536e-1*T)
288 t48=t37*P
289 t59=math.exp(0.1058926000262734e2-0.283021782777692e-1*T)
290 t68=1/(0.4006073948e0*t44+0.23940900987136e18/t37)
291 t69=t1*t44*t68
292 t79=math.exp(0.8383164168746648e1-0.2240589113656728e-1*T)
293 t83=1/(0.8636081627e-1*t79+0.5295727298354483e22/t48)
294 t84=t1*t79*t83
295 t92=math.exp(0.2382583500591152e2-0.6367990112498069e-1*T)
296 t94=math.exp(0.1191291750295576e2-0.3183995056249034e-1*T)
297 t100=1/(-0.8532322921e0*t92+0.3460208861e0*t94
298 +0.1171414878396012e27/t37/t15)
299 t101=t1*(-0.460745437734e2*t92+0.93425639247e1*t94)*t100
300 t108=-0.1787039916770122e2+0.5967497936042021e-1*T
301 t110=0.1324291291154024e1-0.527894635872084e-1*T+0.1931380707e2*t2
302 t111=1/t110
303 t113=t108*t111
304 t145=t37**2

```

```

t148=t110**2
306 t149=t148**2
t150=t149**2
308 s1=0.1823801244130845e1*T+0.2500857472660724e4-0.3036678101852136e2*t2
+0.9182563265566376e2\
310 *t2*t1-0.180178197593935e2*t4-0.317e-2*p*(0.6670375918e-1*(0.5187484164810188e1\
312 +0.1533034656712498e-1*T)*math.exp(0.5735849168089812e1-0.1533034656712498e-1*T) \
314 +0.1388983801e1*(0.1966342499571582e1+0.3537772284721149e-2*T)*t11) \
316 -0.1433092224231465e-6*t15*(0.8390104328e-1*t19+0.2614670893e-1*(0.1644228333047721e1 \
318 +0.2358514856480766e-2*T)*t22-0.3373439453e-1*(0.1322114166523861e1 \
320 +0.1179257428240383e-2*T)*t26)

322 s3=s1-0.6478717107737182e-11*t30*(0.4520918904e0*t19+0.1069036614e0*(0.4221141665238607e1 \
324 +0.1179257428240383e-1*T)*math.exp(0.4412191667761393e1-0.1179257428240383e-1*T))
326 s2=s3-0.2928895618326032e-15*t37*(-0.5975336707e0*(0.9052854163096516e1 \
328 +0.2948143570600958e-1*T)*math.exp(0.1103047916940348e2-0.2948143570600958e-1*T) \
330 -0.8847535804e-1*(0.5509598331334049e1+0.1650960399536536e-1*T)*t44) \
332 -0.1324093860002727e-19*t48*(0.5958051609e0*(0.1130765332876354e2 \
334 +0.3773623770369226e-1*T)*math.exp(0.1411901333683646e2-0.3773623770369226e-1*T) \
336 -0.5159303373e0*(0.100191966626681e2+0.3301920799073073e-1*T) *
338 math.exp(0.123541366697319e2-0.3301920799073073e-1*T)+0.2075021122e0*
(
340 0.8730739996572656e1+0.283021782777692e-1*T)*t59)

342 s3=s2-0.701204e2*(0.1190610271e0*math.exp(0.5294630001313672e1-0.141510891388846e-1*T) \
344 *(0.4865369998286328e1+0.141510891388846e-1*T-0.4281157692242383e1*t69) \
-0.9867174132e-1* \
346 *math.exp(0.4853410834537533e1-0.1297183171064421e-1*T)*(0.4543255831762467e1 \
348 +0.1297183171064421e-1*T-0.4281157692242383e1*t69))*t68

```

```

350     s4=s3-0.701204e2*(  

351         0.1683998803e0*t59*(0.8730739996572656e1+0.283021782777692e-1*T\  

352         -0.1252519705247871e1*t84)-0.5809438001e-1*t18*(  

353         0.6798054997429492e1+0.212266337083269e-1\  

354         *T-0.1252519705247871e1*t84))*t83  

356  

356     s5=s4  

358  

358     s7=-0.701204e2*(  

359         0.6552390126e-2*t59*(0.8730739996572656e1+0.283021782777692e-1*T\  

360         -0.7633333333e0*t101)+0.5710218649e-3*t44*(  

361         0.5509598331334049e1+0.1650960399536536e-1*T\  

362         -0.7633333333e0*t101))*t100  

364  

364     s8=0.1130337239540422e-45*(  

365         0.1936587558e3+0.1936587558e4*t1*t108*t111-0.1388522425e4\  

366         *(0.1e1+t1*(  

367             0.1e2*t113+0.7633333333e0))*t26+0.4126607219e4*(  

368             0.1e1+t1*(0.1e2*t113\  

369             +0.15266666666e1))*t22-0.6508211677e4*(  

370             0.1e1+t1*(0.1e2*t113+0.22899999999e1))*t11\  

371             +0.5745984054e4*(  

372                 0.1e1+t1*(0.1e2*t113+0.3053333332e1))*math.exp(  

373                     0.1764876667104557e1\  

374                     -0.4717029712961533e-2*T)-0.2693088365e4*(  

375                         0.1e1+t1*(0.1e2*t113+0.38166666665e1))\  

376                         *math.exp(  

377                             0.2206095833880697e1-0.5896287141201916e-2*T)+0.5235718623e3*(  

378                             0.1e1+t1\  

379                             *(0.1e2*t113+0.45799999998e1))*math.exp(  

380                             0.2647315000656836e1-0.7075544569442299e-2*T))\  

381                             *t145*t30/t150/t148  

382  

382     s6=s7+s8  

384  

384     epvh2o=s5+s6  

385     return epvh2o  

386  

388     def eqpb1q(self, T, X):  

389  

390         # Equilibrium pressure(kPa) for liquid black liqour as a function of  

391         # temperature ( C ) and concentration (kg water / kg solution).  

392         #  

393         # eqpb1q takes scalars as argument and givs an argument back.  

394         #  

395         # Constructed by Anders Gr nfors november 1997.  

396         #

```

```

398     a = -1.794212795739585
400     b = -0.2918880381183214
402     c = 2.433040569149102
404     d = -1.720704009754379
406     e = 0.6476654714484422
408     f = -4.072742538773348
410     g = 6.56473933777589
412     h = -3.844316226130731
414     j = 0.1112786142523355

416     eqp = math.exp(a+b*T+c*T/math.log(T)+d*T**0.5+e*X**1.5+f*X**2+g*X**2.5+h*X
418             **3+j/math.log(X))
420     # eqp=      exp(a+b*T+c*T/      log(T)+d*T^(0.5)+e*X^(1.5)+f*X^2+ g*X^(2.5)+ h*X
422             ^3+j/      log(X))
424     return eqp

426

428     def eqph2o(self, T):
430
432         # EQPH2O Saturation pressure P = EQPH2O(T) (kPa) of water at temperature T( C ).
434         #
436         # Source: Ernst Schmidt, 1969, Properties of water and steam in SIunits,
438         # Springer-Verlag Berlin Heidelberg New York, R. Oldenbourg Munchen.
440         #
442         # Temperature range: 0 - 374 C. Maximum relative error compared
444         # with the International Skeleton Tables from 1963: 0.05 #.
446         #
448         # Author : C. Engman , Dept. Chem. Eng. I, Lund, March 1981
450         # Revised: Rolf Colschen, Dept. Chem. Eng. I, Lund, August 1985
452         # Revised: Rolf Colschen, Dept. Chem. Eng. I, Lund, May 1987
454         # Revised: Stefan T rn , Dept. Chem. Eng. I, Lund, March 1988
456         # Revised: Rolf Colschen, Dept. Chem. Eng. I, Lund, May 1990
458         # Converted to MATLAB by:
460         # Anders Grnfors, Dept. Chem. Eng. I, Lund, 5 Mars , 1995
462         # Revised: Anders Grnfors, Dept. Chem. Eng. I, Lund, 26 July , 1995
464
466         t1=0.578016375714506410-0.00154487872701992894*T
468         t2=t1*t1
470         t4=t2*t2
472         t5=0.340866205048354704e1-0.643769090066429787e-2*T+0.209750676e2*t2
474         t6=(0.154487872701992894e-2*T+0.42198362428549359)-t1/(0.1e10*t2+0.6e1)
476         eqph2o=0.2212e5*math.exp((-0.444565952745342190e1+0.118820246624439982e-1*T
478             -0.2608023696e2*t2\
480             -0.1681706546e3*t2*t1+0.6423285504e2*t4-0.1189646225e3*t4*t1)/t5/t6)
482         return eqph2o

484     def eqth2o(self, P):

```

```

442 # EQTH2O Saturation temperature, T = EQTH2O(P) (C) of liquid water
443 # at pressure P (kPa).
444 #
445 # Source: Ernst Schmidt, 1969, Properties of water and steam in SIunits,
446 # Springer-Verlag Berlin Heidelberg New York, R. Oldenbourg Munchen.
447 #
448 # The program is an Chebyshev series approximation of the
449 # inverce of the subprogram EQPH2O.
450 #
451 # Pressure range: 0.61 - 22100 kPa. Maximum relative error compared
452 # with the International Skeleton Tables from 1963: 0.04#
453 #
454 # Author : C. Engman , Dept. Chem. Eng. I, Lund, November 1982
455 # Revised: R. Colschen, Dept. Chem. Eng. I, Lund, August 1985
456 # Revised: R. Colschen, Dept. Chem. Eng. I, Lund, May 1987
457 # Revised: Stefan T rn, Dept. Chem. Eng. I, Lund, March 1988
458 # Converted to MATLAB by:143
459 # Anders Gr nfors Dept. Chem. Eng. I, Lund, 5 Mars 1995
460 # Revised: Anders Gr nfors Dept. Chem. Eng. I, Lund, 26 July 1995
461 # TooLowOrHigh = find((P < 0.61) | (P > 22100))
462
463     t1=math.log(0.4520795660036166E-4*B)
464     t2=0.1888400223132171*t1+0.1E1
465     t3=t2*t2
466     t4=t3*t2
467     t5=t3*t3
468     t6=t5*t4
469     t7=t5*t5
470     t9=t5*t2
471     t13=t5*t3
472     t14=+0.284512853142352E2*t1-0.6916844869689344E1*t7*t13+0.1274756995590758E2*t7*
473         t5-0.9465572679331328E1*t7\
474     *t3+0.287775527065216E1*t7+0.902245392406144*t13+0.1459587761857318E2*t5
475         +0.6913856612901767E2*t3
476     eqth2o=0.2532157687873905E3-0.1024333779003474E1*t7*t6-0.5888872353188045E1*t7*
477         t9+0.1436069002308024E2\
478         *t7*t4-0.1259342127811501E2*t7*t2+0.4896246302433824E1*t6+0.460292856119901E1*t9
479         \
480     +0.3269971037796052E2*t4+t14
481     return eqth2o
482
483 def modelo_evaporador(self):
484
485     # States: M1 = x(1) = total mass in evap 1 [ton]
486     # (m1 in kg (m1 = 1000*x(1)))
487     # x1 = x(2) = TS composition in evap 1 [kg/kg]

```

```

484 # h1 = x(3) = enthalpy in evap 1 [kJ/kg]
485 # M2 = x(4) = total mass in evap 2 [ton]
486 # (m2 in kg (m2 = 1000*x(4)))
487 # x2 = x(5) = TS composition in evap 2 [kg/kg]
488 # h2 = x(6) = enthalpy in evap 2 [kJ/kg]

490 # Propriedades fisicas
491 T0 = self.__perturbacao["T0"]
492 x0 = self.__perturbacao["x0"]
493 rho0 = self.dnslblq(T0,x0)
494 h0 = self.entlbql(T0,x0)

496 # Autadores
497 f0 = self.__atuador["f0"]
498 f1 = self.__atuador["f1"]
499 f2 = self.__atuador["f2"]
500 fc = self.__atuador["fc"]
501 fs = self.__atuador["fs"]

502 # Evaporador 1
503 m1 = self.__regime["M1"]*1000
504 x1 = self.__regime["x1"]
505 h1 = self.__regime["h1"]
506 Ts = self.__perturbacao["Ts"]
507 k1 = self.__parametro["k1"]
508 A1 = self.__parametro["A1"]

510 # Prorpiedade do licor
511 T1 = self.tlbql(x1, h1)
512 rho1 = self.dnslblq(T1, x1)
513 V1 = m1/rho1

516 # Propriedade do vapor
517 p1 = self.eqpblq(T1, x1)
518 Tc1 = self.eqth2o(p1)
519 pc1 = self.eqph2o(Tc1)
520 hs1 = self.epvh2o(pc1,Tc1)
521 h11 = self.eplh2o(pc1,Tc1)

522 # Transferencia do calor
523 q1 = k1 * A1 * (Ts - T1)
524 pcs = self.eqph2o(Ts)
525 hs0 = self.epvh2o(pcs,Ts)
526 h10 = self.eplh2o(pcs,Ts)
527 ws = q1 / (hs0 - h10)
528 # ws = ws *fs
529
530

```

```

# Evaporador 2
532 m2 = self.__regime["M2"]*1000
      x2 = self.__regime["x2"]
534 h2 = self.__regime["h2"]
      k2 = self.__parametro["k2"]
536 A2 = self.__parametro["A2"]

538 # Propriedade do licor
      T2 = self.tlbql(x2, h2)
540 rho2 = self.dnslblq(T2, x2)
      V2 = m2/rho2
542

# Propriedade do vapor
544 p2 = self.eqpbqlq(T2, x2)
      Tc2 = self.eqth2o(p2)
546 pc2 = self.eqph2o(Tc2)
      hs2 = self.epvh2o(pc2, Tc2)
548 hl2 = self.eplh2o(pc2, Tc2)

550 # Transferencia do calor
      q2 = k2 * A2 * (Tc1 - T2)
552 w1 = q2 / (hs1 - hl1)

554 # Condensador
      kc = self.__parametro["kc"]
556 Cpc = self.__parametro["Cpc"]
      Ac = self.__parametro["Ac"]
558 rhoc = self.__parametro["rhoc"]
      Tcin = 20
560

# Transferencia do calor
562 Tc = (kc * Ac * Tc2 + fc * rhoc * Cpc * Tcin) / (kc * Ac + fc * rhoc * Cpc)
      qc = kc * Ac * (Tc2 - Tc)
564 w2 = qc / (hs2 - hl2)

566 self.__perturbacao["Tc"] = Tc

568 # Atualizando regime
      _m1 = (rho0*f0 - rho1*f1 - w1 - fs*ws)/1000
570 # _x1 = (rho0*f0*(x0 - x1) + x1*w1 + x0*ws)/m1
      _x1 = (rho0*f0*(x0 - x1) + fs*(x1*w1 + x0*ws))/m1
572

      _h1 = (rho0*f0*(h0-h1)+fs*ws*(h0)-w1*(hs1-h1)+q1)/m1
574
      _m2 = (rho1*f1 - rho2*f2 - w2 - fs*ws)/1000
576 # _x2 = (rho1*f1*(x1 - x2) + x2*w2 + x0*ws)/m2
      _x2 = (rho1*f1*(x1 - x2) + fs*(x2*w2 + x0*ws))/m2

```

```

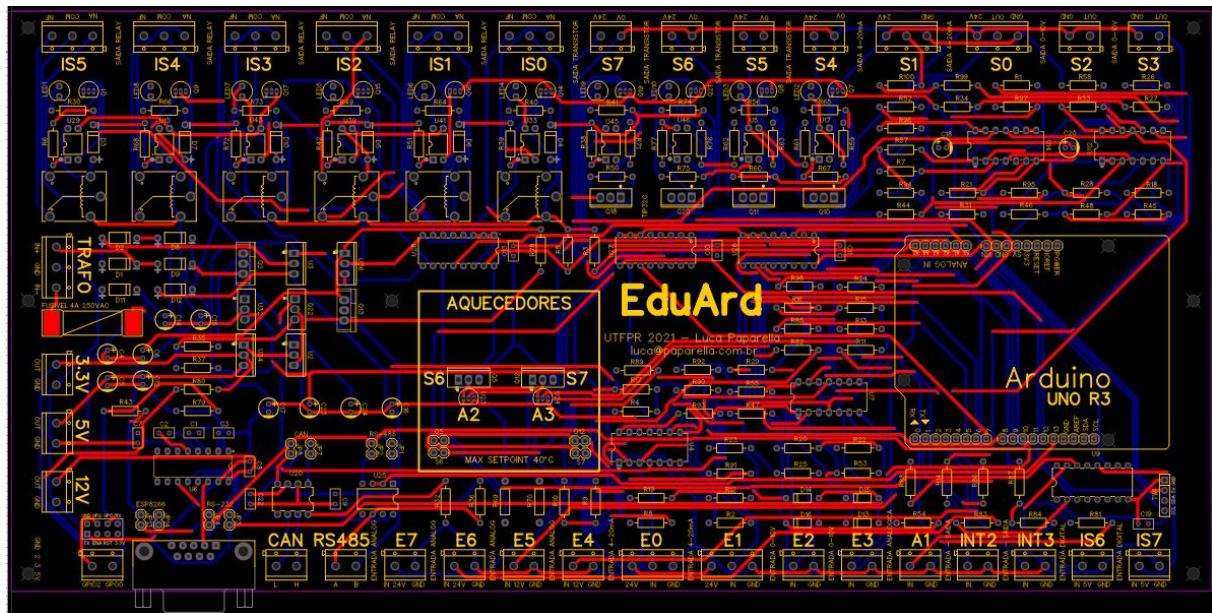
578     _h2 = (rho1*f1*(h1-h2)+fs*ws*(h0)-w2*(hs2-h2)+q2)/m2
580
580     # Atualizando regime
581     Temp_int = self.__Temp_int
582     self.__regime["M1"] += _m1*Temp_int
583     self.__regime["x1"] += _x1*Temp_int
584     self.__regime["h1"] += _h1*Temp_int
585     self.__regime["M2"] += _m2*Temp_int
586     self.__regime["x2"] += _x2*Temp_int
587     self.__regime["h2"] += _h2*Temp_int
588
588     # Dados para impressao
589     self.__impressao["T0"].append(self.__perturbacao["T0"])
590     self.__impressao["X0"].append(self.__perturbacao["x0"])
591
592     self.__impressao["F0"].append(self.__atuador["f0"])
593     self.__impressao["F1"].append(self.__atuador["f1"])
594     self.__impressao["F2"].append(self.__atuador["f2"])
595     self.__impressao["FC"].append(self.__atuador["fc"])
596     self.__impressao["FS"].append(self.__atuador["fs"])
597
598     self.__impressao["TC"].append(self.__perturbacao["Tc"])
599
600     self.__impressao["P1"].append(p1)
601     self.__impressao["P2"].append(p2)
602
603     self.__impressao["Q1"].append(q1)
604     self.__impressao["Q2"].append(q2)
605     self.__impressao["QC"].append(qc)
606
607     self.__impressao["H1"].append(_h1)
608     self.__impressao["H2"].append(_h2)
609
610     # Atualizando sensores
611     self.atualiza_sensores()
612
613     def set_impressao(self):
614
615         self.__impressao = {
616             "V1" : [],
617             "V2" : [],
618             "T0" : [],
619             "T1" : [],
620             "T2" : [],
621             "TC" : [],
622             "TS" : [],
623             "TH2O": []
624         }

```

```
626     "X0" : [],
627     "X1" : [],
628     "X2" : [],
629     "P1" : [],
630     "P2" : [],
631     "PC" : [],
632     "F0" : [],
633     "F1" : [],
634     "F2" : [],
635     "FC" : [],
636     "FS" : [],
637     "Q1" : [],
638     "Q2" : [],
639     "QC" : [],
640     "H1" : [],
641     "H2" : [],
642 }
643
644     def impressao(self, chave):
645
646         return self.__impressao[chave]
```

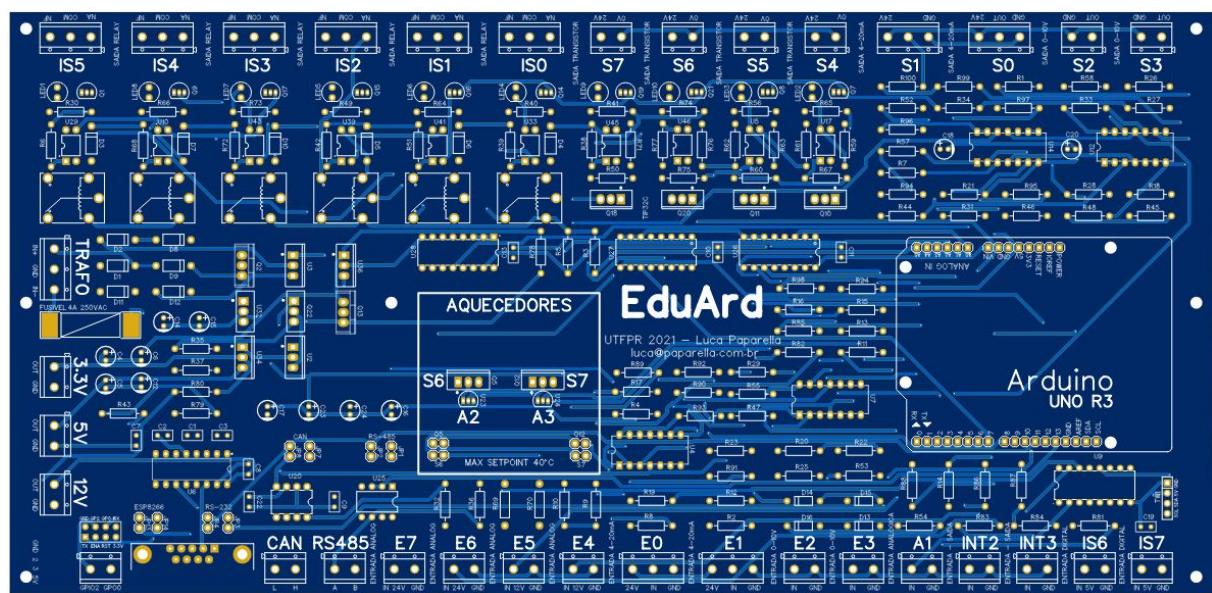
## APÊNDICE B – ARQUIVOS DO PROTÓTIPO HARDWARE

Figura 58 – Vista da conexões.



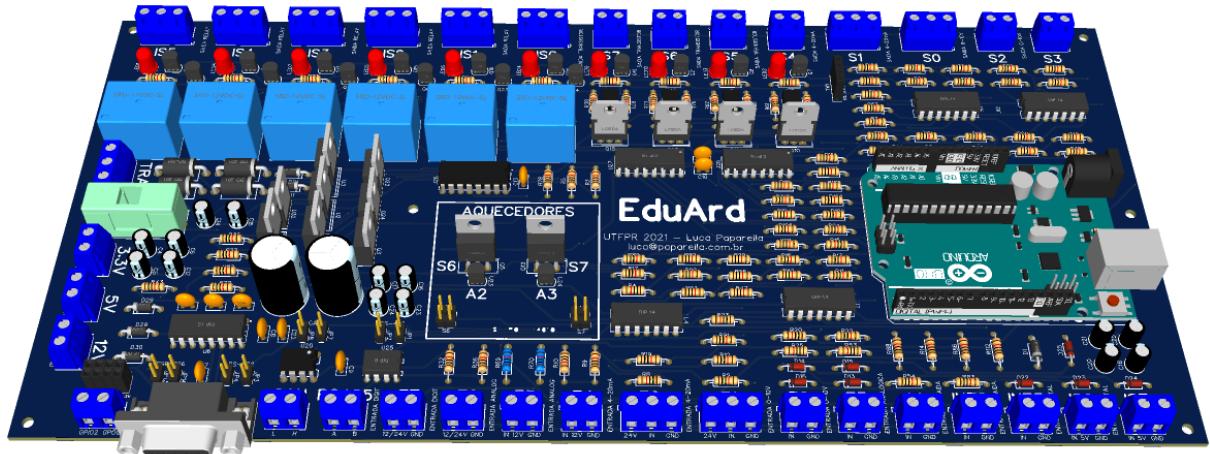
Fonte: Autoria própria

Figura 59 – Vista 2D da placa.



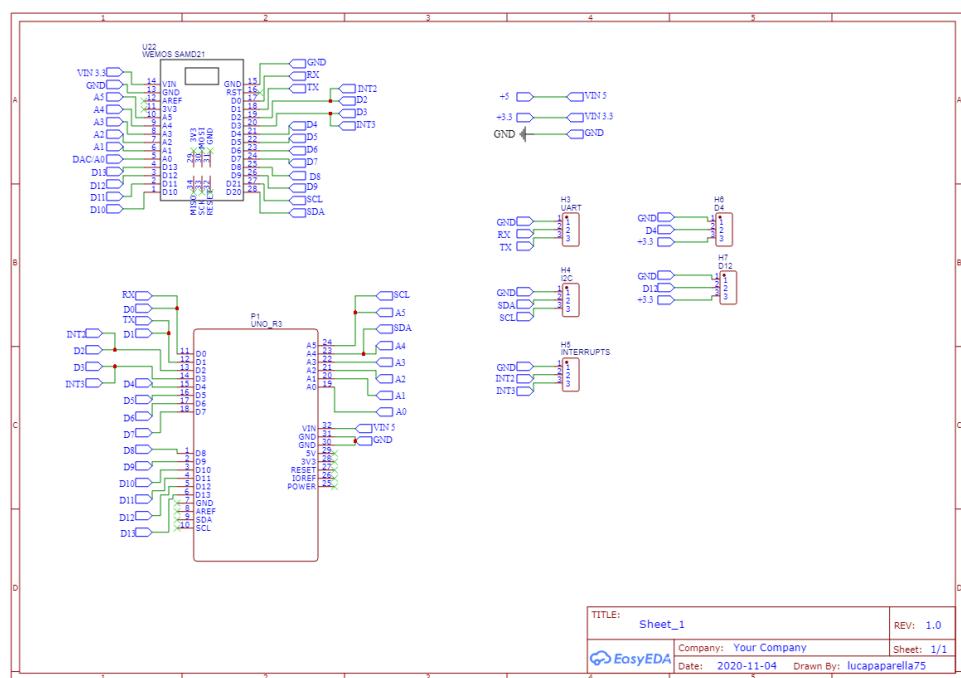
Fonte: Autoria própria

**Figura 60 – Vista 3D do protótipo.**

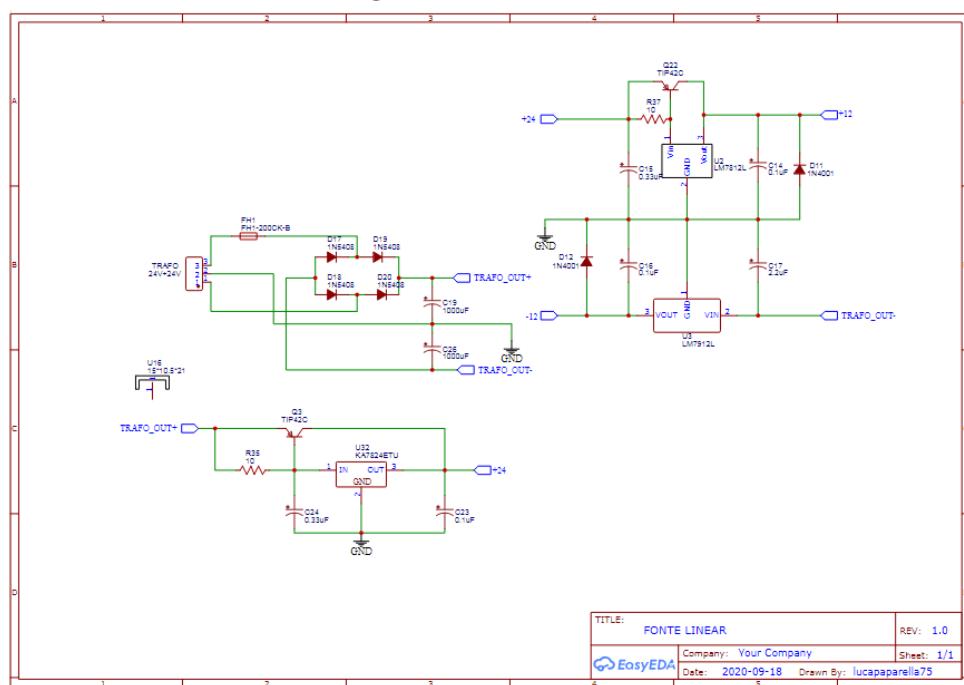
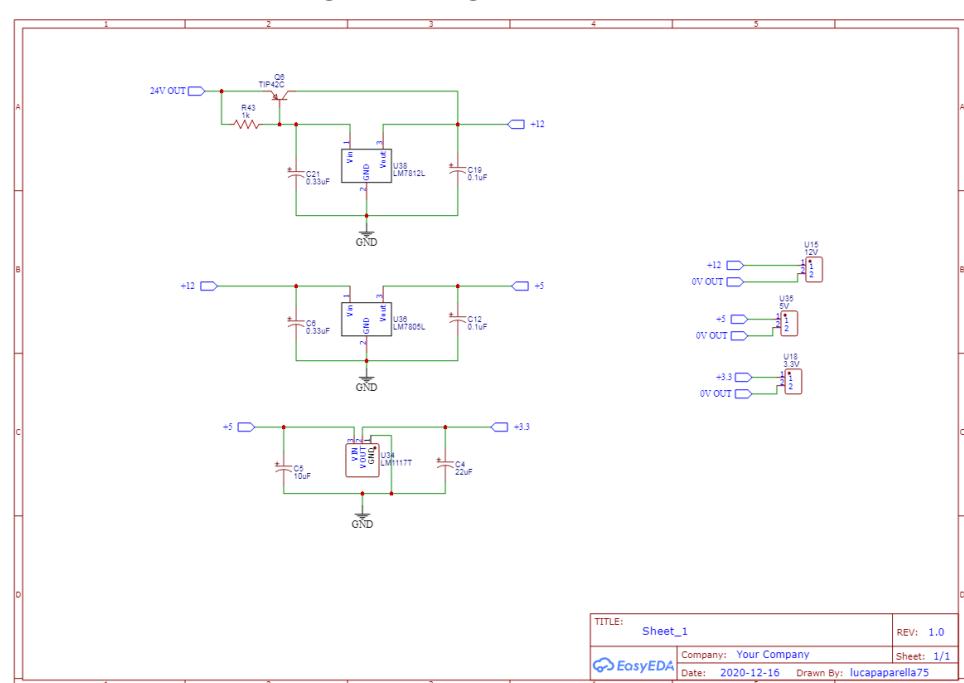


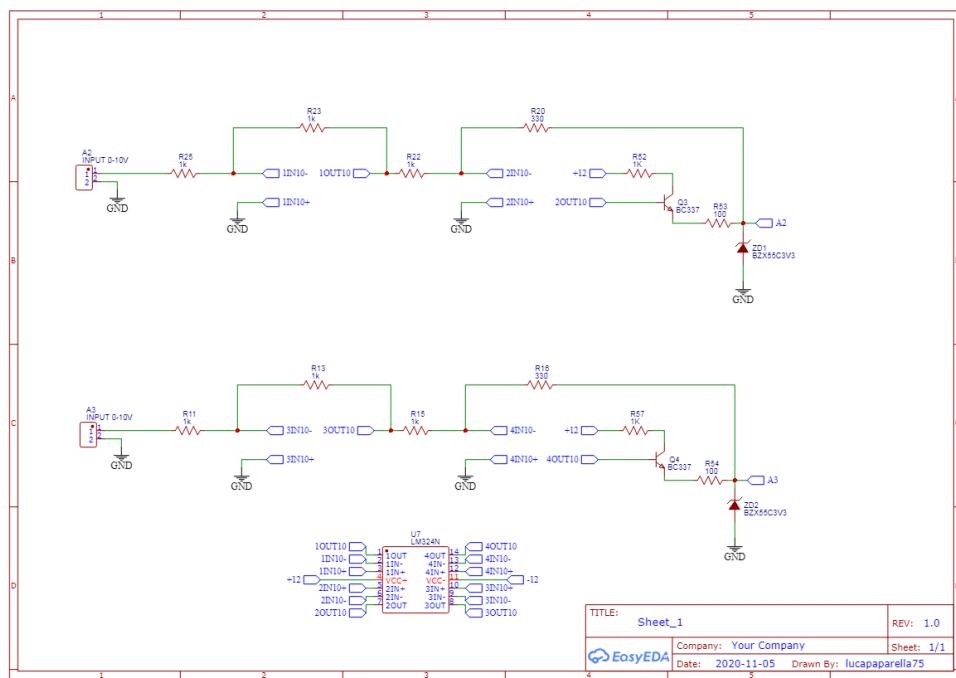
Fonte: Autoria própria

**Figura 61 – Conexões do microcontrolador.**

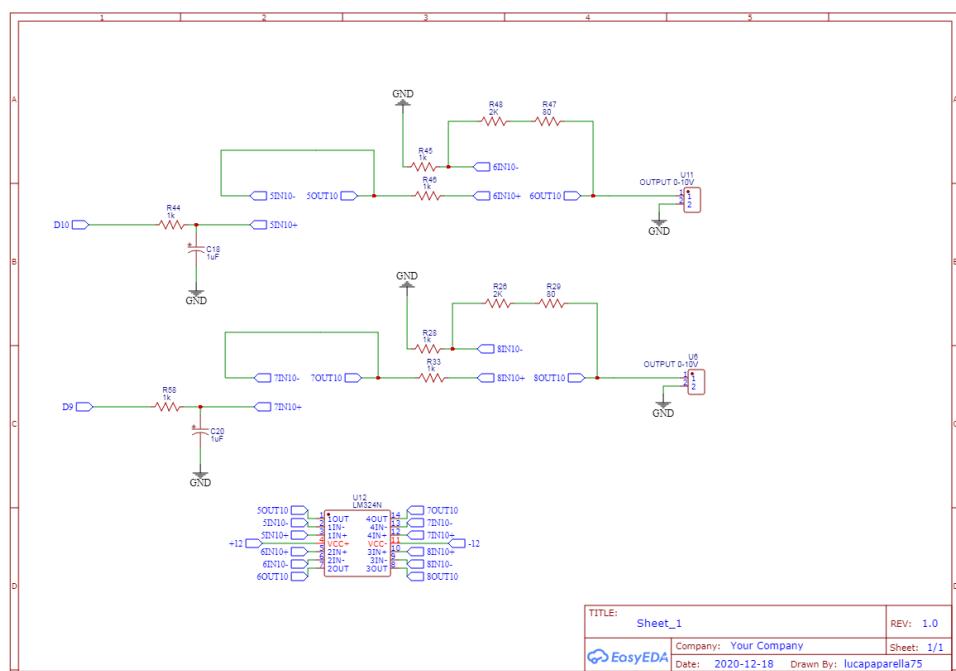


Fonte: Autoria própria

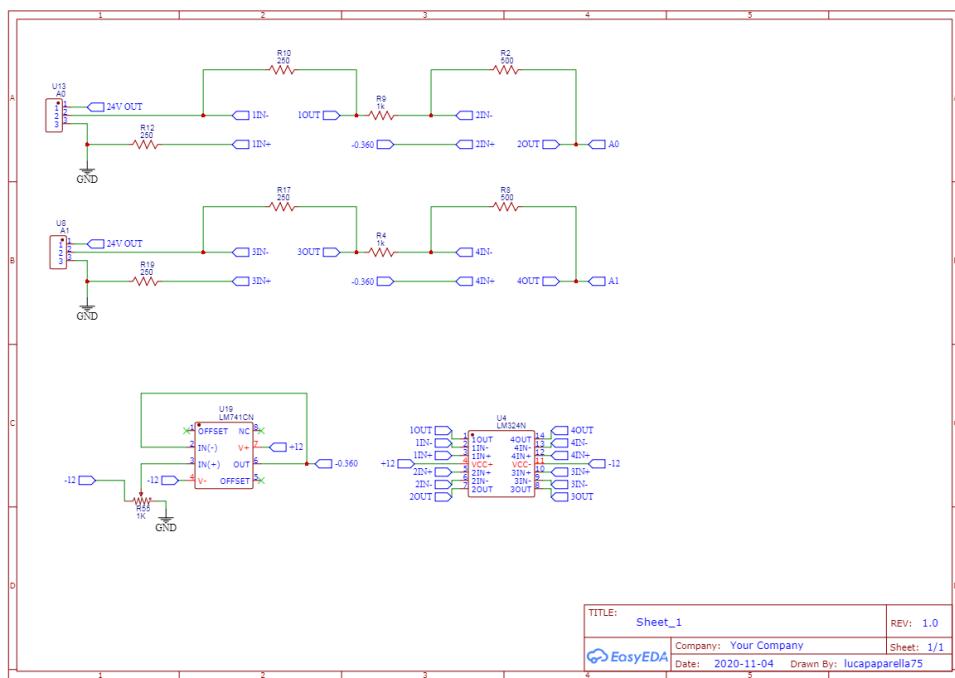
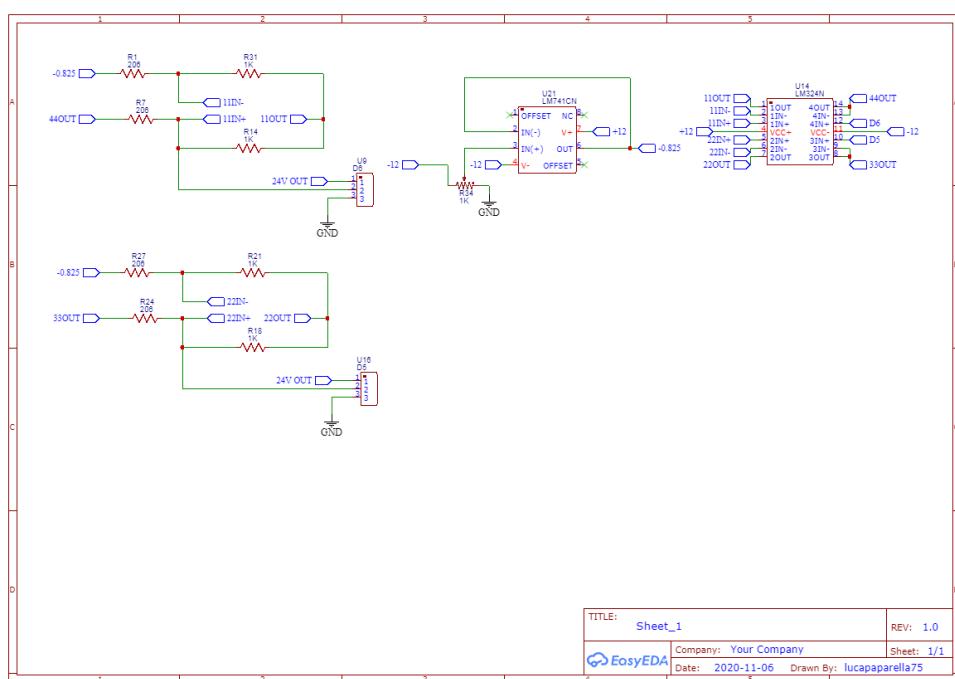
**Figura 62 – Fonte Linear.****Fonte: Autoria própria****Figura 63 – Reguladores da fonte.****Fonte: Autoria própria**

**Figura 64 – Entradas 0-10V.**

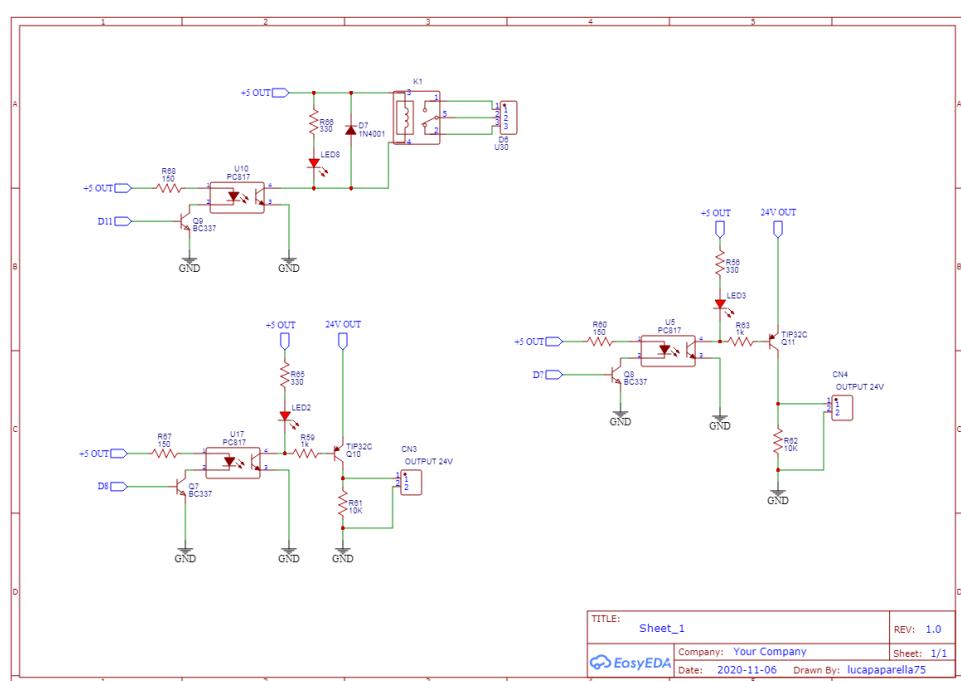
Fonte: Autoria própria

**Figura 65 – Saídas 0-10V.**

Fonte: Autoria própria

**Figura 66 – Entrada 4 a 20mA.****Fonte: Autoria própria****Figura 67 – Saídas 4 a 20mA.****Fonte: Autoria própria**

**Figura 68 – Saídas transistor e relé.**



**Fonte: Autoria própria**