UNIVERSIDADE ESTADUAL DE MARINGÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA QUÍMICA

PROGAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA QUÍMICA

VICTOR HUGO DA SILVA BARLATTI

MÓDULO EXPERIMENTAL PARA MODELAGEM, SIMULAÇÃO, SINTONIA E ANÁLISE DE DESEMPENHO DE SISTEMAS DE CONTROLE

Maringá-PR-Brasil

Novembro de 2018

VICTOR HUGO DA SILVA BARLATTI

MÓDULO EXPERIMENTAL PARA MODELAGEM, SIMULAÇÃO, SINTONIA E ANÁLISE DE DESEMPENHO DE SISTEMAS DE CONTROLE

Dissertação apresentada ao Programa de Pós-Graduação em Engenharia Química da Universidade Estadual de Maringá como parte dos requisitos necessários para a obtenção do título de Mestre em Engenharia Química, área de Modelagem, Controle e Automação de Processos.

Orientador: Prof. Dr. Cid Marcos Gonçalves Andrade
Co-Orientador: Prof. Dr. Marcos de Souza

Maringá-PR-Brasil

Novembro de 2018

Dados Internacionais de Catalogação na Publicação (CIP)
(Biblioteca Central - UEM, Maringá, PR, Brasil)

B257m

Barlatti, Victor Hugo da Silva
Módulo experimental para modelagem, simulação, sintonia e análise de desempenho de sistemas de controle / Victor Hugo da Silva Barlatti. -- Maringá, 2018.
57 f. : il. color., figs., tabs.

Orientador: Prof. Dr. Cid Marcos Gonçalves Andrade.
Coorientador: Prof. Dr. Marcos de Souza.
Dissertação (mestrado) - Universidade Estadual de Maringá, Centro de Tecnologia, Departamento de Engenharia Química, Programa de Pós-Graduação em Engenharia Química, 2018.

1. Módulo experimental. 2. Controle de temperatura. 3. *Proporcional-integral-derivativo* (PID). 4. Modelagem. 5. Arduino (Plataforma). I. Gonçalves, Cid Marcos, orient. II. Souza, Marcos, coorient. III. Universidade Estadual de Maringá. Centro de Tecnologia. Departamento de Engenharia Química. Programa de Pós-Graduação em Engenharia Química. IV. Título.

CDD 21.ed.660.2815
622.43

Elaine Cristina Soares Lira – CRB 1202/9

UNIVERSIDADE ESTADUAL DE MARINGÁ
CENTRO DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA QUÍMICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA QUÍMICA

Esta é a versão final da Dissertação de Mestrado apresentada por Victor Hugo da Silva Barlatti perante a Comissão Julgadora do Curso de Mestrado em Engenharia Química em 24 de agosto de 2018.

COMISSÃO JULGADORA

Prof. Dr. Cid Marcos Gonçalves Andrade
Orientador / Presidente

Prof. Dr. Marcos de Souza
Coorientador

Prof. Dr. Carlos Alexandre Ferri
Membro

Prof. Dr. Abel Fidalgo Alves
Membro

DEDICATÓRIA

Dedico este trabalho aos meus pais Vitorio Tadeu Barlatti e Cicera Ribeiro da Silva Barlatti que sempre me deram apoio, incentivo e confiança, permitindo assim a sua realização.

AGRADECIMENTOS

Agradeço primeiramente a DEUS por ter me dado saúde e a honra de poder cursar esta pós-graduação bem como realizar e finalizar este trabalho.

Aos meus pais Cicera e Vitorio, e aos meus familiares, por todo o apoio e incentivo dados durante toda esta jornada.

Ao meu orientador Prof. Cid Marcos Gonçalves Andrade e meu co-orientador Prof. Marcos de Souza por toda a paciência, ajuda e tempo despendido durante a elaboração deste trabalho.

Ao meu colega Roberto que também participou do desenvolvimento desse projeto.

Ao meu colega Adriano que disponibilizou o seu tempo e o seu laboratório para a construção de algumas partes do equipamento.

A CAPES pela bolsa oferecida.

MÓDULO EXPERIMENTAL PARA MODELAGEM, SIMULAÇÃO, SINTONIA E ANÁLISE DE DESEMPENHO DE SISTEMAS DE CONTROLE

AUTOR: VICTOR HUGO DA SILVA BARLATTI

ORIENTADOR: CID MARCOS GOLÇALVES ANDRADE

CO-ORIENTADOR: MARCOS DE SOUZA

Dissertação de Mestrado; Programa de Pós-Graduação em Engenharia Química; Universidade Estadual de Maringá; Av. Colombo, 5790, BL E46 – 09; CEP: 87020-900 – Maringá – PR, Brasil, defendida em 24 de agosto de 2018.

RESUMO

Os sistemas de controle estão cada dia mais presentes nas indústrias e no cotidiano das pessoas. Em razão disso a importância de laboratórios práticos no ensino da disciplina de controle é primordial para a formação dos alunos. Entretanto plataformas de ensino de controle possuem um alto custo e muitas vezes são plataformas com softwares proprietários, o que dificulta ao aluno a implementação de novos recursos ao sistema. O objetivo deste trabalho foi desenvolver um módulo experimental de controle de temperatura baseado em uma plataforma livre onde os alunos possam posteriormente implementar novos recursos ao módulo. O módulo experimental foi desenvolvido com base em um tanque de aquecimento feito em acrílico e aquecido por meio de uma resistência elétrica. Foi utilizado a plataforma Arduino para a conexão entre os sensores e atuadores e a interface gráfica e o controlador PID desenvolvidos no software Scilab. Foi feita a modelagem do tanque e utilizado o método de sintonia de Ziegler-Nichols. Os resultados mostraram que o módulo desenvolvido funcionou corretamente e mostrou ser uma potencial ferramenta para o ensino da disciplina de controle.

Palavras-chave: *Módulo experimental; Controle de temperatura; PID; Modelagem; Ziegler-Nichols; Scilab; Arduino.*

EXPERIMENTAL MODULE FOR MODELING, SIMULATION, TUNING AND PERFORMANCE ANALYSIS OF CONTROL SYSTEMS

AUTHOR: VICTOR HUGO DA SILVA BARLATTI

SUPERVISOR: CID MARCOS GOLÇALVES ANDRADE

CO-SUPERVISOR: MARCOS DE SOUZA

Master Thesis; Chemical Engineering Graduate Program; State University of Maringá; Av Colombo, 5790, BL E-46 – 09; CEP: 87020-900 – Maringá – PR, Brazil, presented on 24th August 2018.

ABSTRACT

Control systems are increasingly present in industries and people's daily lives. Because of this, the importance of practical laboratories in teaching the discipline of control is essential for the training of students. However control education platforms are expensive and often are platforms with proprietary software, which makes it difficult for the student to implement new resources to the system. The objective of this work was to develop an experimental temperature control module based on a free platform where students can later implement new features to the module. The experimental module was developed based on a heating tank made in acrylic and heated by means of an electric resistance. The Arduino platform was used for the connection between the sensors and actuators and the graphic interface and the PID controller developed in Scilab software. The tank modeling was done and the Ziegler-Nichols tuning method was used. The results showed that the developed module worked correctly and proved to be a potential tool for teaching the control discipline.

Keywords*: Experimental module; Temperature control; PID; Modeling; Ziegler-Nichols; Scilab; Arduino.*

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS E ABREVIATURAS

ρ	densidade
w	vazão
V	volume
C	capacidade calorífica
Δ	variação
τ	constante de tempo
K	ganho
Q	quantidade de calor
W	watts
°C	graus Celsius
K_p	ganho proporcional
K_i	ganho integral
K_d	ganho derivativo
T	temperatura
Te	temperatura de entrada
L	atraso
LED	*Light Emitting Diode*
LDR	*Light Dependent Resistor*
PID	proporcional-integral-derivativo
PWM	*Pulse Width Modulation*
TRIAC	*Triode for Alternating Current*

SUMÁRIO

1. INTRODUÇÃO

1.1. Descrição do Problema

Os sistemas de controle estão cada vez mais presentes no cotidiano das pessoas, desde os sistemas mais simples (como, por exemplo, um simples brinquedo em algum parque de diversões) até, principalmente, os sistemas de controle utilizados na indústria (fornos, caldeiras, bombas, etc) (BARROS, 2013).

Devido a sua importância, a disciplina de controle é uma matéria abordada em muitos cursos de engenharia, onde são apresentados aos alunos os conceitos básicos sobre controle e sistemas (SCHMID, 2000).

Por abordar frequentemente conceitos abstratos e complexos que não são demonstrados facilmente em sala de aula, e também utilizar de uma matemática frequentemente considerada avançada, os alunos acabam não conseguindo assimilar o conteúdo desenvolvido em sala de aula com a prática, em razão disso, disciplinas práticas possuem um papel importantíssimo no ensino da engenharia (SCHMID, 2000).

Entretanto laboratórios de ensino de controle geralmente necessitam de uma ampla quantidade de recursos que muitas vezes não está disponível para o aluno (SCHMID, 2000).

Plataformas de ensino para disciplinas experimentais como o LabView da National Instruments por exemplo oferecem todos os recursos para a realização de práticas de ensino de controle, no entanto essas plataformas possuem um custo de aquisição elevado e também necessitam muitas vezes de taxas anuais de assinatura. Essas plataformas também são de software proprietário e de hardware fechado, o que dificulta o acesso do aluno ao que está acontecendo exatamente no sistema além de não ser possível a implementação pelos próprios alunos de novos recursos e técnicas de controle.

Considerando esses aspectos o desenvolvimento de módulos experimentais para a prática e o ensino de controle utilizando software livre como por exemplo o Scilab e plataformas abertas como o Arduino, aliados a dispositivos e componentes de baixo custo, os módulos experimentais se tornam uma excelente ferramenta para o ensino prático da disciplina de controle.

Na literatura encontramos alguns trabalhos de criação de módulos experimentais relacionados aos sistemas de controle. A Tabela 1.1 apresenta uma lista resumida de alguns desses trabalhos.

Tabela 1.1 – Módulos experimentais na literatura.

Tipo de controle	Sistema	Atuador	Referência
Temperatura	Trocador de calor	Válvula eletropneumática proporcional	(MARTIN; KASSAB JUNIOR, 2006)
Nível	Tanques acoplados	Bomba de água	(RAMOS; WENSE, 2008)
Nível	Tanques acoplados	Válvula proporcional elétrica	(GNOATTO et al., 2016)
Nível	Tanques acoplados	Bomba de água	(CARVALHO et al., 2010)
Nível	Tanques acoplados	Válvula proporcional elétrica	(BERTACHI et al., 2013)
Temperatura	Ambiente aquecido	Lâmpada incandescente	(ARAÚJO et al., 2004)

Fonte: Autor.

1.2. Objetivo

O objetivo deste trabalho foi o desenvolvimento de um módulo experimental de controle de temperatura de um tanque utilizando o software livre Scilab e a plataforma Arduino.

O controlador utilizado foi do tipo PID desenvolvido no software Scilab. Uma interface gráfica de fácil utilizbação também será desenvolvida.

1.3. Estrutura do trabalho

No capítulo 1 fez-se a introdução, descrição do problema e a apresentação dos objetivos do trabalho.

No capítulo 2 aborda-se uma revisão bibliográfica sobre sistemas de controle, a modelagem do tanque de aquecimento, controlador PID, o método de sintonia e as plataformas utilizadas.

No capítulo 3 foi apresentado todo o desenvolvimento e construção do módulo experimental.

No capítulo 4 apresenta-se os resultados obtidos.

No capítulo 5 apresenta-se a conclusão do trabalho.

Nos Apêndices A e B apresenta-se os códigos desenvolvidos no Scilab e no Arduino.

2. REVISÃO BIBLIOGRÁFICA

Neste capítulo apresenta-se uma revisão sobre sistemas de controle, controle por realimentação e modelagem. Descreve-se também o controlador PID e o seu principal método de sintonia, o método de Ziegler-Nichols. Apresenta também a plataforma Arduino e o *software* Scilab.

2.1. Sistemas de controle

Um sistema de controle constitui-se da relação entre elementos e processos organizados de modo a se obter uma resposta de saída desejada (NISE, 2012).

Os sistemas de controle possuem basicamente os componentes fundamentais: processo, elemento de medida, controlador e elemento final de controle. Esses componentes envolvem basicamente os principais elementos de um sistema de controle junto com os termos: referência, variável controlada, variável manipulada e perturbação (COUGHANOWR, 2009).

Variável controlada é o sinal que representa a grandeza física que está sendo controlada. A referência ou set-point é o valor pré-estabelecido que o controlador tentará manter a variável controlada. A variável manipulada é a variável cujo controlador deve intervir no processo e obter o valor desejado da variável controlada. Perturbação ou distúrbio é uma variação em qualquer variável capaz de alterar a variável controlada (COUGHANOWR, 2009).

2.2. Controle por realimentação

Um sistema de controle por realimentação ou também chamado de controle de malha fechada exibe um diagrama de blocos como pode ser visto na Figura 2.1, onde o controlador é a peça chave responsável por receber e gerar um sinal de controle para que a saída do processo esteja sempre o mais próximo do valor de referência.

Nesse tipo de controle o valor do sinal da variável controlada é comparada continuamente pelo controlador com o valor da referência (*set-point*), a diferença entre os dois valores é o erro. Conforme a magnitude do erro medido, o controlador providencia uma ação

de controle por meio do atuador através de um sinal de controle, esse por sua vez atua no processo a fim de ajustar a variável controlada ao valor da referência (COUGHANOWR, 2009).

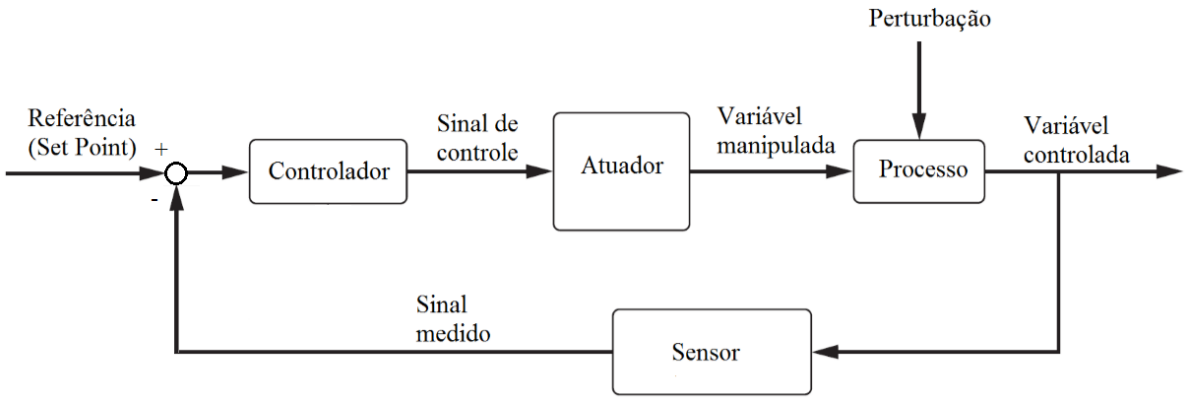

Figura 2.1 – Sistema de controle por realimentação.

Fonte: Autor.

2.3. Modelagem

A obtenção de uma equação matemática que descreve um processo real é denominada de modelagem. A modelagem permite compreender o processo de forma mais simples ao mesmo tempo em que descreve o processo de uma forma precisa (SODRÉ, 2007).

Para analisar o comportamento da temperatura da água no interior do tanque e também obter os parâmetros necessários para sintonizar o controlador é necessário o conhecimento da função de transferência do processo a qual é obtida por meio da modelagem do sistema.

As duas maneiras fundamentais para a obtenção da função de transferência são:

- modelo fenomenológico;
- modelo experimental.

O modelo fenomenológico fundamenta-se nas leis da conservação da massa e da energia por meio das equações de balanço energético que caracterizam o processo. Já o modelo experimental é fundamentado em dados experimentais obtidos do processo por meio de ensaios em malha aberta. A seguir são descritos os dois modelos para o sistema de aquecimento.

Modelo fenomenológico do sistema de aquecimento

A Figura 2.2 ilustra o processo de aquecimento do tanque a ser compreendido.

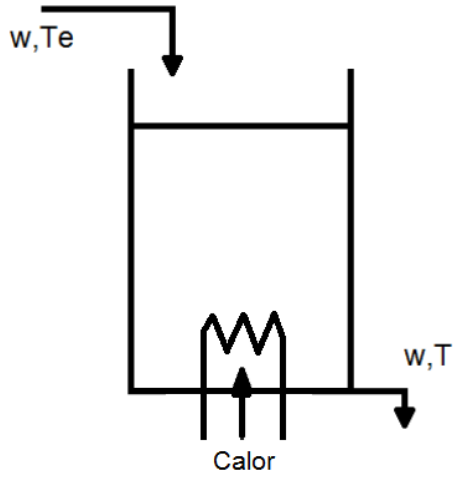

Figura 2.2 – Tanque de aquecimento.

Fonte: Autor.

Por meio de um balanço energético no tanque de aquecimento é obtido o modelo fenomenológico do sistema. O balanço energético pode ser descrito pela Equação 2.1 de uma forma simplificada em que relaciona-se as energias presentes no sistema. As propriedades físicas envolvidas nesse processo são a temperatura de saída do tanque (T), dada em °C, a quantidade de calor (Q) que é adicionada ao sistema pela resistência de aquecimento, dada em W, a vazão de saída do tanque (w), dada em gramas/s, o volume (V) do tanque, dado em gramas, a densidade (ρ) e a capacidade calorífica (C) do fluído, respectivamente em gramas/cm³ e cal/g.°C (COUGHANOWR, 2009).

$$\begin{pmatrix} Taxa\ de \\ fluxo \\ de\ energia \\ que\ entra \\ no\ tanque \end{pmatrix} - \begin{pmatrix} Taxa\ de \\ fluxo \\ de\ energia \\ que\ sai \\ do\ tanque \end{pmatrix} + \begin{pmatrix} Taxa\ de\ calor \\ injetado\ no \\ tanque \end{pmatrix} = \begin{pmatrix} Taxa\ de \\ energia \\ acumulada\ no \\ tanque \end{pmatrix} \quad (2.1)$$

Realizando o balanço energético no sistema junto com as seguintes considerações: regime permanente e temperatura de entrada do fluído no tanque constante, é possível obter por meio da aplicação da Transformada de Laplace (COUGHANOWR, 2009), a função de transferência do sistema dada pela Equação 2.2:

$$\frac{T(s)}{Q(s)} = \frac{1/wC}{\left(\frac{\rho V}{w}\right)s + 1} \tag{2.2}$$

Onde:

$1/wC$ é o ganho K do sistema dado em °C/W;

$\frac{\rho V}{w}$ é a constante de tempo τ do sistema dada em s.

Reescrevendo a Equação 2.2 em termos de K e τ, obtém-se a Equação 2.3.

$$\frac{T(s)}{Q(s)} = \frac{K}{\tau s + 1} \tag{2.3}$$

A função de transferência do tanque resulta em um sistema de primeira ordem como pode ser observado na Equação 2.3 e a sua constante de tempo τ depende principalmente do volume e da vazão de saída do tanque.

Modelo experimental do sistema de aquecimento

Por meio de um ensaio em malha aberta é obtida a curva de resposta da saída do sistema, para isso é aplicado um degrau de potência na resistência elétrica e observa-se o aumento da temperatura de saída até entrar em regime permanente. Na Figura 2.3 é apresentada uma curva de resposta ao degrau do sistema de aquecimento em que pode-se observar o tempo morto do sistema.

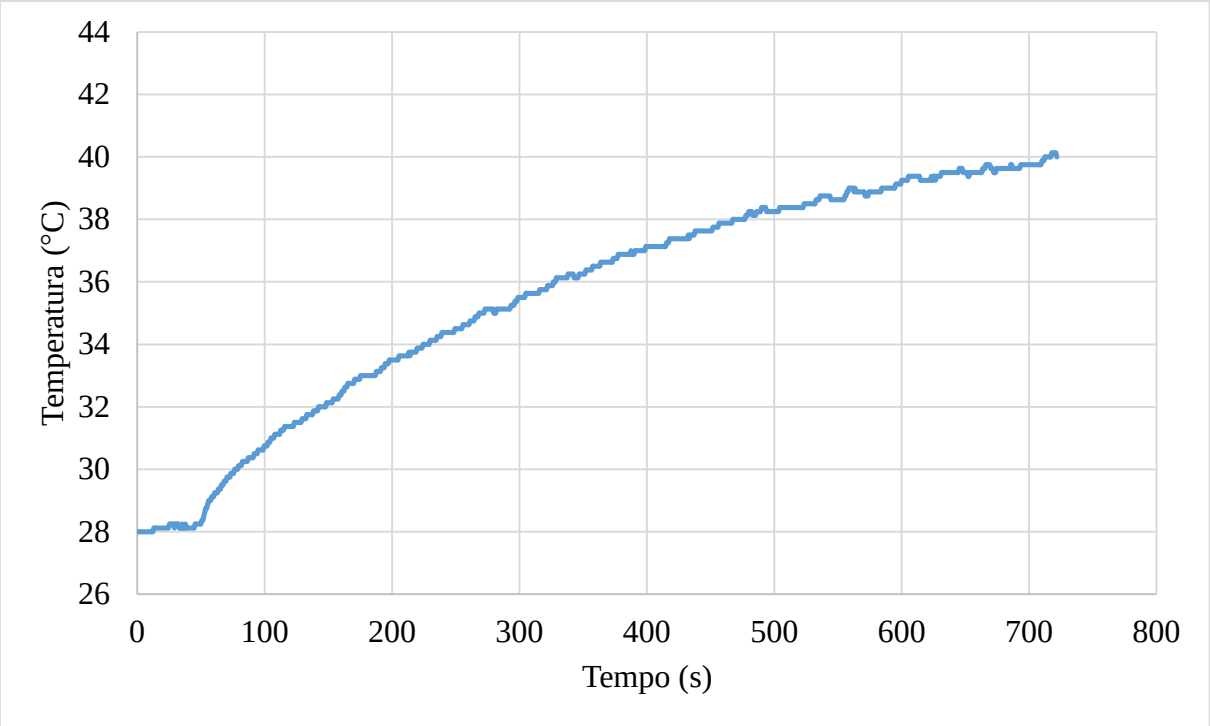

Figura 2.3 – Curva de resposta ao degrau.

Fonte: Autor.

O sistema apresenta um tempo morto significativo, portanto são utilizados os métodos de identificação para sistemas FOPDT (*First-Order Plus Dead-Time*) descritos em (COELHO, 2004), que consiste em uma função de transferência de primeira ordem acrescida de um atraso, como mostra a Equação 2.4.

$$G(s) = \frac{K}{\tau s + 1} e^{-sL} \tag{2.4}$$

Para a obtenção do ganho do sistema, K, divide-se a variação da temperatura pela variação do degrau aplicado, conforme a Equação 2.5.

$$K = \frac{\Delta T}{\Delta D} \tag{2.5}$$

A constante de tempo τ é o tempo que a temperatura de saída leva para atingir 63,2% da temperatura final atingida. O atraso L é o tempo que a saída do sistema leva para responder após o início do degrau.

2.4. PID

Para tornar o erro cada vez mais próximo de zero se faz necessário a utilização de um algoritmo de controle, o algoritmo mais utilizado em sistemas de controle industrial em todo o mundo, devido ao seu desempenho, funcionalidade e facilidade de sintonia, é o controlador PID (Proporcional-Integral-Derivativo), que consiste em três coeficientes que são variados de forma a obter a melhor resposta (SANTOS, 2014).

O controlador PID possui a função de transferência dada pela Equação 2.6 (SANTOS, 2014; NISE, 2012).

$$G_C(s) = K_p + \frac{K_i}{s} + K_d s \tag{2.6}$$

Onde K_p é o ganho proporcional, K_i é o ganho integral e K_d é o ganho derivativo.

2.5. Ziegler Nichols

Em 1942 foi proposto por Ziegler e Nichols um método matematicamente simples para a sintonia do controlador PID, que é utilizados até hoje (SANTOS, 2014). O método se baseia nos parâmetros τ, L e K da Equação 2.7. As constantes do controlador PID, descritas em (O'DWYER, 2009) podem ser obtidas pela Tabela 2.1.

$$G(s) = \frac{K}{\tau s + 1} e^{-sL} \tag{2.7}$$

Tabela 2.1 – Constantes do controlador PID pelo método de Ziegler e Nichols.

Controlador	K_p	K_i	K_d
PI	$\frac{0.9\tau}{KL}$	$\frac{K_p}{3.33L}$	-
PID	$\frac{1.2\tau}{KL}$	$\frac{K_p}{2L}$	$K_p 0.5L$

Fonte: Autor.

2.6. PWM

Modulação por largura de pulso ou PWM (*Pulse Width Modulation*) é um método utilizado principalmente para o controle de potência de equipamentos elétricos. Por meio de uma onda quadrada com frequência fixa, o PWM fundamenta-se na variação da largura dos pulsos dessa onda quadrada obtendo assim uma variação da tensão média aplicada. Quanto maior for a largura desses pulsos maior é a potência fornecida a carga (EMBARCADOS, 2018).

O tempo de duração do pulso em relação ao período completo da onda quadrada é chamado de ciclo ativo ou *duty cicle*, sendo fornecido em percentagem (EMBARCADOS, 2018).

A Figura 2.4 exibe o sinal PWM para alguns *duty cicles*.

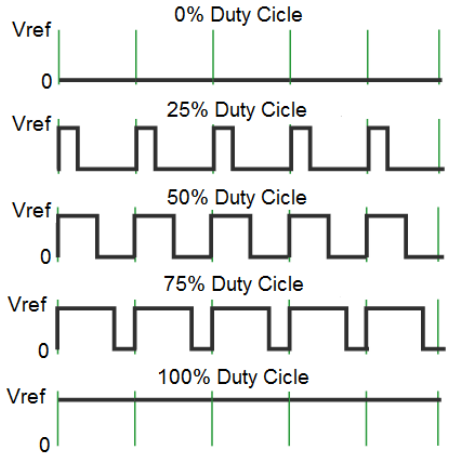

Figura 2.4 – Sinal PWM com vários *duty cicles*.

Fonte: Autor.

Com *duty cicles* diferentes pode-se ter diferentes tensões médias aplicada a carga, na faixa de 0 a 100% do valor da tensão de referência Vref, com isso é possível ter o controle da potência fornecida a carga.

2.7. Arduino

O Arduino é uma plataforma eletrônica de código aberto baseada em hardware e software fáceis de usar. O projeto iniciou-se em 2005 na Itália com a intenção de ser utilizado em atividades educacionais e de interação com aplicações escolares (ARDUINO, 2018; GONZAGA, 2015).

A placa Arduino, Figura 2.5, está apta a ler sinais de entradas (por exemplo luz em um sensor, um botão, etc) e transforma-los em sinais de saída (ativando um motor, acendendo um LED, etc). O usuário pode informar a placa Arduino o que fazer, enviando um conjunto de instruções através da linguagem de programação Arduino (baseada em *Wiring*) e do ambiente de desenvolvimento IDE (baseado em *Processing*) (ARDUINO, 2018).

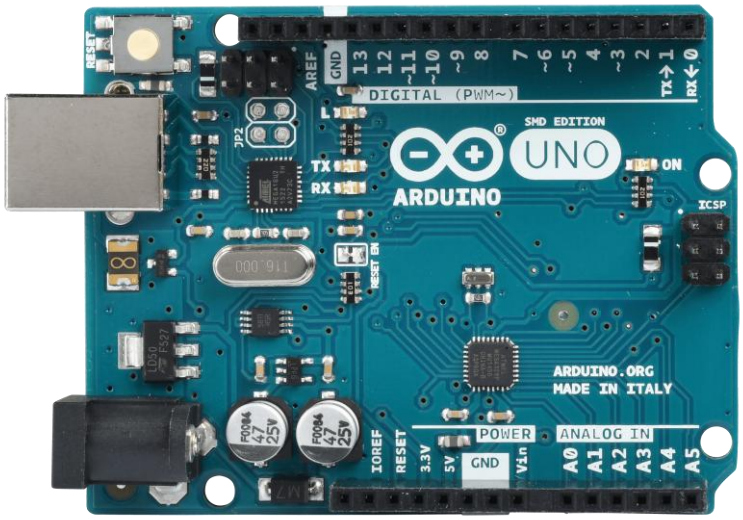

Figura 2.5 – A placa Arduino Uno.

Fonte: Arduino, 2018.

O Arduino Uno, modelo utilizado neste trabalho, é baseado no microcontrolador ATmega328P, ele possui 14 pinos digitais de entrada/saída (dos quais 6 podem ser utilizados como PWM), 6 entradas analógicas e uma conexão USB (ARDUINO, 2018).

2.8. Scilab

O Scilab é um software livre e de código aberto para computação numérica, ele é uma poderosa ferramenta para aplicações científicas e de engenharia (SCILAB, 2018).

O Scilab possui uma linguagem de programação de alto nível e contém centenas de funções matemáticas. Ele proporciona acesso a estruturas avançadas de dados e funções gráficas em duas e três dimensões. Entre suas principais aplicações, destaca-se: controle, simulação, otimização, processamento de sinais (SCILAB, 2018).

O Scilab possui *toolboxes* que são ferramentas adicionais que aumentam a sua funcionalidade. Uma delas é a GUI Builder 3.0 que é uma ferramenta construtora de interface gráfica, ela permite que você construa facilmente uma interface gráfica e gera o código no Scilab automaticamente.

3. METODOLOGIA

O processo de desenvolvimento dos tanques, os componentes utilizados no módulo experimental e a interface gráfica desenvolvida são mostrados nesse capítulo.

3.1. Tanques

Foram desenvolvidos dois tanques em acrílico *cast* transparente com formato cilíndrico com altura de 1 m e diâmetro de aproximadamente 32 cm cada tanque. Foram utilizadas chapas de acrílico de 2 mm e 3 mm de espessura.

Pensando em trabalhos futuros de controle de tanques com interação motivou o desenvolvimento de dois tanques, mas neste trabalho será usado apenas um tanque para o controle de temperatura.

O acrílico *cast* possui uma boa durabilidade e uma excelente transparência que aliadas as suas propriedades térmicas descritas na Tabela 3.1 contempla o necessário para o projeto, como o suporte de uma temperatura alta devido ao aquecimento do tanque, a resistência do acrílico devido a pressão da água que irá abastecer o tanque. Além disso, o acrílico *cast* possui um bom custo x benefício, o que foi primordial para o projeto.

Tabela 3.1 – Propriedades térmicas do acrílico *cast*.

Temperatura de moldagem das chapas	165 a 190 ºC
Temperatura de uso contínuo das chapas	-40 a 80 ºC
Condutividade térmica	0,18 W/m ºC
Auto ignição	Acima de 490 ºC

Fonte: Autor.

A fabricação desse tipo de acrílico se dá no formato de chapas onde a matéria prima básica, o metil metacrilato líquido, juntamente com aditivos, pigmentadores e catalisadores, são introduzidos nos moldes feitos de lâminas de vidro temperado e em seguida aquecidos em autoclave, onde inicia a polimerização do acrílico, passando de líquido para sólido (CENTRAL DO ACRÍLICO, 2018).

Para determinar a qualidade das placas elas passam por um controle de qualidade para que atendam à norma ISO 7823-1 (CENTRAL DO ACRÍLICO, 2018).

Foi escolhido o formato cilíndrico para os tanques por possuir o menor número de pontos de cola e possibilitar uma melhor circulação do fluído em seu interior. As chapas de acrílico foram aquecidas com um soprador de ar quente até se tornarem maleáveis e então foi se dobrando com a ajuda de um molde cilíndrico de latão até atingirem o formato de um cilindro. Ao fundo foi recortada uma base circular de acrílico com as devidas dimensões.

A colagem das bordas e da base foi feita utilizando uma cola especifica para acrílico que garante até 80% da resistência do acrílico.

A colagem seguiu as recomendações do fabricante, o qual recomenda preencher um vão de no mínimo 0,8 mm entre as placas com cola, para isso é utilizado uma fita posicionada atrás do espaço formado entre as placas até que a cola seque por completo e atinja sua cura total.

Uma cinta metálica foi colocada ao redor do tanque a fim de dar maior sustentação e segurança a emenda. A Figura 3.1 exibe os tanques finalizados.

Figura 3.1 – Tanques de acrílico desenvolvidos.

Fonte: Autor.

3.2. Bomba de água

Para o correto funcionamento do sistema de controle de temperatura do tanque, o nível e a vazão do tanque devem permanecer constantes, portanto foi feito o uso de uma bomba de água na saída do tanque que atendesse as necessidades do projeto.

A bomba de água utilizada que atende aos requisitos do projeto apresenta as seguintes características:

- potência: 60 W;
- tensão: 12 V;
- fluxo máximo: 5 Litros/min.

Para um melhor controle da vazão e a obtenção de uma resposta mais rápida do sistema de aquecimento (uma vazão menor implica em uma constante de tempo menor) se faz necessário um controle da vazão da bomba utilizada, para isso foi desenvolvido um circuito de controle de potência PWM para a bomba de água.

Para esse circuito foi utilizado um transistor de potência do tipo MOSFET modelo IRF540N operando em conjunto com um transistor de polarização BC547. Esse circuito recebe o sinal PWM do Arduino e fornece um sinal de potência para a bomba de água. Figura 3.2 apresenta o diagrama esquemática do circuito.

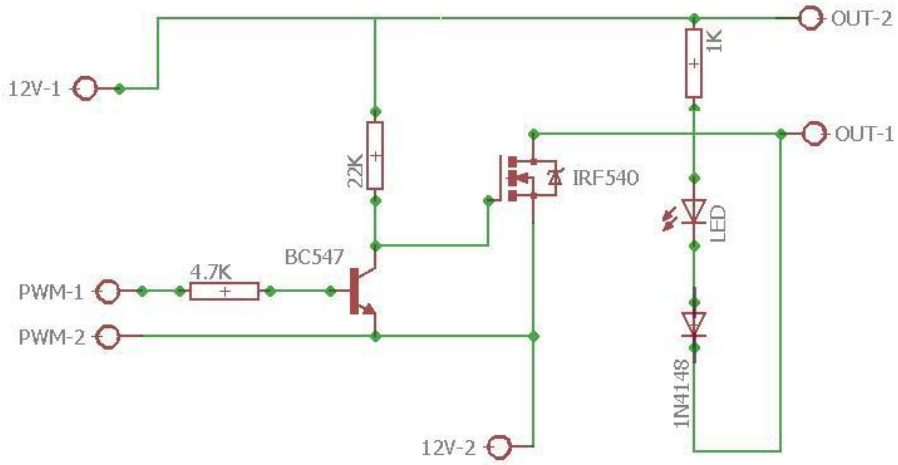

Figura 3.2 – Circuito de controle PWM de potência da bomba de água.

Fonte: Autor.

A Figura 3.3 demonstra a bomba de água utilizada.

Figura 3.3 – Bomba de água e o circuito de controle de potência finalizados.

Fonte: Autor.

3.3. Sensor de temperatura

O DS18B20 é um sensor digital de temperatura que opera com uma resolução de 9 a 12 bits. Ele possui uma versão instalada dentro de um encapsulamento a prova de água.

A Figura 3.4 mostra o sensor DS18B20 dentro e fora do encapsulamento a prova de água.

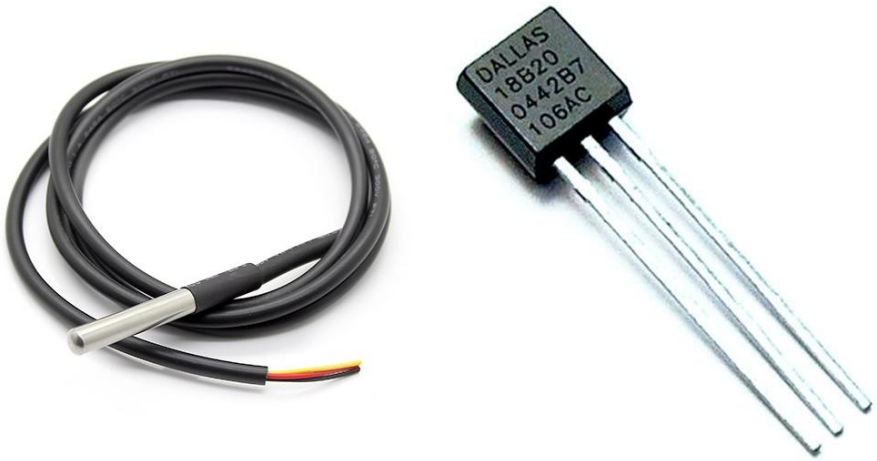

Figura 3.4 – Sensor digital de temperatura DS18B20.

Fonte: Autor.

O DS18B20 se comunica através de um barramento *one-wire* que, por definição, requer apenas uma linha de dados (e terra) para comunicação com um microncontrolador. Cada DS18B20 possui um código serial exclusivo de 64 bits, permitindo que múltiplos DS18B20 funcionem no mesmo barramento. Assim, é simples usar um microcontrolador para controlar muitos DS18B20 distribuídos por uma grande área. O DS18B20 pode ser utilizado em muitas áreas, como: controles ambiental, monitoramento de temperatura de sistemas dentro de edifícios, equipamentos ou maquinaria, monitoramento de processos e sistemas de controle (MAXIM INTEGRATED, 2015).

O DS18B20 possui três terminais sendo: um termina de alimentação que é conectado a saída 5 V do Arduino, um terminal de GND (Terra) que é conectado ao terminal GND do Arduino e o terminal de dados que é conectado em uma das portas digitais do Arduino.

Neste trabalho o DS18B20 operou com uma resolução de 9 bits o que garantiu uma precisão de 0,5 °C ainda segundo Maxim Integrated (2015) os sensores já são calibrados de fábrica e não necessitam de qualquer ajuste por parte do usuário, permitindo sua imediata aplicação como termômetro.

3.4. Atuador

O atuador é o elemento do sistema de controle de temperatura que converte o sinal da saída do controlador em um sinal elétrico utilizado para acionar os dispositivos que causem o efeito desejado no sistema (CERCHIARO, 2006).

Neste trabalho o atuador converte o sinal digital da saída do controlador para um sinal elétrico que acionará uma resistência elétrica no interior do tanque.

O dispositivo fundamental do atuador é o TRIAC. O TRIAC é um dispositivo eletrônico que conectado em série com a resistência elétrica permite controlar quando a resistência elétrica é ligada, funcionando assim como uma chave.

Visto que o TRIAC permite controlar o acionamento da resistência elétrica, então ele permite controlar a quantidade de calor introduzida no sistema.

Neste trabalho foi utilizado um TRIAC modelo BTA41, esse modelo suporta uma corrente de 40 A e uma tensão de 600 V, ele está conectado a uma rede bifásica 220 V. Para o devido arrefecimento do componente, foi utilizado um cooler de computador.

Para o acionamento do TRIAC é necessário a utilização de um optoacoplador, ele tem a função de compatibilizar o sinal vindo do controlador por meio do Arduino com o TRIAC, ele também isola o Arduino eletricamente do circuito de potência, funcionando também como uma proteção para o Arduino. O modelo utilizado neste trabalho foi o MOC3020.

O elemento final do atuador, que é o que aquece a água no tanque, é uma resistência elétrica de chuveiro, 220 V, com potência de 6800 W. A Figura 3.5 ilustra o circuito.

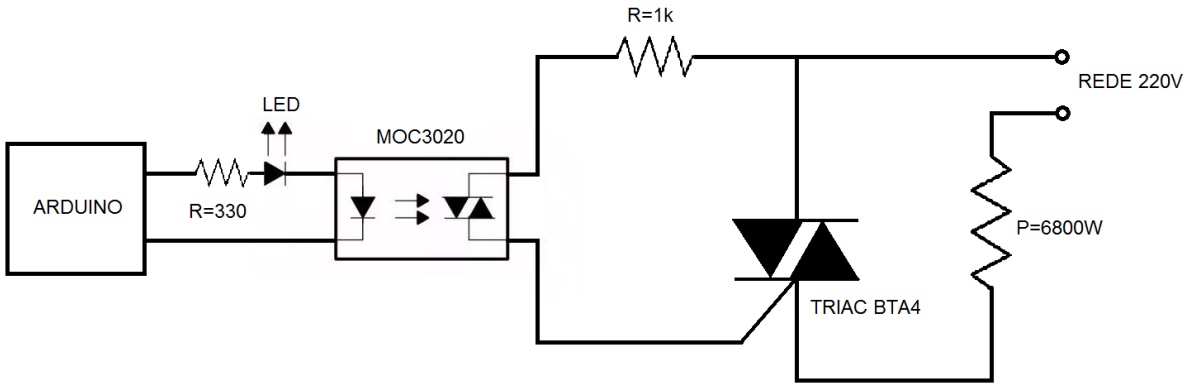

Figura 3.5 – Circuito do atuador.

Fonte: Autor.

O controle da potência se dará da seguindo forma: o controlador enviará um sinal para o Arduino que por sua vez irá enviar um sinal PWM, com o duty cicle estabelecido pelo controlador, para o atuador. Esse sinal PWM recebido pelo atuador irá fazer o chaveamento do TRIAC permitindo a passagem do sinal de tensão da rede 220 V para a resistência elétrica durante o tempo de ciclo ativo do sinal PWM. A figura 3.6 ilustra o processo.

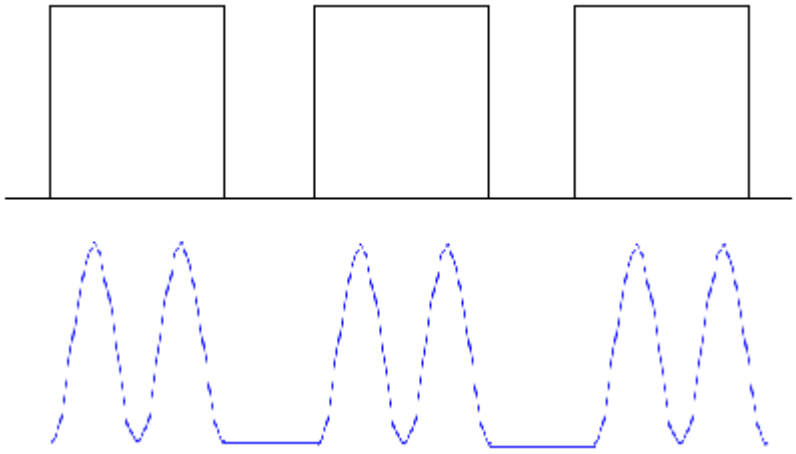

Figura 3.6 – Controle PWM do sinal de tensão senoidal da rede.

Fonte: Autor.

A frequência do PWM do Arduino foi alterada para um valor inferior a 60 Hz devido ao funcionamento do TRIAC, que se desliga quando a tensão passa pelo zero da senóide e uma frequência muito alta acaba as vezes não tendo tempo para o desligamento do TRIAC.

3.5. Controlador

O controlador é a parte fundamental do sistema de aquecimento que irá determinar a temperatura desejada do sistema por meio de uma técnica de controle estabelecida. Os sensores de temperatura enviam a informação para o controlador e ele faz os cálculos necessários, conforme a técnica de controle estabelecida, para enviar o sinal para o atuador que atuará no sistema com a energia necessária para se estabelecer a temperatura definida.

Implementado por meio do software Scilab, o controlador desenvolvido foi do tipo PID, foi utilizada uma versão discretizada da Equação 2.6 devido ao controlador ter sido implementado em um ambiente digital. A seguir está o código implementado no Scilab responsável pelo controlador PID:

```
//Função responsavel pelo controle PID
function
   exec(kp);
   exec(ki);
   exec(kd);
   exec(vazao);
   global %Output;

    timeChange=timer();    //mede o tempo entre cada medição da temperatura
    timeChange = evstr(timeChange);
    //Equação do controlador PID digital simples
    erro = %Setpoint - data2;
    errSum = (erro * timeChange);
    errSum = errSum + (erro * timeChange);
    dErr = (erro - lastErr) / timeChange;
    Output = Kp1 * erro + Ki1 * errSum + Kd1 * dErr;
   //conversão dos sinais
   Output=round(Output);
   Output=string(Output);
   ee=string(data1);
   s=string(data2);
   t=string(timeChange);
   lastErr = erro;
   data2=0;
   //restringindo saída do controlador
   %Output=evstr(Output);
      if(%Output > 255) then
         Output = '255';
      end
      if(%Output < 0) then
         Output = '0';
      end
```

Como o sistema não é contínuo são realizadas medições de temperatura a cada intervalo de tempo, esse intervalo de tempo não é fixo e possui um valor vriável. O tempo de amostragem médio nesse sistema está sendo de aproximadamente 0,73 s. A variável *timeChange* armazena esse intervalo de amostragem.

A variável *erro* calcula o erro do sistema e então é utilizada para o cálculo de cada parcela de erro do controlador PID:

- *erro*: já é o próprio erro referente a parcela proporcional.
- *errSum*: é o erro referente a parcela integral.
- *dErr*: é o erro referente a parcela derivativa.

A saída do controlador *Output* nada mais é que as constantes proporcional, integral e derivativa multiplicadas pelos seus respectivos erros. Após o cálculo do *Output* é feito uma restrição para um valor máximo de 255 em razão da saída PWM do Arduino operar na faixa de 0 a 255.

O valor do *Output* é então enviado para o Arduino o qual gera um sinal PWM com um *duty cicle* proporcional a esse valor, esse sinal é enviado ao atuador que faz o disparo do TRIAC liberando uma potência proporcional ao valor do *Output*.

A seguir é apresentado um pseudocódigo do algoritmo de controle para um melhor entendimento:

1.Realiza a medida da temperatura de saída do sistema.

2.Lê as entradas inseridas no programa das constantes Kp, Ki e Kd.

3.Mede e armazena o tempo entre cada medida de temperatura.

4.Compara a temperatura de saída com o *set point*.

5.Calcula os erros proporcional, integral e derivativo.

6.Calcula o sinal de saída do controlador utilizando a equação de controle PID.

7.Restringe o sinal de saída do controlador a faixa de 0 a 255.

8.Repete o passo1.

3.6. Interface gráfica de comando

Foi elaborada uma interface gráfica a partir do *toolbox* do Scilab GUI Builder 3.0, onde o operador pode visualizar e dar os comandos para o sistema.

São mostrados na tela a temperatura de entrada e saída do tanque, a vazão e um gráfico em tempo real que exibe as temperaturas, o sinal de ação do controlador e o set-point. O operador também pode selecionar o tipo de controle por meio de botões na interface.

O valor do degrau pode ser estabelecido pelo operador na caixa degrau assim como o valor do set-point na sua respectiva caixa. As constantes Kp, Ki e Kd do controlador PID devem ser indicadas pelo operador quando selecionado o tipo de controle PID Feedback.

O programa também cria um arquivo *txt* com um nome especificado pelo usuário toda vez que é iniciado, salvando todas as informações fornecidas pelo sistema para que possam ser utilizadas depois para a criação de gráficos e análise de resultados de cada experimento realizado.

Ao pressionar o botão iniciar o usuário inicia um novo experimento e gera um novo arquivo *txt*. Ao fim de cada experimento o usuário deve clicar no botão parar para o programa interromper a gravação de dados no arquivo *txt*.

Na Figura 3.7 pode ser visto o arquivo *txt* gerado e na Figura 3.8 pode ser visto a interface gráfica desenvolvida.

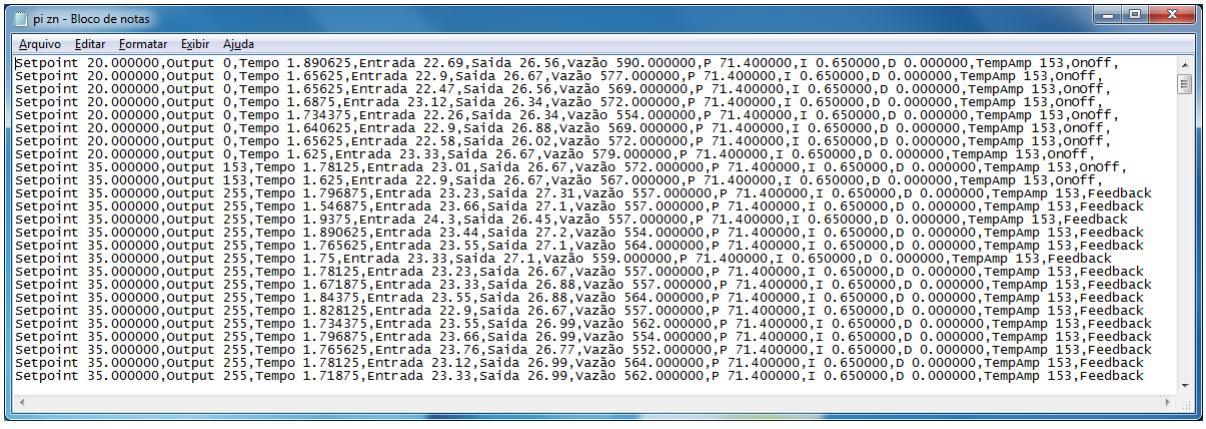

Figura 3.7 – Arquivo *txt* gerado pelo programa.

Fonte: Autor.

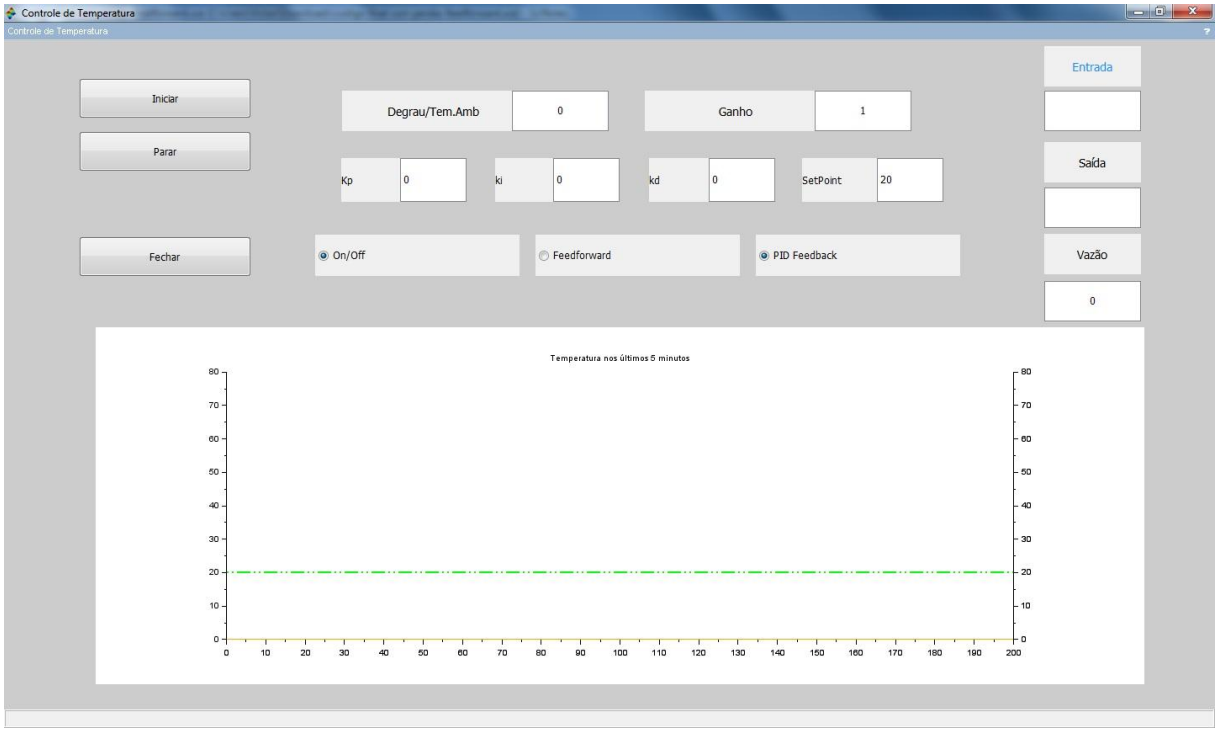

Figura 3.8 – Interface gráfica desenvolvida.

Fonte: Autor.

3.7. O código desenvolvido no Scilab

O fluxograma demonstrado na Figura 3.9 ilustra de forma simplificada o princípio de funcionamento de todo o código desenvolvido no Scilab desde a interface gráfica até o controlador.

No Apêndice A encontra-se o código completo do programa Controle de Temperatura desenvolvido no Scilab.

No Apêndice B encontra-se o código utilizado no Arduino para o seu correto funcionamento com a comunicação serial com o Scilab e a correta leitura de todos os sensores.

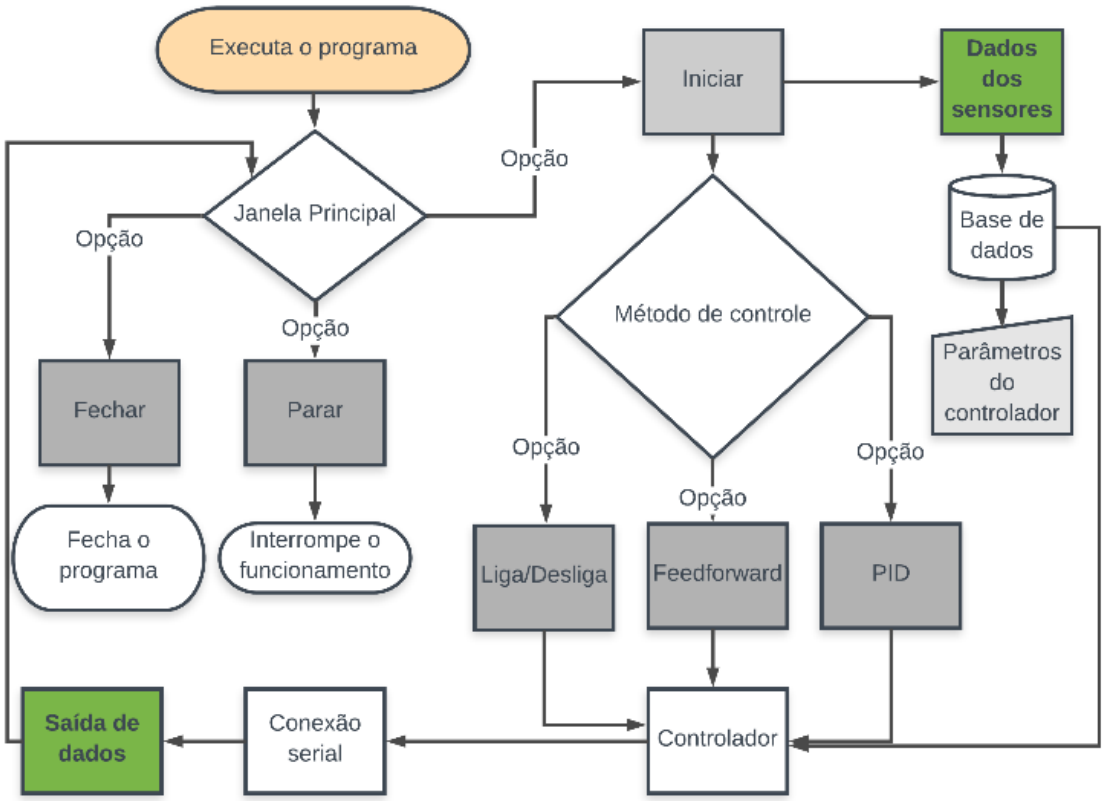

Figura 3.9 – Interface gráfica desenvolvida.

Fonte: Autor.

3.8. O módulo experimental finalizado

Com todos os componentes do sistema devidamente montados, o módulo experimental está finalizado. O Arduino Uno faz toda a interface entre os sensores e atuadores do sistema e o controlador e a interface gráfica desenvolvidos no Scilab. O Arduino se comunica com um computador onde se encontra o programa Controle de Temperatura por meio de uma porta USB. Uma fonte ATX fornece a tensão 12 V necessária para o funcionamento dos dispositivos.

A Figura 3.10 representa o diagrama de funcionamento do módulo experimental, a Figura 3.11 mostra o módulo experimental finalizado e pronto para uso. O módulo se encontra no bloco D90 da UEM.

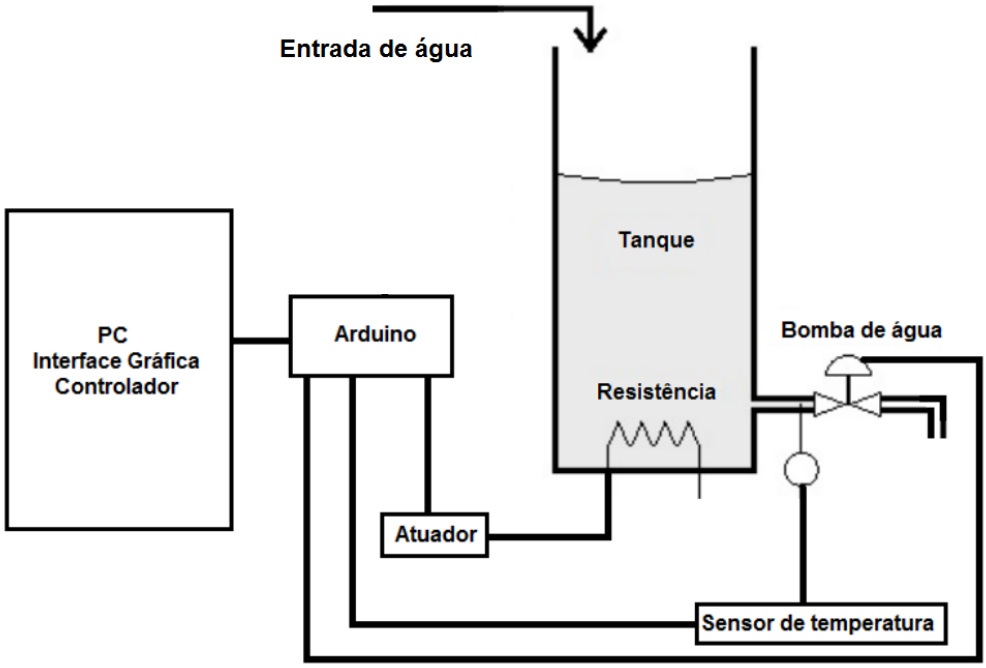

Figura 3.10 – Diagrama do módulo experimental.

Fonte: Autor.

Figura 3.11 – Módulo experimental finalizado.

Fonte: Autor.

Destaca-se também o sistema de proteção que foi adicionado ao sistema, além do disjuntor do quadro de energia ao lado, que protege a rede que alimenta o módulo, usou-se também um disjuntor bifásico para a proteção do sistema atuador.

A Tabela 3.2 apresenta o custo de cada componente do módulo experimental e o valor total gasto no desenvolvimento do módulo experimental.

Tabela 3.2 – Custo do módulo experimental

Componentes	Valor
Placa de acrílico	R$100,00
Cola S330	R$65,80
Sensores de temperatura	R$61,70
Resistência elétrica	R$20,00
Bomba de água	R$120,00
Circuito atuador	R$60,00
Circuito controle bomba	R$30,00
Mangueiras e chuveiro	R$55,00
Caixas MDF	R$10,00
Disjuntores	R$57,35
Cabos e encanamentos	R$39,00
Arduino UNO	R$40,00
Total	R$658,85

Fonte: Autor.

4. RESULTADOS E DISCUSSÃO

Este capítulo apresenta os resultados dos testes realizados com o módulo experimental completamente finalizado.

4.1. Teste do módulo experimental

Nesse trabalho o tanque foi preenchido com 16,5 litros de água, com uma vazão de entrada e saída mantidos em 49 gramas/s. A resistência elétrica utilizada no interior do tanque para o aquecimento da água foi de 6800W. Com essas características físicas do sistema em mãos foi possível calcular a função de transferência teórica do sistema por meio do modelo fenomenológico. A função de transferência teórica, que é calculada por meio das Equação 2.2, é dada pela Equação 4.1.

$$G(s) = \frac{0.0049}{337s + 1} \tag{4.1}$$

A cada teste realizado o sistema vai apresentar uma função de transferência com valores diferentes mas próximos aos da Equação 4.1, isso ocorre devido a vazão variar e também a temperatura ambiente que muda conforme o dia.

Com o módulo experimental finalizado, foram feitos alguns testes a fim de verificar o devido funcionamento do sistema.

Foi realizado um ensaio em malha aberta no sistema e obtida a função de transferência experimental do sistema. A Figura 4.1 mostra a curva de resposta ao degrau do sistema.

Com os dados adquiridos é obtida a função de transferência experimental do sistema dada pela Equação 4.2.

$$G(s) = \frac{0{,}0054}{287s + 1} e^{-20s} \tag{4.2}$$

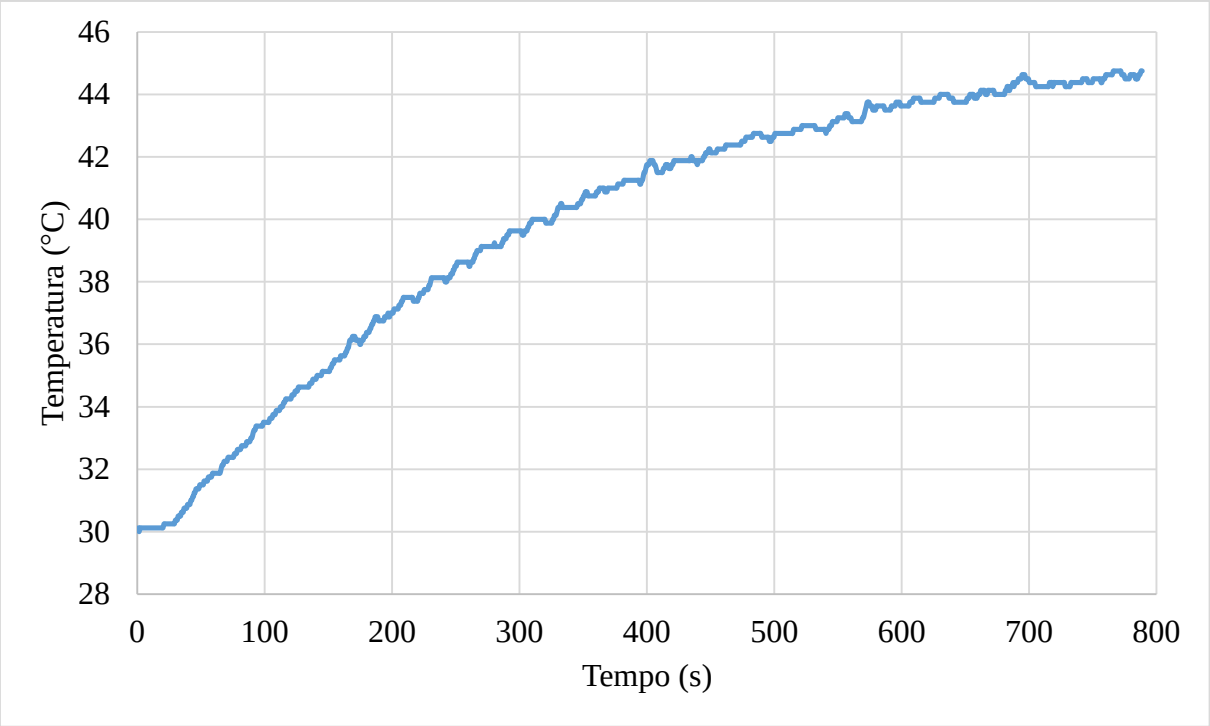

Figura 4.1 – Resposta ao degrau.

Fonte: Autor.

Desconsiderando o atraso na função de transferência experimental, pode-se fazer uma comparação da resposta ao degrau tanto da função de transferência experimental quanto a teórica. A Figura 4.2 mostra as respostas ao degrau feitas no Scilab.

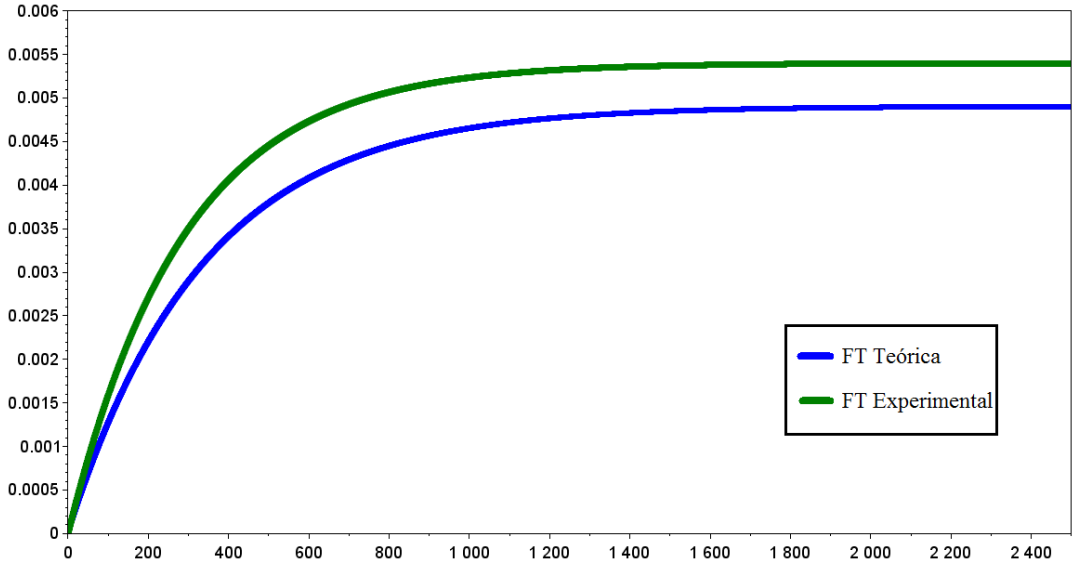

Figura 4.2 – Resposta ao degrau teórica e experimental.

Fonte: Autor.

Observando a Figura 4.2 pode-se notar que a função de transferência experimental se aproxima da função de transferência teórica, elas se distanciam apenas por um erro de 0,0005, portanto o módulo experimental é válido.

Para a sintonia do controlador utiliza-se o ganho K em relação a escala de 0 a 255 do microcontrolador Arduino devido a sua saída também estar nessa escala, a Equação 4.3 fornece a função de transferência experimental obtida com esse ganho utilizado.

$$G(s) = \frac{0{,}142}{287s + 1} e^{-20s} \tag{4.3}$$

A partir da função de transferência experimental encontrada, são calculados os valores das constantes do controlador utilizando o método de Ziegler-Nichols dado pela Tabela 2.1 para um controle PI e PID. A Tabela 4.1 mostra os valores das constantes encontrados.

Tabela 4.1 – Constantes do controlador encontradas.

Método	Controlador	K_p	K_i	K_d
Ziegler e Nichols	PI	91	1,4	-
	PID	121,3	3	1213

Fonte: Autor.

Com as constantes devidamente calculadas, primeiramente são inseridas no programa as constantes para um controlador PI e dada uma variação no *set point*. A Figura 4.3 apresenta a curva de resposta obtida e a Figura 4.4 apresenta o sinal da saída do controlador.

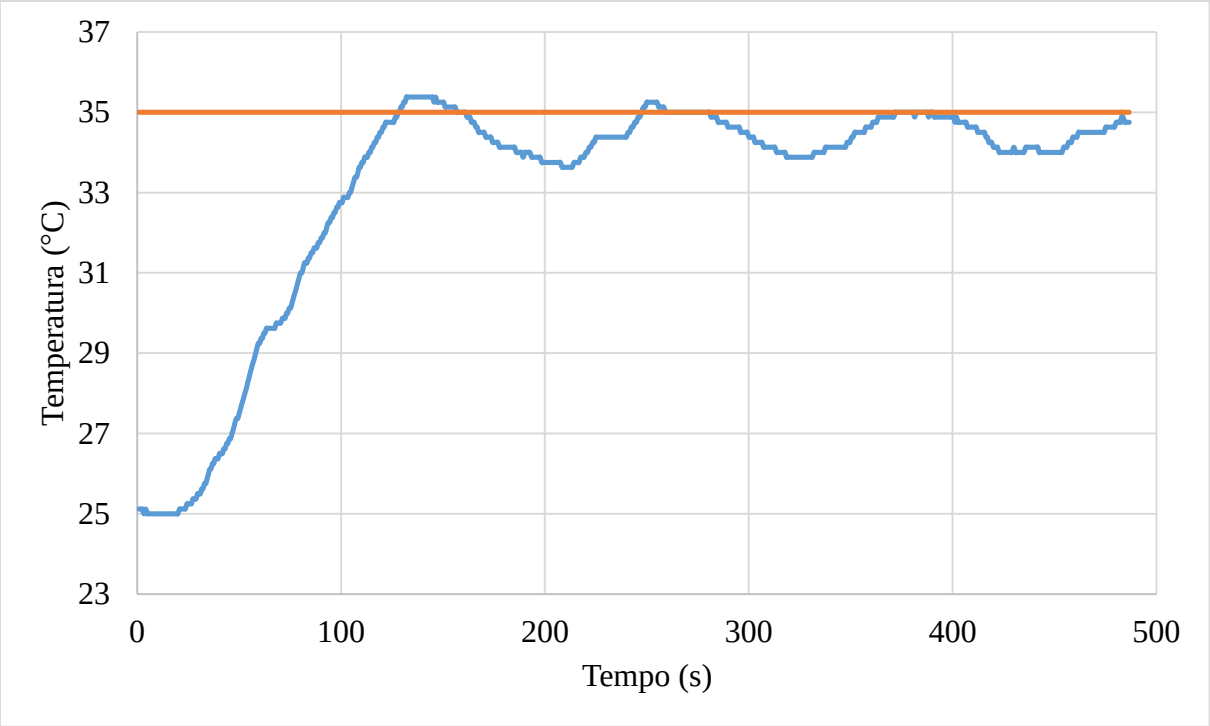

Figura 4.3 – Resposta do controlador PI.

Fonte: Autor.

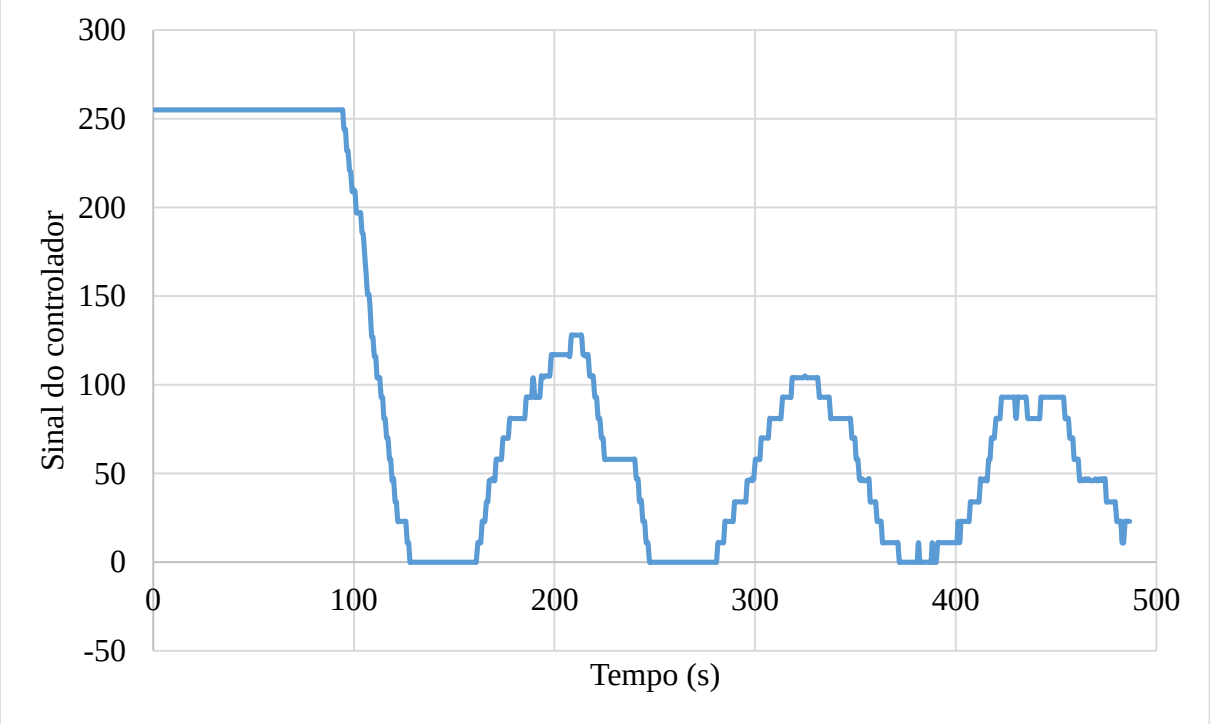

Figura 4.4 – Sinal de atuação do controlador PI.

Fonte: Autor.

Após o sistema retornar a temperatura ambiente é realizado o próximo teste, agora com o controlador PID. As constantes são inseridas no programa e é dada uma variação no *set point*. A Figura 4.5 apresenta a curva de resposta obtida e a Figura 4.6 apresenta o sinal da saída do controlador.

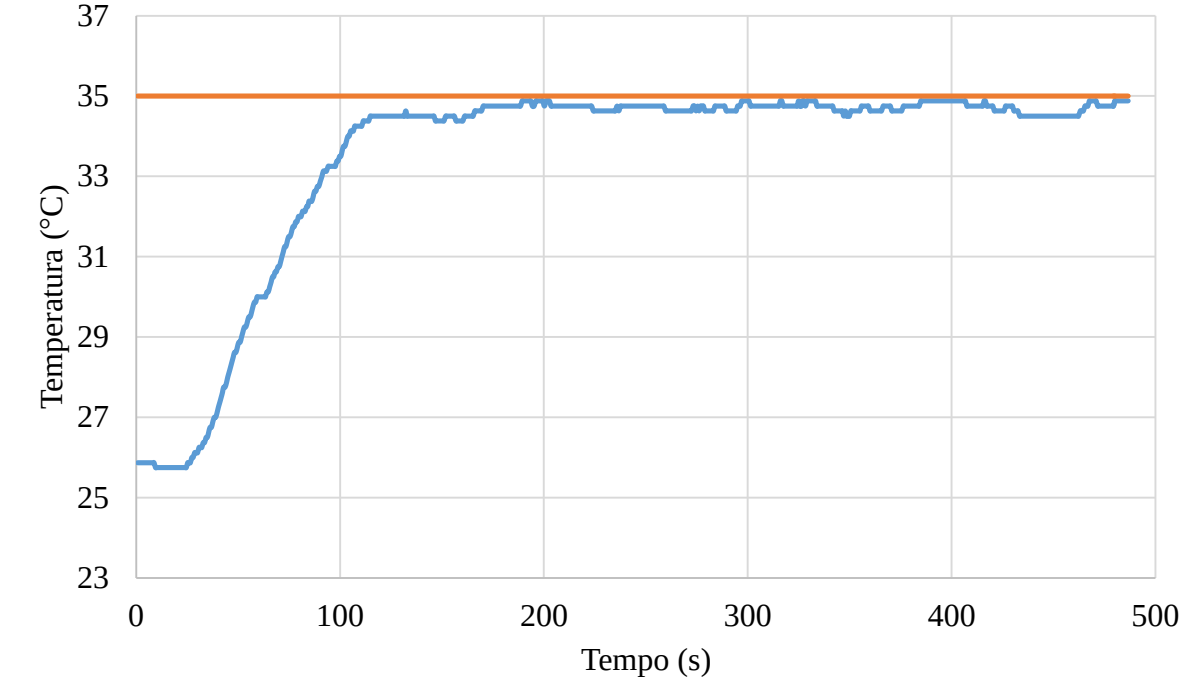

Figura 4.5 – Resposta do controlador PID.

Fonte: Autor.

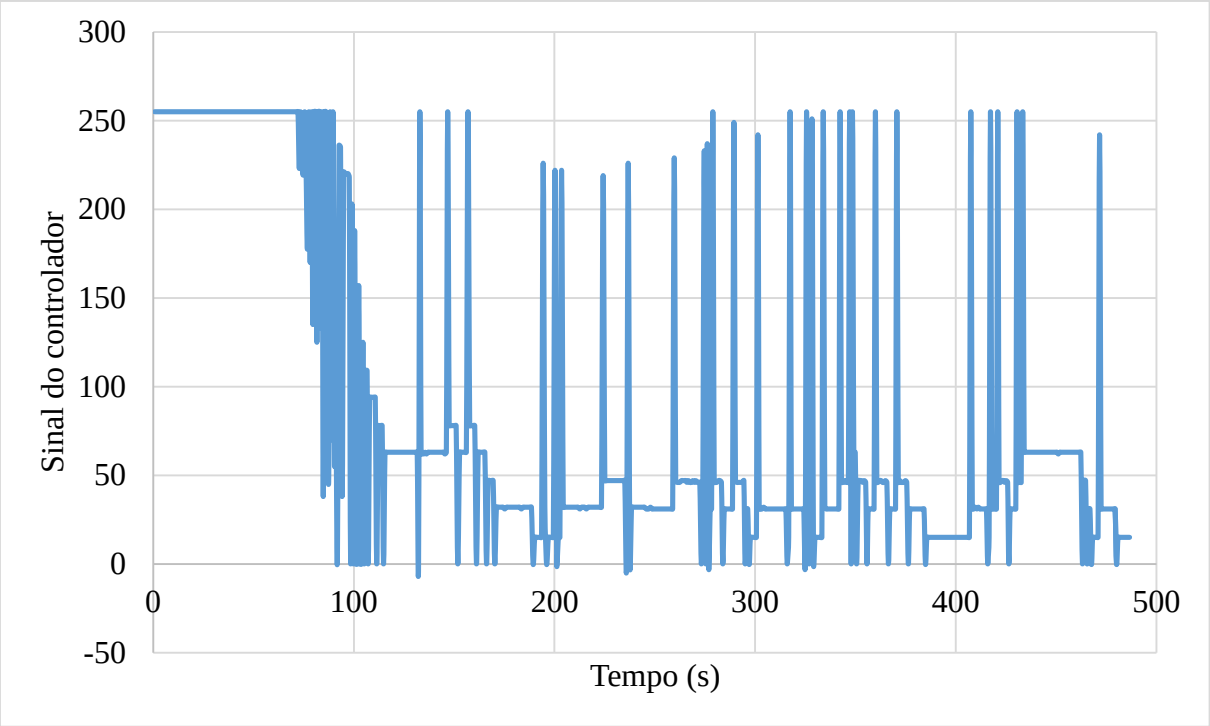

Figura 4.6 – Sinal de atuação do controlador PID.

Fonte: Autor.

Como pode ser observado as respostas se comportaram conforme o esperado e o descrito pela literatura. As duas respostas apresentaram a oscilação característica conforme vai se estabilizando.

A saída do controlador PI atuou conforme o esperado, entregando uma potência maior no início e reduzindo conforme se aproximava do *set point*. Já a saída do controlador PID acabou oscilando muito, funcionando de maneira aproximada a um controle liga-desliga, isso ocorreu devido ao fato da constante derivativa Kd ter apresentado um valor muito alto. Devido as características do sistema, como possuir uma constante de tempo elevada, o controle PID nesse caso não é muito indicado já que é um sistema lento, o controle PI se torna mais recomendado nesse caso.

Durante os testes o módulo experimental funcionou como o esperado, a interface gráfica, o sistema atuador, os sensores e o controlador funcionaram corretamente, não foi apresentado nenhum comportamento inesperado.

4.2. Experimento didático

Foi demonstrado para a turma do 5° ano de Engenharia Química o módulo experimental desenvolvido e foram realizados alguns experimentos junto com eles, onde, foi encontrado a função de transferência do sistema, calculado as constantes para um controle do tipo PI e PID utilizando o método de Ziegler-Nichols e feito uma variação no *set point* para os dois tipos de controle.

Durante os testes foi explicado para os alunos o funcionamento do sistema e a utilização da interface gráfica e obtenção dos dados. Os alunos compreenderam facilmente o funcionamento do sistema.

5. CONCLUSÃO E SUGESTÕES DE TRABALHOS FUTUROS

O módulo experimental desenvolvido funcionou perfeitamente bem conforme o esperado. Durante todos os testes feitos ele se mostrou uma ferramenta robusta e apresentou as curvas de respostas esperadas.

O método de sintonia de Ziegler-Nichols se mostrou um método adequado e rápido de sintonizar o controlador, o controle PI se mostra mais adequado devido a dinâmica do processo ser lenta.

Por fim, a utilização do módulo experimental se mostrou uma excelente ferramenta de ensino que possibilita: simular processos em uma escala menor, estudar diversos métodos de sintonia, ser utilizado por diferentes áreas da engenharia, além de permitir futuras melhorias e implementação de novos recursos, graças ao seu desenvolvimento ser baseado no software livre Scilab e na plataforma Arduino.

Sugere-se para trabalhos futuros a implementação de um controle de nível no sistema e um possível aumento na potência do atuador, reduzindo assim o tempo de resposta do sistema.

Também sugere-se um estudo quantitativo da aplicação do módulo experimental no ensino da disciplina de controle.

6. REFERÊNCIAS

ARAÚJO, André M. de et al. CONTROLE E MONITORAÇÃO DE TEMPERATURA EM MALHA FECHADA: CONSTRUÇÃO DE UM MÓDULO DIDÁTICO. Cobenge, Brasilia, 2004.

ARDUINO. https://www.arduino.cc/ Acesso em: 04 de junho de 2018.

BARROS, Fabio Fernandes de. SISTEMAS DE CONTROLE DE NÍVEL E DE TEMPERATURA DE UMA BANCADA DIDÁTICA DO LTTC. 2013. 92 f. TCC (Graduação) - Curso de Engenharia Mecânica, Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2013.

BERTACHI, Arthur Hirata et al. IMPLEMENTAÇÃO DE UM PID DIGITAL EM AMBIENTE COMPUTACIONAL APLICADO A UMA PLANTA DIDÁTICA PARA ENSINO DE CONTROLE PARA ENGENHARIA. COBENGE, Gramado, 2013.

CARVALHO, Renato T. de et al. MÓDULO LABORATORIAL PARA EDUCAÇÃOEMCONTROLE, EMTEMPO REAL, BASEADO EM LINUX/RTAI. XVIII Congresso Brasileiro de Automática, Bonito, 2010.

CENTRAL DO ACRÍLICO. http://www.centraldoacrilico.com.br/produtos/chapa-de-acrilico/acrilico-cast Acesso em: 11 de maio de 2018.

CERCHIARO, Denis Fava. CONTROLE DE TEMPERATURA DE UM TROCADOR DE CALOR. 125 f. Dissertação (Mestrado) - Curso de Engenharia de Sistemas, Poli-usp, São Paulo, 2006.

COELHO, Antônio Augusto Rodrigues; COELHO, Leandro dos Santos. Identificação De Sistemas Dinâmicos Lineares. Florianópolis: Editora da Ufsc, 2004.

COUGHANOWR, Donald R.; LEBLANC, Steven E.. PROCESS SYSTEMS ANALYSIS AND CONTROL. 3. ed. New York: Mcgraw-hill, 2009.

EMBARCADOS. https://www.embarcados.com.br/domine-o-pwm/ Acesso em: 03 de junho de 2018.

GNOATTO, Felipe et al. PROJETO E DESENVOLVIMENTO DE MÓDULO DE CONTROLE DE NÍVEL EM ESCALA PILOTO. Engevista, v. 18, n. 2, p.280-293, dez. 2016.

GONZAGA, Diego Augusto. CONTROLE MULTIVARIÁVEL UTILIZANDO PLATAFORMA ARDUINO PARA SECADOR DE PLANTAS MEDICINAIS E SOFTWARE DE AQUISIÇÃO DE DADOS. 47 f. Dissertação (Mestrado) - Curso de Mestrado em Engenharia Agrícola, Universidade Federal de Viçosa, Viçosa, 2015.

MARTIN, Paulo Alexandre; KASSAB JUNIOR, Fuad. USO DE TROCADOR DE CALOR COMO FERRAMENTA DIDÁTICA PARA O ENSINO DE MODELAGEM E SISTEMAS DE CONTROLE. Revista de Ensino de Engenharia, v. 25, n. 2, p.3-9, 2006.

MAXIM INTEGRATED. DATASHEET DS18B20. Maxim Integrated, v. 92, p. 20, 2015.

NISE, Norman S.. ENGENHARIA DE SISTEMAS DE CONTROLE. 6. ed. Rio de Janeiro: Ltc, 2012.

O'DWYER, Aidan. Handbook Of PI And PID Controller Tuning. 3. ed. London: Imperial College Press, 2009.

RAMOS, Adriano Peixoto; WENSE, Gabriel Lula Barros. SISTEMA DIDÁTICO DE NÍVEL DE LÍQUIDOS. 2008. 89 f. TCC (Graduação) - Curso de Engenharia de Controle e Automação, Universidade de Brasília, Brasília, 2008.

SANTOS, Carla M. M. dos et al. DESENVOLVIMENTO DE UM MODULO DE CONTROLE DE NIVEL UTILIZANDO O KIT ARDUINO UNO. Anais do XX Congresso Brasileiro de Automática. Belo Horizonte, 2014.

SCHMID, C. REMOTE EXPERIMENTATION TECHNIQUES FOR TEACHING CONTROL ENGINEERING. 4th International Scientific – Technical Conference, Process Control. Czech Republic, 2000.

SCILAB. https://scilab.io/ Acesso em: 04 de junho de 2018.

SODRÉ, Ulysses. MODELOS MATEMÁTICOS. Matemática, UEL. Londrina, 2007.

APÊNDICE A

Código de programação desenvolvido no Scilab:

```
//Limpa
clc
//Pergunta porta de entrada do Arduino
port_name=evstr(x_choose(['  1';'  2';'  3';'  4';'  5'],['Selecione o número da porta: ']))
if port_name == 0 then
   delete(f);
   mclose;
end
////////// Janela
f=figure('figure_position',[400,50],'figure_size',[878,631],'auto_resize','on','background',[33],'figure_name','Temperature Control','tag','mainWindow','menubar_visible','off','toolbar_visible','off');
////////// Botões
handles.dummy = 0;
handles.obj1= newaxes();handles.obj1.margins = [ 0 0 0 0];handles.obj1.axes_bounds = [0.0754060,0.4299242,0.8665893,0.5321970];
handles.Iniciar=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','ForegroundColor',[-1,-1,-1],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.0623666,0.8825757,0.142529,0.0606061],'Relief','default','SliderStep',[0.01,0.1],'String','Start','Style','pushbutton','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','Iniciar','Callback','iniciar')
handles.Parar=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','ForegroundColor',[-1,-1,-1],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.0623666,0.8036616,0.142529,0.0606061],'Relief','default','SliderStep',[0.01,0.1],'String','Stop','Style','pushbutton','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','Parar','Callback','Parar')
handles.Fechar=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','ForegroundColor',[-1,-1,-1],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.0623666,0.6458333,0.142529,0.0606061],'Relief','default','SliderStep',[0.01,0.1],'String','Close','Style','pushbutton','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','Fechar','Callback','Fechar')
handles.obj6=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','ForegroundColor',[-1,-1,-1],'HorizontalAlignment','left','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.2790255,0.7578788,0.1032483,0.0662879],'Relief','default','SliderStep',[0.01,0.1],'String','Kp','Style','text','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','obj6','Callback','')
handles.obj7=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','ForegroundColor',[-1,-1,-1],'HorizontalAlignment','left','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.4058623,0.7578788,0.1032483,0.0662879],'Relief','default','SliderStep',[0.01,0.1],'String','ki','Style','text','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','obj7','Callback','')
handles.obj8=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','ForegroundColor',[-1,-1,-1],'HorizontalAlignment','left','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.5326992,0.7578788,0.1032483,0.0662879],'Relief','default','SliderStep',[0.01,0.1],'String','kd','Style','text','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','obj8','Callback','')
handles.obj9=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','ForegroundColor',[-1,-1,-1],'HorizontalAlignment','left','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.659536,0.7578788,0.1032483,0.0662879],'Relief','default','SliderStep',[0.01,0.1],'String','SetPoint','Style','text','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','obj9','Callback','')
handles.OnOff=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','ForegroundColor',[-1,-1,-1],'HorizontalAlignment','left','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.2574246,0.6480303,0.1693735,0.0625],'Relief','
```

```
default','SliderStep',[0.01,0.1],'String','On/Off','Style','radiobutton','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','
OnOff','Callback','OnOff')
handles.feedforward=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','left','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.4392575,0.6480303,0.1693735,0.0625],'Relief','
default','SliderStep',[0.01,0.1],'String','Feedforward','Style','radiobutton','Value',[0],'VerticalAlignment','middle','Visible','on','
Tag',' feedforward','Callback',' feedforward')
handles.feedback=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','left','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.6210905,0.6480303,0.1693735,0.0625],'Relief','
default','SliderStep',[0.01,0.1],'String','PID
Feedback','Style','radiobutton','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','feedback','Callback','feedback')
handles.kp=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','left','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.3272668,0.7578788,0.0551276,0.0662879],'Reli
ef','default','SliderStep',[0.01,0.1],'String','0','Style','edit','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','kp','Callba
ck','kp')
handles.ki=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','left','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.4539443,0.7578788,0.0551276,0.0662879],'Reli
ef','default','SliderStep',[0.01,0.1],'String','0','Style','edit','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','ki','Callbac
k','ki')
handles.kd=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','left','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.5826218,0.7578788,0.0551276,0.0662879],'Reli
ef','default','SliderStep',[0.01,0.1],'String','0','Style','edit','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','kd','Callba
ck','kd')
handles.setpoint=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','left','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.7212993,0.7578788,0.0551276,0.0662879],'Reli
ef','default','SliderStep',[0.01,0.1],'String','20','Style','edit','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','setpoint','
Callback','setpoint')

handles.obj17=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[14],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[0.12,0.56,1],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.8596288,0.9306818,0.080
0464,0.0606061],'Relief','default','SliderStep',[0.01,0.1],'String','Input','Style','text','Value',[0],'VerticalAlignment','middle','Vi
sible','on','Tag','obj17','Callback','')
handles.entrada=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.8596288,0.8636616,0.0800464,0.0606061],'
Relief','default','SliderStep',[0.01,0.1],'String','
','Style','edit','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','entrada','Callback','entrada')
handles.obj19=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[14],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.8596288,0.7866414,0.0800464,0.0606061],'
Relief','default','SliderStep',[0.01,0.1],'String','Output','Style','text','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','o
bj19','Callback','')
handles.saida=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.8596288,0.7196212,0.0800464,0.0606061],'
Relief','default','SliderStep',[0.01,0.1],'String','
','Style','edit','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','saida','Callback','saida')
handles.obj20=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[14],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.8596288,0.6496212,0.0800464,0.0606061],'
```

```
Relief','default','SliderStep',[0.01,0.1],'String','Flow','Style','text','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','obj
19','Callback','')
handles.vazao=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.8596288,0.5796212,0.0800464,0.0606061],'
Relief','default','SliderStep',[0.01,0.1],'String','0','Style','edit','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','saida','
Callback','vazao')
handles.obj21=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[14],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.2796288,0.8636616,0.1500464,0.0606061],'
Relief','default','SliderStep',[0.01,0.1],'String','Step/Tem.Amb','Style','text','Value',[0],'VerticalAlignment','middle','Visible','on
','Tag','obj19','Callback','')
handles.degrau=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.4196288,0.8636616,0.0800464,0.0606061],'
Relief','default','SliderStep',[0.01,0.1],'String','0','Style','edit','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','saida','
Callback','vazao')
handles.obj22=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[14],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.5296288,0.8636616,0.1500464,0.0606061],'
Relief','default','SliderStep',[0.01,0.1],'String','Gain','Style','text','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','obj
19','Callback','')
handles.ganho=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','center','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.6696288,0.8636616,0.0800464,0.0606061],'
Relief','default','SliderStep',[0.01,0.1],'String','1','Style','edit','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','saida','
Callback','vazao')

handles.obj23=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','left','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.4309412,0.5870506,0.1235294,0.0506329],'Reli
ef','default','SliderStep',[0.01,0.1],'String','Ganho
FB+FF','Style','radiobutton','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','obj23','Callback','')

handles.fffbgain=uicontrol(f,'unit','normalized','BackgroundColor',[-1,-1,-
1],'Enable','on','FontAngle','normal','FontName','Tahoma','FontSize',[12],'FontUnits','points','FontWeight','normal','Foregroun
dColor',[-1,-1,-
1],'HorizontalAlignment','left','ListboxTop',[],'Max',[1],'Min',[0],'Position',[0.5526471,0.5870506,0.0644118,0.0506329],'Reli
ef','default','SliderStep',[0.01,0.1],'String','0.5','Style','edit','Value',[0],'VerticalAlignment','middle','Visible','on','Tag','fffbgain',
'Callback','')

//////////
// Callbacks are defined as below. Please do not delete the comments as it will be used in coming version
//////////
top_axes_bounds = [0.0754060,0.4299242,0.8665893,0.5321970];
bottom_axes_bounds = [0.0754060,0.4299242,0.8665893,0.5321970];
//limites do gráfico
minTempDisplay = 0;
maxTempDisplay = 80;
timeBuffer = 200;
//Grafico do sensor 1
subplot(222);
a = gca();
a.axes_bounds = top_axes_bounds;
a.tag = "minuteAxes";
plot2d(0:timeBuffer, zeros(1,timeBuffer + 1), color("blue"));
a.title.text="Last 5 minutes";
a.data_bounds = [0, minTempDisplay; timeBuffer, maxTempDisplay];
e = gce();
e = e.children(1);
e.tag = "minutoSensor1";
```

```
//Grafico do sensor 2
a = newaxes();
a.y_location = "right";
a.filled = "off"
a.axes_bounds = top_axes_bounds;
plot2d(0:timeBuffer, zeros(1,timeBuffer + 1), color("black"));
a.data_bounds = [0, minTempDisplay; timeBuffer, maxTempDisplay];
a.axes_visible(1) = "off";
a.foreground=color("black");
a.font_color=color("black");
e = gce();
e = e.children(1);
e.tag = "minutoSensor2";
//Grafico da atuação do controlador
a = newaxes();
a.y_location = "right";
a.filled = "off"
a.axes_bounds = top_axes_bounds;
plot2d(0:timeBuffer, zeros(1,timeBuffer + 1), color("orange"));
a.data_bounds = [0, minTempDisplay; timeBuffer, maxTempDisplay];
a.axes_visible(1) = "off";
//a.foreground=color("orange");
//a.font_color=color("orange");
e = gce();
e = e.children(1);
e.tag = "atuador";
//Linha de SetPoint
%Setpoint=20;
plot([0, 200], [%Setpoint, %Setpoint]);
e = gce();
e = e.children(1);
e.tag = "instantMinTemp";
e.line_style = 5;
e.thickness = 2;
e.foreground = color("green");
e.data(:,2) = %Setpoint;
lastErr=0;

//Funções secundárias

function
   global %serial_port1
   saida= 'L;'+'L;';
   writeserial(%serial_port1,saida);
   closeserial(%serial_port1);
endfunction

function
   e = findobj("tag", "minuteSensor");
   e.data(:, 2) = 0;
   clc
   clear
endfunction

function
   Parar();
   global %serial_port1
   f = findobj("tag", "mainWindow");
   delete(f);
   mclose;
endfunction

function
   global %vazao
   Vazao1=0;
   Vazao1=get(handles.vazao,'String');
   ganho=get(handles.ganho,'String');
```

```
    ganho=evstr(ganho);
    %Vazao1=evstr(Vazao1);
    %Vazao2=255-%Vazao1;
    Vazao1=string(%Vazao2);

    //agora precisa calcular o a vazao equivalente para cada valor do 0 ao 255
    //foi encontrado 2 equações para intervalos distintos do pwm relacionado a vazão
    Vaztemp=0;
    if %Vazao1>=20 & %Vazao1<=112 then
       Vaztemp=873.73*log(%Vazao1)-2570.6;
       Vaztemp=Vaztemp*ganho;
       end
    if%Vazao1>112 & %Vazao1<=255 then
       Vaztemp=-0.0191*%Vazao1*%Vazao1+8.7598*%Vazao1+753.47;
       Vaztemp=Vaztemp*ganho;
    end
endfunction

function
    exec(vazao);
    global %Output;
        if data2< %Setpoint then
          Degrau=get(handles.degrau,'String');
          Output = string(Degrau);
          saida= Output+';'+Vazao1+';';
          writeserial(%serial_port1,saida);
        else
          Output = '0';
          saida= Output + ';'+Vazao1+';';
          writeserial(%serial_port1,saida);
        end
    timeChange=timer();    //mede o tempo entre cada medição da temperatura
    timeChange = evstr(timeChange);
    t=timeChange;
    t=string(timeChange);
    e=string(data1);
    s=string(data2);

endfunction

function
    global %Output;
    global %Q;
    global %TempAmp;
    Degrau=get(handles.degrau,'String');
    exec(vazao);
    %TempAmp=evstr(Degrau);
    if %Setpoint>data1 & %TempAmp<%Setpoint then
       Qperd=-22.973269*(%TempAmp - %Setpoint);
       Qprecisa=180*Vaztemp*(%Setpoint-data1)*0.00116299354;
       xx=15.18-Qperd-Qprecisa;
       p=poly([xx,87.439,- 1.1124,0.009,- 0.00003,0.00000004],'x','coeff');
       raizes=roots(p);
       posicao=1;
       while posicao<=5 do
          raiz1=evstr(string (real(raizes(posicao:posicao))));
          raiz2=evstr(string (imag(raizes(posicao:posicao))));
          if raiz2==0 then
              if raiz1<=255 & raiz1>=0 then
                %Output=round(raiz1);
                %Output=round(%Output);
              end
            end
          raiz1=0;
          raiz2=0;
          posicao = posicao + 1;
      end
```

```
      else
        %Output=0;
    end

    Output=round(%Output)
    Output=string(%Output)
    timeChange=timer();    //mede o tempo entre cada medição da temperatura
    timeChange = evstr(timeChange);
    t=timeChange;
    t=string(timeChange);
    ee=string(data1);
    ss=string(data2);

endfunction

//Função responsavel pelo controle PID
function
     exec(kp);
     exec(ki);
     exec(kd);
     exec(vazao);
     global %Output;
     timeChange=timer();    //mede o tempo entre cada medição da temperatura
     timeChange = evstr(timeChange);
     //Equação do controlador PID digital simples
     erro = %Setpoint - data2;
     errSum = (erro * timeChange);
     errSum = errSum + (erro * timeChange);
     dErr = (erro - lastErr) / timeChange;
     Output = Kp1 * erro + Ki1 * errSum + Kd1 * dErr;
    //conversão dos sinais
     Output=round(Output);
     Output=string(Output);
     ee=string(data1);
     s=string(data2);
     t=string(timeChange);
     lastErr = erro;
     data2=0;
     //restringindo saída do controlador
     %Output=evstr(Output);
        if(%Output > 255) then
           Output = '255'
        end
        if(%Output < 0) then
           Output = '0'
        end

endfunction

    Degrau=get(handles.degrau,'String');



function
    global %onoff
    onoff=0;
    onoff=get(handles.OnOff,'Value');
    %onoff=onoff;
endfunction

function
    global %Feedforward
    Feedforward=0;
    Feedforward=get(handles.feedforward,'Value');
    %Feedforward = Feedforward;
endfunction
```

```
function
   global %Feedback
   Feedback=0;
   Feedback=get(handles.feedback,'Value');
   %Feedback=Feedback;

endfunction

function
   global %Obj23
   Obj23=0;
   Obj23=get(handles.obj23,'Value');
   %Obj23=Obj23;

endfunction

function
   global %kp
   Kp=get(handles.kp,'String');
   Kp1=evstr(Kp);
endfunction

function
   global %ki
   Ki=get(handles.ki,'String');
   Ki1=evstr(Ki);
endfunction

function
   global %kd
   Kd=get(handles.kd,'String');
   Kd1=evstr(Kd);
endfunction

function
   global %Setpoint
   Setpoint=get(handles.setpoint,'String');
   %Setpoint=evstr(Setpoint);
   if  %Setpoint>50 then
      messagebox(["TEMPERATURA CRÍTICA" "Selecione um valor menor"], "AVISO!!")
      %Setpoint=50;
      set(handles.setpoint,"string",'50');
   end

   e = findobj("tag", "instantMinTemp");
   e.data(:,2) = %Setpoint;
endfunction

//function obj24_callback(handles)
 //
//endfunction

//Função Prinncipal
function
   arq=x_dialog('Nome do arquivo: ',".txt")
   fd = mopen(arq,'wt');
   timer();
   tic();
   global %MaxTemp
   global %serial_port1
   global %Acquisition
   %Acquisition = %t;
   global %fanStatus
   %fanStatus = 0;
   %serial_port1=openserial(port_name,"9600,n,8,1");
  // Arduino toolbox
  lastErr=0;
```

```
    values1=[];
    value1=ascii(0);
    while %Acquisition
      while(value1~=ascii(13)) then
       value1=readserial(%serial_port1,1);
       values1=values1+value1;
       v1=strsubst(values1,string(ascii(10)),'')
       v1=strsubst(v1,string(ascii(13)),'')
       s1=part(v1,1:5)
       s2=part(v1,8:13)
       s3=part(v1,16:20)
       data1=evstr(s1);
       data2=evstr(s2);
       data3=evstr(s3);
       end
    set(handles.entrada,"string",string(s1));
    set(handles.saida,"string",string(s2));
    values1=[]
    value1=ascii(0);
    //plotando os dados do sensor 1
    e=findobj('tag',"minutoSensor1");
    dataanterior2 = e.data(:, 2);
    e.data(:, 2) = [dataanterior2(2:$) ; data1];
    //plotando os dados do sensor 2
    e=findobj('tag',"minutoSensor2");
    dataanterior2 = e.data(:, 2);
    e.data(:, 2) = [dataanterior2(2:$) ; data2];
    //executando as funções para as variáveis do controle
    exec(OnOff);
    exec(feedback);
    exec(feedforward);
    exec(setpoint);
    exec(kp);
    exec(ki);
    exec(kd);
    exec(obj23);

    //codigo responsavel pelo controle
    if %onoff == 1 & %Feedback == 0 & %Feedforward == 0 then
        set(handles.obj21,'String','Degrau');
        exec(Onoff1);
        xinfo("Selecione o tipo de controle");
        mfprintf(fd,'Setpoint %f,Output %s,Tempo %s,Entrada %s,Saida %s,Vazão %f,P %f,I %f,D %f,TempAmp
%s,%s,\n',%Setpoint,Output,t,e,s,%Vazao1,Kp1,Ki1,Kd1,Degrau,'OnOff');

end

    //Codigo do FeedBack
    if %onoff == 1 & %Feedback == 1 & %Feedforward == 0 then
       exec(PID);
    //Limites para visualização no grafico
       k=(80*%Output)/255;
    //Envia os dados do controlador para o gráfico
       e=findobj('tag',"atuador");
       dataanterior2 = e.data(:, 2);
       e.data(:, 2) = [dataanterior2(2:$) ; k];
       saida= Output+';'+Vazao1+';';
       writeserial(%serial_port1,saida);
       xinfo("Feedback");
       mfprintf(fd,'Setpoint %f,Output %s,Tempo %s,Entrada %s,Saida %s,Vazão %f,P %f,I %f,D %f,TempAmp
%s,%s\n',%Setpoint,Output,t,ee,s,%Vazao1,Kp1,Ki1,Kd1,Degrau,'Feedback');
       tic();
    end

    //Codigo do FeedForward
    if %onoff == 1 & %Feedback == 0 & %Feedforward == 1 then
       set(handles.obj21,'String','Tem.Amb');
```

```
        exec(REALIM);
    //Limites para visualização no grafico
        k=(80*%Output)/255;
    //Envia os dados do controlador para o gráfico
        e=findobj('tag',"atuador");
        dataanterior2 = e.data(:, 2);
        e.data(:, 2) = [dataanterior2(2:$) ; k];
        saida= Output+';'+Vazao1+';'
        writeserial(%serial_port1,saida);
        xinfo("Feedforward");
        mfprintf(fd,'Setpoint %f,Output %s,Tempo %s,Entrada %s,Saida %s,Vazão %f,P %f,I %f,D %f,TempAmp
%s,%s\n',%Setpoint,Output,t,ee,ss,%Vazao1,Kp1,Ki1,Kd1,Degrau,'Feedforward');
        tic();
    end

    //Caso nada tenha sido selecionado
    if %Feedback == 0 & %Feedforward == 0 & %onoff == 0 then
        set(handles.obj21,'String','Degrau/Tem.Amb');
        exec(vazao);
        xinfo("Nenhuma opção selecionada");
        saida= 'L;'+Vazao1+';';
        writeserial(%serial_port1,saida);
        end

  end

endfunction
```

APÊNDICE B

Código de programação utilizado no Arduino:

```
#include <OneWire.h>
#include <DallasTemperature.h>
#define ONE_WIRE_BUS 10

OneWire oneWire(ONE_WIRE_BUS);
DallasTemperature sensors(&oneWire);
DeviceAddress sensor1, sensor2;

const int Actuatorresistencia = 11;
const int Actuatorbomba = 3;
int estado1;
int bomba;
int resistencia;
int indice;
String new_str[2]; //array de 2 posições
String tempString; //temporario
char a; //para armazenar o caracter da serial


void setup(){
  pinMode(Actuatorresistencia, OUTPUT);
  pinMode(Actuatorbomba, OUTPUT);
  Serial.begin(9600);
  sensors.begin();
  sensors.getAddress(sensor1, 0);
  sensors.getAddress(sensor2, 1);
  TCCR2B = TCCR2B & 0b11111000 | 0x07;
  //precisão do sensor (9, 10, 11, ou 12 bits)
  sensors.setResolution(sensor1, 11);
  sensors.setResolution(sensor2, 11);

}

void loop(){
  float inByte;
  estado1=0;
  sensors.requestTemperatures();
  float tempC1 = sensors.getTempC(sensor1);
  float tempC2 = sensors.getTempC(sensor2);
  if(!Serial.available()){
    Serial.print(tempC1);
    Serial.print("   ");
    Serial.print(tempC2);
    Serial.print("   ");
    Serial.print(bomba);
    Serial.println();
  }
  indice=0;
    while (Serial.available() > 0) {
      tempString = " "; //Um espaço para esvaziar a string.
      while(true){
         a = Serial.read(); //leia um caractere
         if(a == ';') break; //ser for ponto-e-virgula, sai do while
         else tempString += a; //senao, adiciona na string temporaria
      }
      new_str[indice] = tempString; //armazeno no array de strings, se for
necessario utilizar adiante
```

```
      indice++; //incremento o contador
      resistencia=new_str[0].toInt();
      bomba=new_str[1].toInt();
      analogWrite(Actuatorresistencia, resistencia);
      analogWrite(Actuatorbomba, bomba);
    }
}
```