UNIVERSIDADE FEDERAL DE MATO GROSSO
FACULDADE DE ENGENHARIA
ENGENHARIA QUÍMICA

Felipe Santos Pulcherio

CONTROLE DE NÍVEL PID INFERENCIAL EM TANQUE COM UTILIZAÇÃO DE ARDUINO

CUIABÁ - MT

Maio, 2021

Felipe Santos Pulcherio

CONTROLE DE NÍVEL PID EM TANQUES EM SÉRIE COM UTILIZAÇÃO DE ARDUINO

Trabalho de conclusão de curso apresentado como requisito de finalização do período de graduação do curso de Engenharia Química pela Universidade Federal de Mato Grosso, Campus Várzea Grande.

Orientador: Prof. Dr. Júlio Cesar de Carvalho Miranda

CUIABÁ - MT

Maio, 2021

RESUMO

O controle de nível de fluidos está presente na grande maioria das plantas industriais, este pode ser simulado pelo uso de unidades didáticas e plantas pilotos, que têm como função qualificar o usuário e treiná-lo quanto ao uso dos instrumentos envolvidos. O controlador PID é um exemplo comum neste contexto devido a sua grande aplicabilidade. Este trabalho tem por objetivo a construção e operação de uma unidade didática modular de baixo custo capaz de realizar o controle de nível visando demonstrar o funcionamento e manuseio do controlador PID. Para isto, foi desenvolvido um algoritmo que opera em arduino em conjunto com um suplemento para elaboração de visuais gráficos em tempo real de execução. Prevê-se que o uso desta unidade possibilite o desenvolvimento da capacitação técnica e do pensamento crítico aplicado à resolução de problemas e implementação de soluções de acordo com os aprendizados em sala de aula.

Palavras-chaves: Controle PID. Unidade didática. Arduino. Controle de nível. Controle de Processos.

ABSTRACT

Fluid level control is present in the vast majority of industrial plants, and it can be simulated by the use of teaching units and pilot plants, whose function is to qualify the user and instruct the use of instruments involved. The PID controller is a common example in this context due to its wide applicability. This work aims to build a low cost modular teaching unit capable of performing level control in order to demonstrate operation and handling of a PID controller. To reach this, an algorithm was developed using Arduino interface in conjunction with a plotter to elaborate graphic visuals in real time. It is expected that using this unit will provide development of critical thinking applied to problem solving and applicability of solutions learned in a classroom environment.

Keywords: PID Control. Teaching Unit. Arduino. Level Control. Process Control.

LISTA DE ILUSTRAÇÕES

Figura 2.1 – Fluxograma do controle inferencial ……....3
Figura 2.2 – Fluxograma do controle em malha fechada ……....3
Figura 2.3 – Fluxograma do controle em malha aberta ……..4
Figura 2.4 – Diagrama de blocos de um sistema de 1ª Ordem …….....5
Figura 2.5 – Diagrama de blocos de um controlador PID ……...6
Figura 2.6 – Comportamento do ganho proporcional em um controlador P ……...7
Figura 2.7 – Comportamento do tempo integral em um controlador PI …….....8
Figura 2.8 – Comportamento do tempo derivativo em um controlador PD …….....9
Figura 2.9 – Caracterização de um curva de resposta a partir de uma perturbação degrau em malha aberta …….....9
Figura 2.10 – Mapa de sintonização para um controlador PID. As curvas representam a resposta de uma perturbação degrau unitária para um valor de T_d fixo …….....……....10
Figura 3.1 – Vista frontal da unidade didática …….....12
Figura 3.2 – Bomba centrífuga utilizada ……...…....15
Figura 3.3 – Célula de carga de 50 Kg …….....16
Figura 3.4 – Circuito elétrico de uma ponte de wheatstone ..……...17
Figura 3.5 – Conversor amplificador analógico-digital HX711 ……..…..…....17
Figura 3.6 – Sensor de fluxo YF-S201 …….....18
Figura 3.7 – Ímã próximo (a) e afastado (b) do sensor hall …….....19
Figura 3.8 – Caixa de madeira onde estão instalados os componentes elétricos …….....20
Figura 3.9 – Circuito do Microcontrolador ……...22
Figura 3.10 – Circuito da balança …….....23
Figura 3.11 – Circuitos externos. a) Bomba 2. b) Válvulas solenoides ……....24
Figura 3.12 – Painel elétrico …….....25
Figura 3.13 – Software TelemetryViewer v0.7 ……....26
Figura 3.14 – Inicialização dos dados CSV …….....26
Figura 3.15 – Layout gráfico proposto …….....27
Figura 3.16 – Tipos de exportação de dados …….....27
Figura 3.17 – IDE Arduino …….....28
Figura 4.1 - Modelagem do tanque de controle ……....31
Figura 4.2 - Altura do fluido no tanque x Vazão de saída, válvula 60% aberta …….....32
Figura 4.3 – Curva de resposta a perturbação no sistema em malha-aberta (kg x seg) …….. 34

Figura 4.4 - Diagrama de blocos do sistema real …….....34
Figura 4.5 – Curva de resposta ao salto em malha-fechada …….....36
Figura 4.6 – Curva de resposta do sistema a perturbação pulso ……...37
Figura 4.7 – Curva de resposta do sistema a perturbação rampa …….....37

LISTA DE QUADROS

Quadro 4.1 – Comparativo das respostas geradas para diferentes perturbações …….....38

LISTA DE ABREVIATURAS E SIGLAS

PID	Proporcional Integral Derivativo
u	Sinal de Entrada ou Sinal de Controle
y	Sinal de Saída
e	Erro
SP	Set-point
SPO	Sistema de Primeira Ordem
EDO	Equação Diferencial Ordinária
K	Ganho Proporcional
T_i	Tempo Integral
T_d	Tempo Derivativo
P	Proporcional
PI	Proporcional Integral
PD	Proporcional Derivativo
mV	Milivolt
mA	Miliampere
mΩ	Miliohm
DC	Corrente Contínua
cm	Centímetro
cm^2	Centímetro quadrado
PVC	Policloreto de Vinila
IDE	Ambiente de Desenvolvimento Integrado
KB	Kilobyte
EM	Energia Mecânica
EC	Energia Cinética
EP	Energia de Pressão
Vdc	Volts em Corrente Contínua
L	Litro
h	Hora
min	Minuto
s	Segundo
A	Amperes

MLP	Modulação por Largura de Pulso
PWM	Pulse Width Modulation
Kg	Quilograma
R	Resistor
AD	Analógico Digital
mm	Milímetro
KΩ	Kiloohm
GND	Terra
LD	Liga Desliga
CSV	Valores Separados por Vírgula
USB	Porta Serial Universal
C++	Linguagem de Programação C++
ρ	Densidade
EE	Estado Estacionário
ET	Estado Transiente
M	Massa
F_{in}	Vazão de entrada
F_{out}	Vazão de Saída
ρ_{in}	Densidade na entrada
ρ_{out}	Densidade na saída
V	Volume
h	Altura

SUMÁRIO

1. INTRODUÇÃO ……..…….……... 1

1.1 Objetivos Gerais …….……...… 2

1.2 Objetivos Específicos ……..….. 2

2. REVISÃO DA LITERATURA ……..…… 2

2.1 Sistemas de Controle ……...…….. 2

2.1.1 Malha Aberta e Malha Fechada …….…….. 3

2.2 Modelo de Sistemas Dinâmicos ……..…….. 4

2.2.1 Sistemas de Primeira Ordem ……..…….……..5

2.3 Controlador PID ……..…...…….6

2.3.1 Ação Proporcional ……..…….....7

2.3.2 Ação Integral ……7

2.3.3 Ação Derivativa ……....8

2.3.4 Sintonização …….....9

2.4 Instrumentação ……..……...10

2.4.1 Calibração ……...11

3. DESCRIÇÃO DO PROTÓTIPO ……..11

3.1 Descrição da Unidade Didática ……...12

3.2 Projeto em Hardware ……..14

3.2.1 Componentes do Sistema ……....14

3.2.2 Painel Elétrico ……20

3.3 Projeto em Software ……24

3.3.1 Painel de Controle ……...24

3.3.2 Código Fonte da Unidade ……...27

4. RESULTADOS E DISCUSSÕES …….....……..30

4.1 Balanço Dinâmico e Elaboração do Modelo ……..30

4.2 Sintonização do Controlador PID ……...……...33

4.3 Comportamento do Controlador PID……35

5. CONCLUSÃO ……....38

REFERÊNCIAS BIBLIOGRÁFICAS ……40

APÊNDICES ……...41

1. INTRODUÇÃO

Ao estudar engenharia química, são necessárias aos alunos ferramentas para desenvolver suas habilidades técnicas e experimentais, sendo de extrema importância reforçar a compreensão dos conceitos básicos e suas aplicações aos processos industriais. Um espaço dedicado a essa aplicação de conhecimento são os laboratórios que, por muitas vezes, não estão totalmente equipados ou possuem equipamentos que não mais condizem com a atualidade.

A modelagem física de uma etapa do processo em uma planta industrial pode ocorrer por meio do uso de unidades didáticas e plantas pilotos. Estas têm como função qualificar o usuário e treiná-lo quanto ao uso dos instrumentos envolvidos. Geralmente, as unidades são fabricadas com um único objetivo, sendo que o treinamento possui passos a serem seguidos, sempre tendendo a um resultado pré estabelecido. Este tipo de abordagem evita a estimulação da criatividade na solução de problemas[9].

Uma outra abordagem de ensino é a que se baseia no aprendizado por solução de problemas e realização de projetos[9]. Dessa forma, ao contrário da aquisição de unidades didáticas, os alunos envolvidos podem projetar e construir suas próprias unidades, e posteriormente, realizar uma análise de otimização e mudanças possíveis, desenvolvendo assim, as habilidades técnicas e práticas.

A grande maioria das plantas industriais, envolvem o controle automático do processo[1]. Dentre eles, um dos sistemas de controle mais comuns é o que aborda o controle de nível de fluidos, seja diretamente em alguma operação unitária, ou em outras etapas do processo. Assim, faz-se imprescindível ao aluno de engenharia química ter o conhecimento e domínio na execução de um controle de nível e sua instrumentação. Para tanto, ao longo deste trabalho serão apresentadas uma revisão bibliográfica sobre controle de processo e instrumentação industrial e, por fim, uma proposta de unidade didática para o estudo de controle de nível de fluido em um reservatório.

1.1 Objetivo Geral

O principal objetivo deste trabalho é propor um sistema de tanques modular e com controle de nível de fluido, de baixo custo e facilmente reproduzível, utilizando um microcontrolador com a execução do algoritmo de controle PID.

1.2 Objetivos Específicos

Para alcançar o objetivo principal, foram estabelecidos os seguintes objetivos específicos: montar um sistema de tanques em uma estrutura simples e resistente de madeira, utilizar um microcontrolador da família Arduino e implementar e ajustar o controle do nível de água utilizando um algoritmo PID.

2. REVISÃO

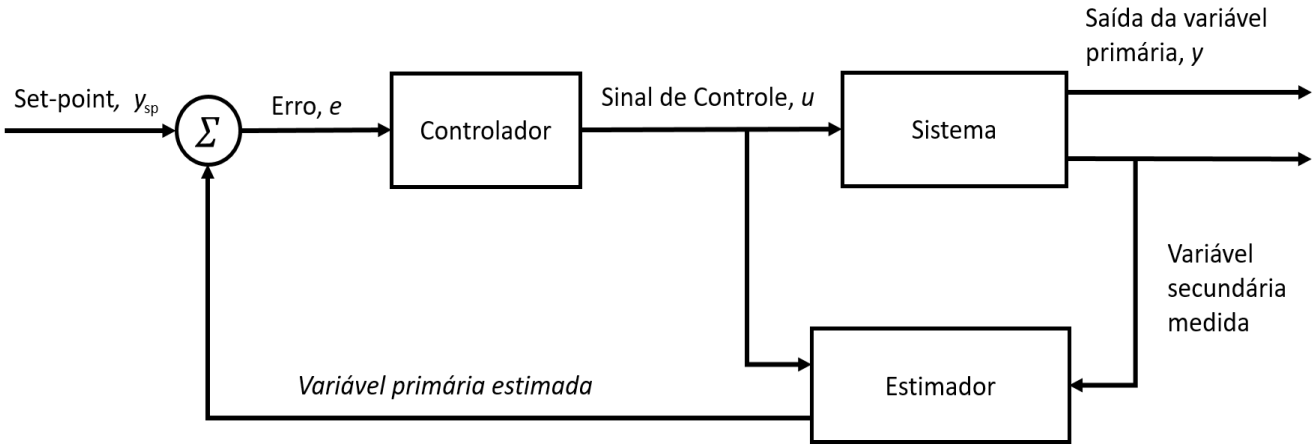
Neste capítulo é exposta a teoria mínima necessária para a compreensão do tema proposto. A *Seção 2.1* apresenta a teoria base sobre *Sistemas de Controle*, aprofundando na *Seção 2.1.1* sobre *Malha Aberta* e *Malha Fechada*. Na *Seção 2.2* é discutido o tema *Modelos de Sistemas Dinâmicos,* com o complemento da *Seção 2.2.1* sobre *Sistemas de Primeira Ordem*. A *Seção 2.3* visa esclarecer os conceitos envolvidos no *Controlador PID*, explicando cada ação de controle separadamente, onde a *Ação Proporcional* está na *Seção 2.3.1*, a *Ação Integral* está na *Seção 2.3.2*, e a *Ação Derivativa* está na *Seção 2.3.3*. Por último, a *Seção 2.4* expõe conceitos acerca da *Instrumentação*, enfatizando o tópico *Calibração* na *Seção 2.4.1*.

2.1 Sistemas de Controle

De acordo com Ogata (2010), um sistema pode ser definido como a combinação de componentes que agem em conjunto para atingir determinado objetivo. Este, é comumente separado em termos de *sinal de entrada u(s)* e *sinal de saída y(s)*. O primeiro é representado pelas *variáveis manipulada*s, ou *sinais de controle*, que são grandezas passíveis de alteração de valor por meio de um controlador. O segundo, é composto de *variáveis controladas*, que são grandezas cujos valores sofrem influência direta do *sinal de controle*.

Nos casos em que a *variável controlada* não pode ser mensurada, é possível estimar seu valor a partir de outras variáveis do processo que tenham maior facilidade de medição. Este método é chamado de *controle inferencial*, e a imagem a seguir o representa.

Figura 2.1 – Fluxograma do controle inferencial.



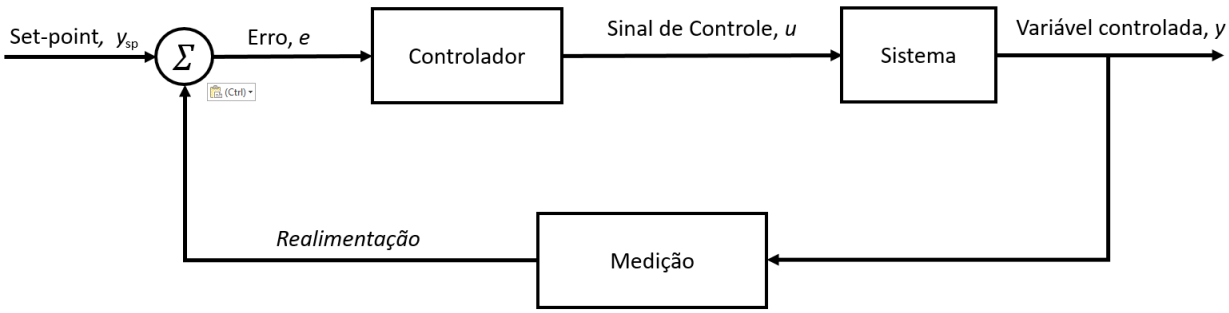
Fonte: Elaborado pelo autor.

Em suma, controlar é o ato de medir o valor de uma variável que se deseja manipular e inserir sinais de controle no sistema para corrigir ou atenuar desvios no valor medido, tendendo sempre ao *set-point*, ou *valor de referência* y_{sp}.

2.1.1 Malha Aberta e Malha Fechada

Um sistema de controle pode ser classificado de acordo com seu tipo de malha de controle. O controle em *malha fechada*, calcula a diferença entre o *sinal de saída y* e o *sinal de entrada u* e realimenta o *erro atuante e* em determinada etapa do processo, garantindo maior precisão no sinal de saída[1]. A malha fechada pode ser do tipo *feedback* (Figura 2.2), onde o *erro e* é realimentado na entrada, tornando o sistema relativamente insensível a distúrbios do meio, porém por conta das constantes correções, é possível o surgimento de oscilações no sinal de controle[4]. Ou ainda, do tipo *feedforward*, onde o *distúrbio v* pode ser medido e corrigido antes de sua influência na malha, ou seja, antes de resultar em erros no controle[6].

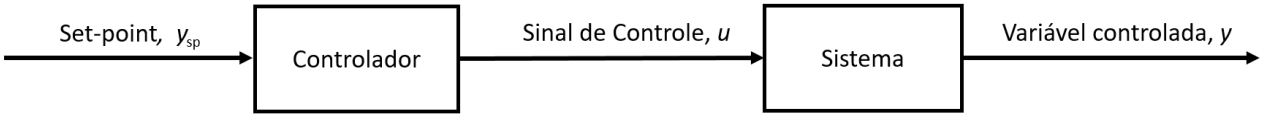
Figura 2.2 – Fluxograma do controle em malha fechada.



Fonte: Elaborado pelo autor.

Em contrapartida, o controle em *malha aberta* (Figura 2.3) é aquele que não sofre nenhuma influência do *sinal de saída y*, ou seja, as ações que ocorrem no processo são baseadas somente no tempo de sua execução[1]. Dessa forma, distúrbios podem alterar de maneira significativa o processo, visto que o erro não será corrigido. De maneira geral, opta-se pela *malha aberta* em sistemas mais simples com pouca exposição a distúrbios e alterações.

Figura 2.3 – Fluxograma do controle em malha aberta.



Fonte: Elaborado pelo autor.

2.2 Modelo de Sistemas Dinâmicos

Um *modelo matemático* é caracterizado por um conjunto de equações matemáticas (diferenciais e, eventualmente algébricas) cuja solução, considerando um conjunto de dados de entrada, é representativa da resposta dinâmica do modelo físico ou do processo[3]. As equações envolvidas neste tipo de representação são uma aproximação do processo real e, geralmente, não incorporam todas as características, sendo necessário estabelecer um propósito específico para sua criação.

A obtenção de um *modelo matemático* capaz de descrever o sistema real ou físico é imprescindível para dar início a análise do processo real[1]. Para Garcia (1997), a maioria dos processos industriais pode ser representada partindo da combinação de quatro elementos básicos: *ganho estático K*, *atraso de transporte*, *atraso de transferência* e *integrador*. Uma vez obtido o modelo, é possível analisar o desempenho e comportamento do sistema a partir de métodos já conhecidos.

Um dos critérios ao projetar um sistema é a resposta que este possui para um sinal de entrada ou a uma mudança das condições iniciais desses sinais[1]. Por existir uma correlação entre as características das respostas do sistema e um sinal de entrada, utiliza-se os sinais de teste (perturbações), como as funções rampa e degrau, para analisar a capacidade de o sistema responder a estes sinais.

De acordo com Ogata (2010), a resposta temporal de um sistema de controle pode ser representada pela *resposta transitória* $c_{tr}(t)$, a qual varia do estado inicial ao estado final, e

resposta estacionária $c_{ss}(t)$, sendo o comportamento do sinal de saída quando o tempo tende ao infinito. Assim, a *resposta c(t)* pode ser escrita como:

$$c(t) = c_{tr}(t) + c_{ss}(t) \tag{2.1}$$

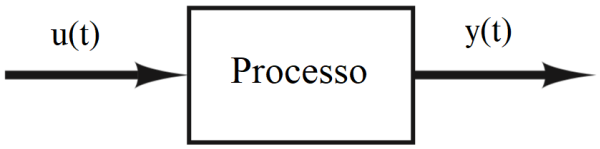
A *resposta transitória* corresponde aos atrasos na resposta de uma *variável*. Estes podem ser de dois tipos:

Atraso de transferência, sendo definido como o resultado de um efeito combinado entre propriedades do sistema que oferecem uma resistência à transferência de energia e/ou matéria, *elementos resistivos*, e partes do processo que armazenam energia e/ou matéria, *elementos capacitivos*[3]; e *Atraso de transporte* (*tempo morto*), sendo definido como o intervalo de tempo do transporte de massa ou energia de um ponto a outro do processo e durante o qual a perturbação ainda não atingiu o ponto de observação[3].

2.2.1 Sistemas de Primeira Ordem

Quando a resposta de um sistema é afetada por um *atraso de transferência* de 1º Ordem diz-se que o sistema é de 1ª Ordem (SPO)[3], este pode ser representado por um diagrama de blocos como na figura a seguir:

Figura 2.4 – Diagrama de blocos de um sistema de 1ª Ordem.



Fonte: Elaborado pelo autor.

Ao interpretar o sistema como uma equação diferencial ordinária (EDO) de primeira ordem, considerando a EDO linear, com coeficientes constantes e condição inicial nula (y(0)=0), seu comportamento dinâmico é descrito por:

$$a_1 \frac{dy(t)}{dt} + a_0 y(t) = b_0 u(t), \quad y(0) = 0 \tag{2.2}$$

$$\frac{a_1}{a_0}\frac{dy(t)}{dt} + y(t) = \frac{b_0}{a_0} u(t), \quad y(0) = 0 \tag{2.3}$$

Ao aplicar a transformada de Laplace na equação 2.3 faz-se possível estabelecer uma relação simples entre a entrada e saída do processo no domínio de frequência, chamada *função de transferência G(s)*:

$$G(s) = \frac{y(s)}{u(s)} = \frac{b_0}{a_1 s + a_0} \tag{2.4}$$

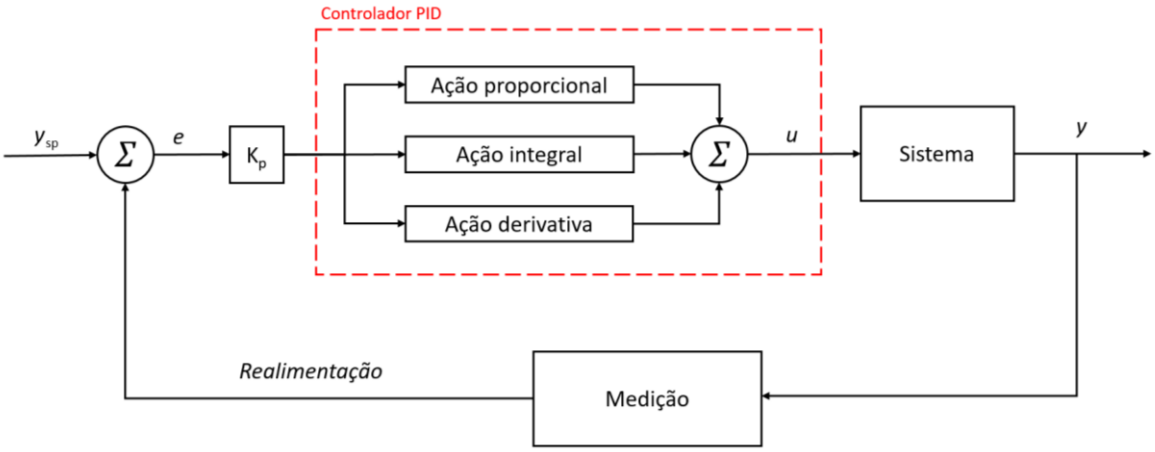
A forma padrão de um SPO é obtida ao reescrever a equação anterior utilizando as constantes $\tau = \frac{a_1}{a_0}$ e $K = G(0) = \frac{b_0}{a_1}$, onde a primeira representa a *constante de tempo do sistema* que indica a rapidez do sistema responder a uma variação na entrada[1], e a segunda o *ganho estático do processo* que mede a capacidade do sistema aumentar ou diminuir o sinal de entrada partindo de um *estado estacionário (EE)*.

$$G(s) = \frac{K}{\tau s + 1} \tag{2.5}$$

2.3 Controlador PID

O controlador Proporcional-Integral-Derivativo (PID) é um tipo de controlador que pode ser implementado em sistemas de malha fechada, operando a partir de três tipos de *ação de controle:* a *ação proporcional, integral e derivativa*. Dessa forma, o *sinal de controle final* é obtido a partir da soma dos *sinais de controle* de cada uma das ações [4], como mostra a Figura 2.5.

Figura 2.5 – Diagrama de blocos de um controlador PID.



Fonte: Elaborado pelo autor.

É possível representar este diagrama de blocos como uma equação no domínio de tempo, considerando a soma das três ações e admitindo os parâmetros: *sinal de controle u, erro e, ganho proporcional K, tempo integral* T_i, e *tempo derivativo* T_d.

$$u(t) = K\left(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d\,\frac{de(t)}{dt}\right) \qquad (2.6)$$

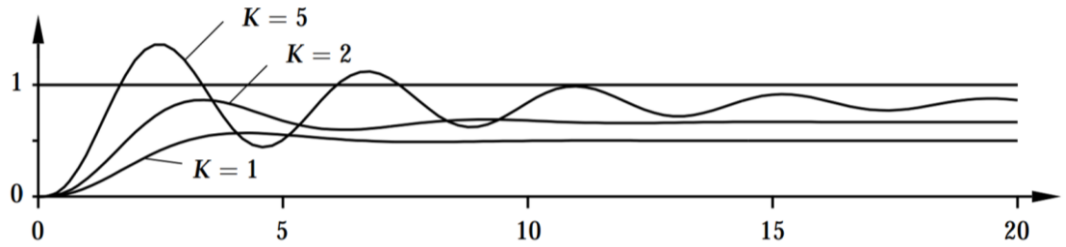
2.3.1 Ação Proporcional

A *ação proporcional* possui uma amplitude de correção proporcional à amplitude do erro. Quando usada de maneira isolada (controlador P), percebe-se que após a tentativa de correção do desvio na *variável manipulada*, o novo ponto de equilíbrio apresenta um *erro* [5]. Este, é chamado de *off-set* ou *erro de regime*. A equação a seguir caracteriza o controle puramente proporcional [4]:

$$u(t) = Ke(t) + u_b \qquad (2.7)$$

O *ganho proporcional K* relaciona a variação de saída do controlador e sua variação de entrada. Dessa forma, quanto maior o ganho, maior será a variação na saída do controlador, para uma mesma variação na variável [5]. A variável u_b é um *reset*, geralmente recebe o valor de 0, porém pode ser ajustado manualmente para minimizar os efeitos do *off-set* em um ponto de trabalho específico [4].

Figura 2.6 – Comportamento do ganho proporcional em um controlador P.



Fonte: Åström e Hägglund (2009).

O ajuste do *ganho proporcional K* influencia no *off-set* deixado pela correção, onde quanto maior o valor de *K* menor será o *off-set*, porém maior será a instabilidade (oscilação)[5]. Do contrário, menor a instabilidade e maior o *off-set*.

2.3.2 Ação Integral

A *ação integral* possui uma velocidade de correção proporcional à amplitude do desvio. Ao ser aplicada em um sistema, é capaz de reajustar automaticamente o novo valor de

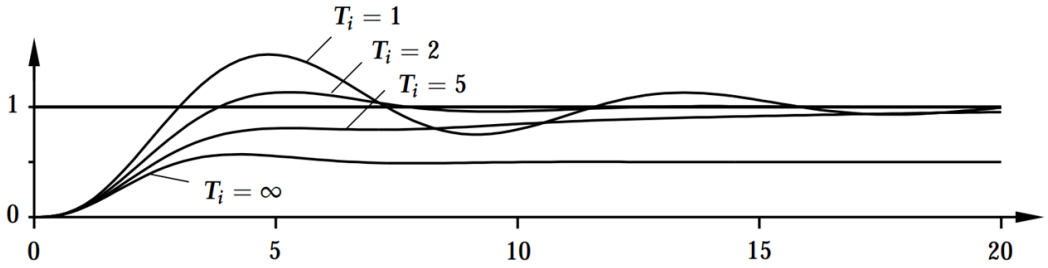
equilíbrio, dessa forma, assegura que a saída do processo coincida com o *set-point* ou reduza o *off-set* [5]. Esta ação geralmente possui uma correção muito lenta, sendo necessário utilizá-la em conjunto com outro controle.

Analisando a interação entre as ações *proporcional* e *integral* (controlador PI), percebe-se que a *ação integral* pode automaticamente resetar os valores de u_b na Equação 2.7, assim, afirma-se que um controlador que possua *ação integral* sempre resultará em *erro* zero em *estado estacionário (EE)* [4]. A equação a seguir caracteriza o controle PI [4]:

$$u(t) = K\ (\ e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau\ ) \qquad (2.8)$$

O *tempo integral* T_i é definido como o tempo, em minutos, necessário para que a *ação integral* repita uma vez o efeito da *ação proporcional* [5]. Ao analisar a Figura 2.7, percebe-se que quanto menor o T_i, mais rápida será a correção realizada pela *ação integral*, porém maior a oscilação.

Figura 2.7 – Comportamento do tempo integral em um controlador PI.



Fonte: Åström e Hägglund (2009).

2.3.3 Ação Derivativa

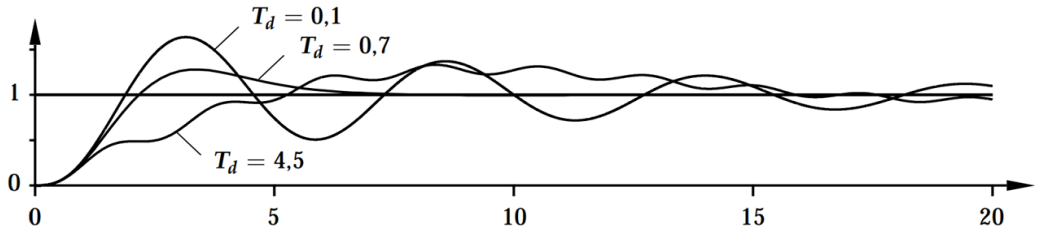
A *ação derivativa* possui uma velocidade de correção proporcional à velocidade do desvio. Quando utilizada em um sistema, é capaz de variar a saída em maior magnitude, ao comparar com o controlador P. Assim, quanto maior a distância entre a *variável manipulada* e o *set-point* mais rápida é a atuação, e quanto menor a distância, mais lenta é a atuação [5].

A interação entre as *ações proporcional* e *derivativa* (Controlador PD), resulta em uma menor amplitude das oscilações e, consequentemente, um menor tempo de estabilização. Entretanto, o modo derivativo não elimina o *off-set*, visto que não atuará em situações com *desvio permanente*. A equação a seguir caracteriza o controle PD [4]:

$$u(t) = K\ (\ e(t) + T_d\frac{de(t)}{dt}\ ) \qquad (2.9)$$

O *tempo derivativo* T_d é definido como o tempo, em minutos, em que a *ação derivativa* adianta o efeito da *ação proporcional* [5]. Uma observação da Figura 2.8, permite afirmar que quanto maior o T_d, mais rápido será alcançada a estabilidade do sistema pela *ação derivativa*.

Figura 2.8 – Comportamento do tempo derivativo em um controlador PD.



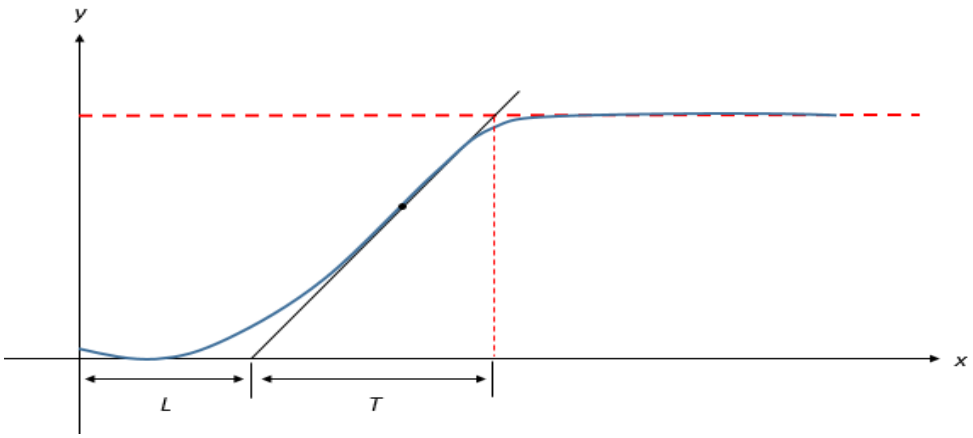
Fonte:Åström e Hägglund (2009).

2.3.4 Sintonização

Como visto, um controlador possui diversos parâmetros que devem ser ajustados. A malha de controle apresentará bons resultados com uma boa escolha destes parâmetros, caso contrário, pode apresentar falhas ou se tornar instável [6]. O procedimento de encontrar os melhores valores para os parâmetros chama-se *sintonização*.

Um dos métodos mais usuais de *sintonização* foi proposto por Ziegler e Nichols em 1942, principalmente por ser simples e intuitivo, e podem ser aplicados com pouco ou nenhum conhecimento do processo [6]. O *método da curva de reação*, consiste em aplicar um perturbação degrau no sistema em *malha aberta*, e na curva de resposta desenha-se uma tangente a partir do ponto de inflexão , obtendo-se os parâmetros L e T, sendo o primeiro a intersecção entre a tangente e o eixo y, e o segundo a distância entre L e a intersecção do novo estado estacionário com o eixo x. A figura a seguir explicita estes pontos:

Figura 2.9 – Caracterização de um curva de resposta a partir de uma perturbação degrau em malha aberta.



Fonte: Elaborado pelo autor.

Ziegler e Nichols estabeleceram os parâmetros de um controlador PID como funções de L e T [6]:

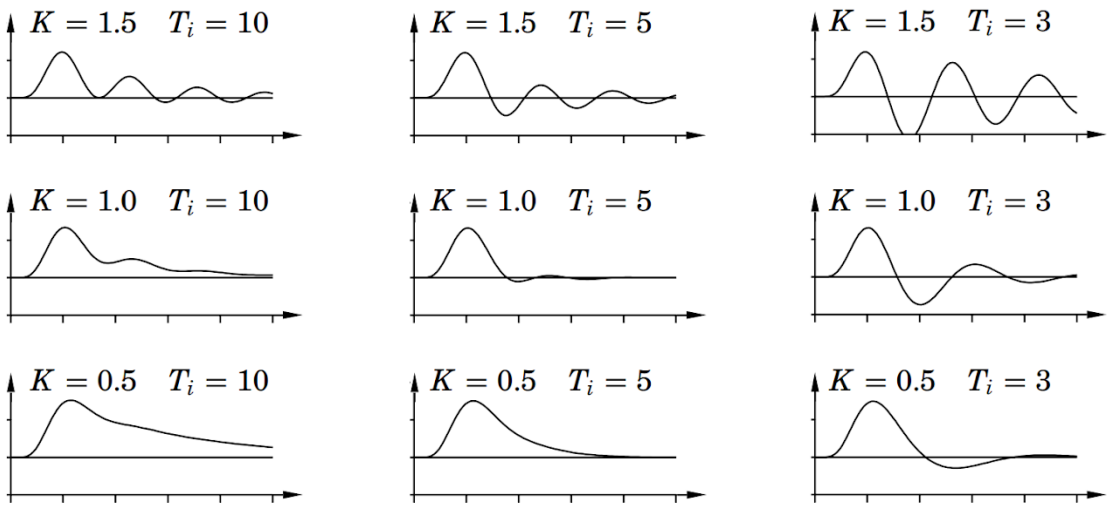
$$K = 1{,}2\frac{T}{L} \quad (2.10)$$

$$T_i = 2L \quad (2.11)$$

$$T_d = \frac{L}{2} \quad (2.12)$$

Faz-se necessário um ajuste fino manual dos parâmetros encontrados visto que o método em questão retorna somente um ponto de partida para estes valores. Assim é possível realizar um *mapeamento de sintonização* (*tuning maps*), cuja função é fornecer intuição sobre como as alterações nos parâmetros do controlador influenciam o comportamento do sistema de circuito fechado. Estes mapas são gráficos bidimensionais das respostas transientes, organizados de maneira sistemática, como por exemplo:

Figura 2.10 – Mapa de sintonização para um controlador PID. As curvas representam a resposta de uma perturbação degrau unitária para um valor de T_d fixo.



Fonte: Åström e Hägglund (2009).

2.4 Instrumentação

De acordo com Placko (2007) o termo instrumentação se refere a um grupo de ferramentas utilizadas na medição e metrologia, com o intuito de medir, indicar, registrar ou controlar as variáveis de um processo ou objeto.

Todo e qualquer instrumento de medida possui suas especificações que podem ser descritas pelos atributos: *Faixa de medição*, refere-se à diferença entre os extremos do intervalo

de medida do equipamento; *Resolução,* é o menor incremento que pode ser lido ou armazenado; *Sensibilidade,* é uma quantia absoluta que representa a menor variação detectável pelo instrumento em mV, mA ou mΩ; *Precisão,* é o menor valor que um instrumento pode ler repetidamente e de forma confiável.

Um exemplo útil de objeto de instrumentação é o transdutor de medição, que é um dispositivo capaz de gerar uma relação entre uma *variável de entrada m* e uma *variável de saída s*. Caso a variável de saída deste transdutor seja um sinal elétrico, passa a ser chamado *Sensor*. A relação que une *s* e *m* é:

$$s = F(m) \tag{2.13}$$

Esta depende principalmente da lei física que rege o sensor, sua estrutura, objetivo e ambiente de instalação.

2.4.1 Calibração

Todo instrumento deve passar por uma operação, chamada de calibração, com o intuito de se estabelecer a expressão *F(m)*. Para tanto, são feitos testes com diferentes valores de *m* (m_1, m_2, ..., m_i) e os sinais elétricos enviados pelo sensor (s_1, s_2, ..., s_i) são traçados, gerando assim, a *curva de calibração F(m)*[8].

De acordo com Placko (2007), caso não haja variáveis externas que influenciam na expressão, faz-se possível a utilização da calibração simples, que pode ser realizada de duas maneiras:

Por meio da *calibração direta* são obtidos os valores de *m* partindo-se de objetos-padrão ou de referência cujo valor de *s* já é conhecido, dada uma certa incerteza. Ou ainda pela *calibração por comparação*, onde para diversos *m*, os valores de *s* obtidos pelo sensor a ser calibrado é comparado com os valores de *s* dados por um sensor já calibrado. Em outras palavras, usa-se um instrumento calibrado como padrão onde a incerteza é conhecida.

3. DESCRIÇÃO DO PROTÓTIPO

Este é o maior capítulo presente no trabalho. Na *Seção 3.1 Descrição da Unidade Didática* estão detalhados os dados de construção e montagem, em seguida, na *Seção 3.2* é discutido sobre o *Projeto em Hardware* onde todo o hardware utilizado e seu princípio de funcionamento, assim como, os dados detalhados do sistema elétrico, se encontram. Por fim, é

apresentado na *Seção 3.3* o *Projeto em Software*, que descreve a operação da unidade, o algoritmo de controle e suas funcionalidades.

3.1 Descrição da Unidade Didática

A unidade proposta foi dividida em 7 partes estruturais, sendo elas os recipientes contendo o fluido circulante, um tanque com o sistema de controle, a estrutura metálica juntamente com os suportes em madeira, as bombas e o painel elétrico. Cada parte recebe um título para facilitar sua identificação, conforme demonstrado na Figura 3.1. O *reservatório 1* é o recipiente fabricado em polietileno, que alimenta a *bomba 1*. Esta envia o fluido para o *tanque de controle*, que possui o nível medido e tem saída conectada ao *reservatório 2*. A *bomba 2*, eleva o fluido do *reservatório 2* para o *reservatório 1*. Por fim, a *estrutura* é o suporte composto por tábuas de compensado e barras roscadas e o *painel elétrico* contém o microcontrolador, a parte elétrica, e os botões de acionamento da unidade.

Figura 3.1 – Vista frontal da unidade didática.

Fonte: Elaborado pelo autor.

A *estrutura* é responsável por dar suporte e sustentação aos componentes da unidade, além de estabelecer posições fixas para estes. Como o custo foi o fator principal neste protótipo optou-se pela utilização de tábuas de madeira compensada com 1 cm de espessura em conjunto com barras roscadas de ferro com 1 metro de comprimento e ½ polegada de diâmetro. As elevações das tábuas foram definidas de modo que a distância entre elas fosse igual e também fosse possível alocar entre elas os *reservatórios 1 e 2*, assim, foram utilizadas porcas e arruelas de ½ polegada na parte superior e inferior de cada tábua. Optou-se pela utilização de dois reservatórios simplesmente pela capacidade de ampliação de funcionalidades da unidade e de possível interação com outras unidades didáticas. Seguindo esta lógica, a melhor dinâmica foi estabelecida ao colocar o *reservatório 1* e o *tanque de controle* com bases em mesmo nível, visto que a ação da gravidade seria capaz de esvaziar por completo o *tanque de controle*. As distâncias e medidas relativas das tábuas que compõem a *estrutura* podem ser encontradas no Apêndice A, enquanto a posição relativa entre os *reservatórios*, *tanque de controle* e *estrutura* são apresentadas no Apêndice B.

Considera-se que a unidade será instalada em cima de uma mesa, logo, o *painel elétrico* foi instalado na primeira tábua de suporte. Este, é feito da mesma madeira da *estrutura* e atua na proteção da parte elétrica do sistema contra acidentes envolvendo o fluido utilizado. Em seu interior é possível encontrar a fonte, o microcontrolador, o driver de motor, uma protoboard, um conversor DC-DC, e as fiações dos botões.

Os *reservatórios 1 e 2* têm a função de armazenar o fluido utilizado no controle quando o sistema está em repouso ou desligado. Ambos são do mesmo modelo, onde o material de fabricação é o polietileno e possuem um volume de 20 litros cada. O primeiro é indispensável para o funcionamento da unidade. O segundo reservatório pode ser substituído por outro componente, como um reator, ou até mesmo se tornar o ponto de conexão com outras unidades. Este fato, é um dos motivos que promovem a versatilidade deste protótipo. Para o caso de utilização da unidade com o *reservatório 1 e 2* é necessário que ambos estejam preenchidos com o volume mínimo de segurança de 15 e 10 litros respectivamente, dessa forma, evita-se que as bombas operem a seco.

Durante o funcionamento da unidade, a *bomba 1* envia água para a parte superior do *tanque de controle*. Logo em sua saída há uma bifurcação que leva a dois caminhos diferentes, sendo um mais curto com 111 cm e outro mais longo com 215 cm de comprimento, onde o

usuário pode optar por qual destes o fluido pode passar pelo acionamento de duas válvulas solenóides. Aqui a intenção é simular o tempo morto.

O *tanque de controle* é o principal elemento deste sistema, onde ocorre o controle de nível. É construído com um cano de PVC com 10 cm de diâmetro e 78 cm de altura, sendo aberto na parte superior e fechado com um cap em sua base, possuindo capacidade total de 6,1 litros. Na parte frontal do *tanque de controle* foi instalado um visor de nível simples, feito com conexões de ½ polegada e mangueira flexível transparente, para rápida visualização do nível. Optou-se pela alimentação na parte superior do tanque, para que a vazão de entrada não tenha relação de pressão com o *reservatório 1*, e para evitar a presença de bolhas de ar e turbulência, esta foi colocada próxima a parede do *tanque de controle*, dessa forma, reduzindo a interferência na medição do nível e consequentemente na execução do controle. A saída do fluido em direção do *reservatório 2* ocorre por ação da gravidade, por uma abertura no cap instalado, dessa forma o ponto zero do *tanque de controle* se dá a 1,0 cm acima da base. O *tanque de controle* é instalado em cima de um suporte quadrado de madeira onde se encontram os quatro sensores de peso.

Por fim, deve-se relatar que todos os recipientes possuem uma abertura, com o intuito de que as pressões internas dos mesmos sejam igualadas à pressão atmosférica e não interfira no funcionamento das bombas e do controle.

3.2 Projeto em Hardware

O projeto em hardware é subdividido em duas partes principais. A *Seção 3.2.1 Componentes do Sistema* descreve separadamente cada componente elétrico presente na unidade e detalha seu funcionamento. A *Seção 3.2.2 Painel elétrico* apresenta a interação elétrica (eletro-eletrônica) que ocorre entre os componentes citados anteriormente.

3.2.1 Componentes do Sistema

- **Microcontrolador**

A escolha do microcontrolador foi baseada nos seguintes fatores: custo, facilidade de utilização, tamanho da memória e quantidade de portas digitais e analógicas. Para atender os dois primeiros requisitos, a família Arduino de microcontroladores foi selecionada, visto que são facilmente encontrados no mercado a um preço acessível, possuem uma IDE própria simplificada e existem inúmeras fontes de informação como sites, fóruns e livros, sobre sua

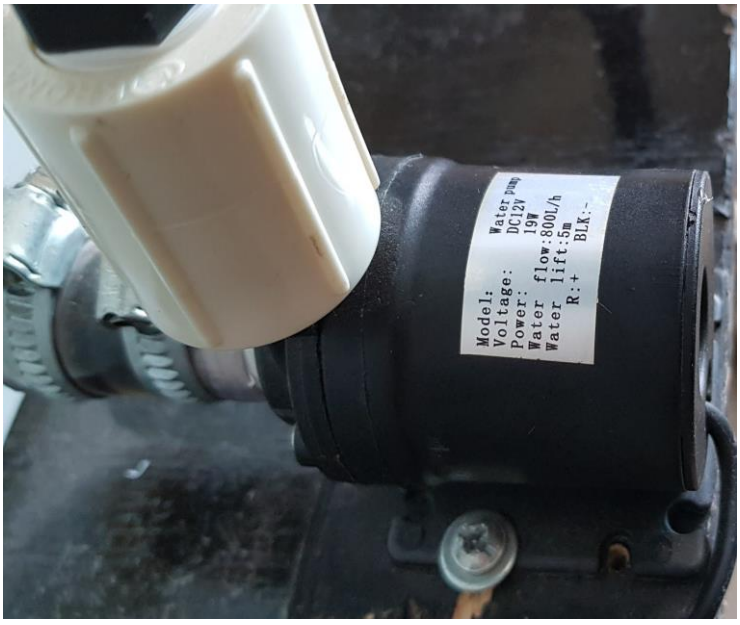
utilização, além de algoritmos propostos pela comunidade. Dentro desta família, o modelo Arduino MEGA 2560 se destaca por possuir 256KB de memória flash, 8KB de memória dinâmica, 54 portas digitais e 16 portas analógicas.

- **Bomba centrífuga**

Segundo Júnior (1991) uma bomba centrífuga é caracterizada por possuir um rotor dotado de pás e um difusor, sendo que o primeiro transfere a *energia mecânica (EM)*, provinda do motor, para o líquido em forma de *energia cinética (EC)*, e o segundo transforma a *EC* em *energia de pressão (EP)*.

A bomba escolhida para o projeto, é de fabricação genérica e não possui marca específica (Figura 3.2). De acordo com o fabricante, esta pode bombear líquidos não corrosivos a uma temperatura máxima de 80ºC. Dentre os detalhes técnicos encontra-se o fato desta ser uma bomba centrífuga operando com corrente contínua a 12 *volts (Vdc)*, seu motor não possui escova (*brushless*), é capaz de elevar o fluido a 5 metros de altura e trabalhar a uma vazão máxima de 800 L/h ou 13,3 L/min. Como a unidade didática possui um volume total de 46,1 litros, a vazão da bomba foi um dos principais motivos de sua escolha.

Figura 3.2 – Bomba centrífuga utilizada.



Fonte: Elaborado pelo autor.

Seu principal objetivo no sistema é movimentar o fluido, seja do *reservatório 2* para o *reservatório 1* (*Bomba 2*) , ou do *reservatório 1* para o *tanque de controle* (*Bomba 1*), sendo que a *Bomba 1* é o elemento *atuador* do loop de controle, que modifica o estado do sistema até

que o equilíbrio seja atingido no valor de *set-point* estabelecido pelo usuário. Para auxiliar neste objetivo e ser capaz de atuar no controle foi necessário utilizar uma bomba que opere com corrente contínua, possibilitando a regulagem da vazão pela variação da voltagem aplicada.

Durante testes de funcionamento constatou-se que cada bomba operando continuamente à 12Vdc consome entre 1,6 e 1,7 *amperes (A)* de corrente.

- **Driver de motor**

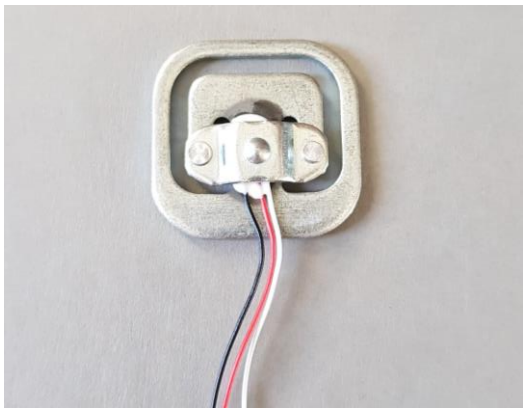
O microcontrolador escolhido não suporta correntes acima de 40 mA por entrada, o que impossibilita a ligação direta das bombas no próprio arduino. Dessa forma, fez-se necessário a implementação de um módulo *driver de motor*, visto que este seria capaz de usar os sinais emitidos pelo arduino e a energia de uma fonte externa para controlar os motores das bombas.

O módulo *Ponte H L298N*, é capaz de controlar até 2 motores DC de até 2A cada, permitindo variar o sentido de rotação e sua velocidade de rotação por meio da técnica de *modulação por largura de pulso - MLP* (em inglês, *pulse width modulation - PWM)*. Por se tratar de um módulo com tecnologia antiga, existe uma grande perda de tensão em sua saída, logo a tensão de entrada deve ser mais alta para compensar a baixa eficiência.

- **Sensor de peso**

Com o intuito de obter o peso do *tanque de controle*, e posteriormente o nível de líquido em seu interior, é necessário utilizar um instrumento capaz de medir a pressão exercida em um ponto. Dentre os sensores elétricos de pressão, o escolhido foi a *célula de carga* (*load cell*) de 50 Kg (Figura 3.3), que varia sua resistência elétrica em função da deformação exercida.

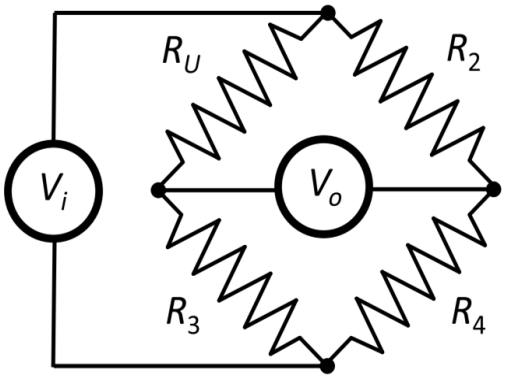
Figura 3.3 – Célula de carga de 50 Kg.



Fonte: Elaborado pelo autor.

É possível medir a voltagem resultante da deformação da *célula de carga* utilizando uma *ponte de wheatstone* (Figura 3.4), que consiste em um circuito elétrico com dois resistores em série (R_U e R_2) em paralelo com dois resistores em série (R_3 e R_4). As resistências R_2, R_3 e R_4 são conhecidas e R_U é a *célula de carga*, com resistência variável. Dessa forma, ao aplicar uma voltagem V_i, é possível obter uma voltagem diferente V_o, no interior da ponte.

Figura 3.4 – Circuito elétrico de uma ponte de wheatstone.

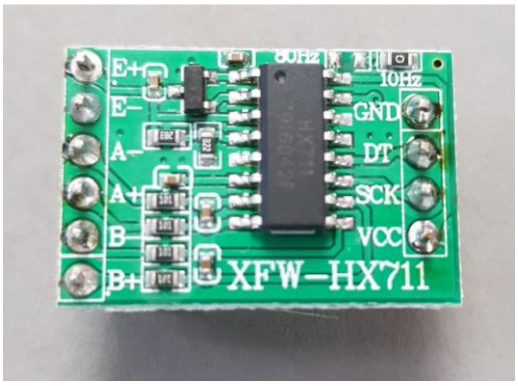


Fonte: Patience (2018).

Com o intuito de criar uma balança estável e nivelada com o plano da *estrutura* do protótipo, optou-se por utilizar 4 *células de carga* na *ponte de wheatstone*, em conjunto a uma placa de madeira para suportar o *tanque de controle*.

A comunicação entre a ponte e o arduino é feita a partir de um *conversor analógico-digital (AD)*, sendo assim, capaz de converter a entrada analógica de tensão variável em valores binários. O *conversor AD* escolhido foi o HX711 (Figura 3.5), esta decisão se deve ao fato deste módulo ser projetado para operar com *células de carga*, resolução de 24 bits e possuir um amplificador de sinal embutido.

Figura 3.5 – Conversor amplificador analógico-digital HX711.



Fonte: Elaborado pelo autor.

A calibração da balança pode ser feita com o auxílio do *algoritmo de calibração da balança*, disponível no Apêndice C, que é uma adaptação do algoritmo disponível no site Curtocircuito. Para sua utilização faz-se necessário o uso da biblioteca HX711.h do autor Bodge, publicado no fórum GitHub. Com o auxílio de um objeto com peso conhecido, a balança pode ser calibrada, variando o valor da variável *fator_balanca*, até que o valor de peso medido seja igual ao valor do objeto usado como referência.

Após calibração, o melhor valor encontrado para o *fator_balanca* foi de -23270,40, resultando em um erro máximo de 1,25% e médio de 0,88%. Percebeu-se que o erro tende a aumentar com o aumento do peso medido, no entanto, para o range de 0 a 4,5 kg o *sensor de peso* apresentou boa *acurácia* e *precisão*. Ademais, percebeu-se que a posição de um objeto na balança não influenciou no peso medido.

- **Sensor de fluxo**

O *sensor de fluxo YF-S201* (Figura 3.6) é utilizado como sensor de monitoramento de vazão. Seu objetivo é medir constantemente o comportamento e variações no fluxo de fluido na entrada e saída do *tanque de controle*, assim, sendo necessário dois sensores, um em cada posição citada. Dessa forma, faz-se possível analisar e registrar as mudanças nas variáveis *vazão de entrada* e *vazão de saída* de acordo com as alterações na variável *nível* provindas do controle PID.

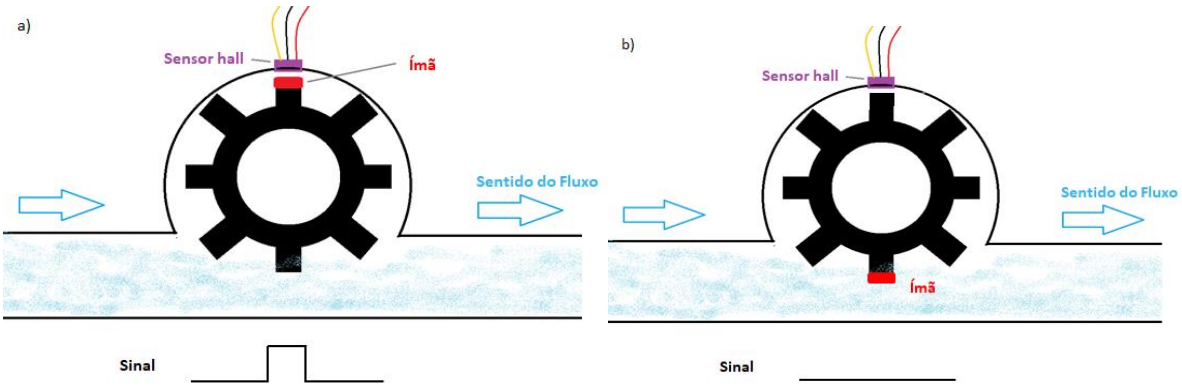
Figura 3.6 – Sensor de fluxo YF-S201.

Fonte: Elaborado pelo autor.

Este sensor é composto por um corpo plástico com conexões roscadas de ½ polegada, um impelidor acoplado a um rotor, um ímã em uma das pás e um *sensor hall*. Seu princípio de funcionamento é baseado no *efeito hall*, ou seja, é capaz de detectar variações no campo

magnético. Ao escoar um líquido no interior do *sensor de fluxo,* o impelidor entra em rotação, movimento este que irá aproximar e afastar o ímã do *sensor hall* e será convertido em pulsos. A Figura a seguir explicita o funcionamento do *sensor de fluxo*:

Figura 3.7 – Ímã próximo (a) e afastado (b) do sensor hall.



Fonte: Adaptado de TheoryCircuit (2018).

Para determinar a vazão volumétrica passando pelo sensor utilizou-se a Equação 3.1 que relaciona pulsos por unidade de tempo e vazão:

$$Vazão = \frac{Pulsos}{2 \times constante\ do\ equipamento \times \frac{Variação\ no\ tempo}{1000}} \qquad (3.1)$$

Onde, a divisão por 2 é necessária visto que o equipamento detecta a mudança do sinal baixo para o alto e também do sinal alto para o baixo, e o interesse é detectar este par como um único movimento. A divisão por 1000 vem da conversão de milissegundos (ms) para segundos (s). A variável *constante do equipamento* possui unidade [Pulsos/s. L/min] e foi obtida por meio da calibração do equipamento.

A determinação das constantes conta com o auxílio do *algoritmo de calibração do fluxímetro,* disponível no Apêndice D. Inicialmente, aplica-se uma vazão constante no interior do *sensor de fluxo,* obtendo-se a leitura deste pelo algoritmo. O valor da constante deve ser alterada até que o valor da leitura seja suficientemente próximo do valor real.

Após calibração, a constante obtida para o fluxímetro na entrada do *tanque de controle* foi de 7,0 Pulsos/s. L/min com erro máximo de 2%, já no que se encontra na saída do tanque, foi de 7,5 Pulsos/s. L/min com erro máximo de 4%.

- **Buzzer Ativo e Sonda de Nível**

Na tentativa de evitar acidentes no manuseio da unidade, foi implementado no *sistema de controle* um aviso sonoro no caso de o *reservatório 2* atingir o *nível crítico*, que é próximo

do transbordamento. Para tanto, foi desenvolvida uma sonda de nível que consiste em duas agulhas de 4 cm de comprimento separadas por 1 mm e é introduzida na lateral do *reservatório 2* a 30 cm da base, totalizando 20,5 litros. O microcontrolador alimenta a sonda com 5Vdc para quando a água atingir o nível crítico e a submergir o circuito feche, tornando possível a percepção pelo microcontrolador. Após a detecção, este envia um sinal ao buzzer ativo que produzirá o efeito sonoro alertando que o usuário deve, manualmente, ativar a *bomba 2*.

3.2.2 Painel elétrico

Para fixar os componentes citados no tópico anterior, assim como, os botões de acionamento, foi utilizada uma caixa de madeira. Esta foi feita através de um trabalho artesanal, onde as partes superior e frontal, foram mantidas abertas intencionalmente para manter a circulação de ar e, como a proposta da unidade é ser multifuncional, possibilitar alterações em seu interior.

As dimensões da caixa manufaturada são 23 cm de profundidade, 18 cm de largura e 31 cm de altura. Na tampa lateral direita estão os botões de comunicação entre usuário e máquina.

Figura 3.8 – Caixa de madeira onde estão instalados os componentes elétricos.

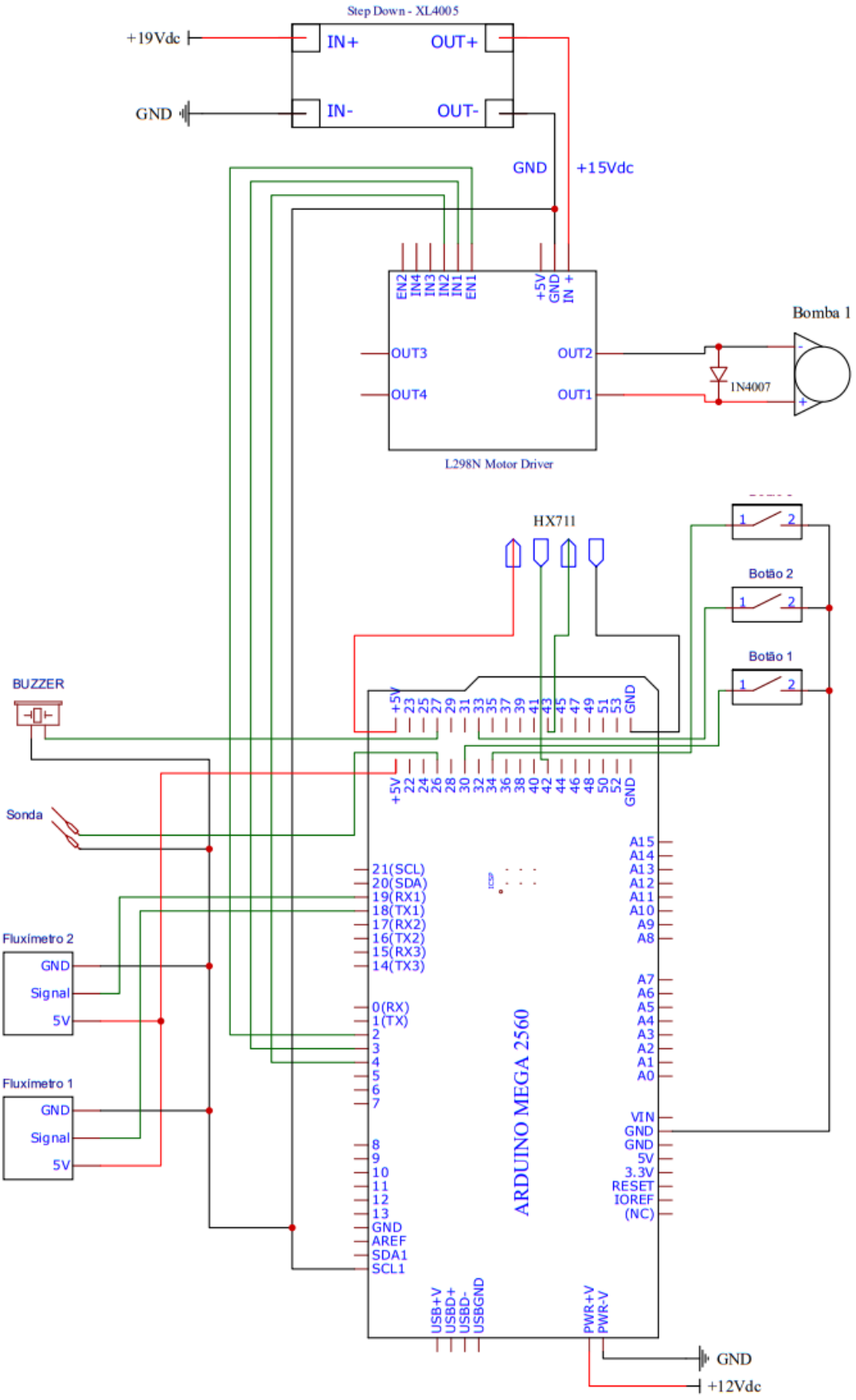
Fonte: Elaborado pelo autor.

Com o auxílio do software online gratuito EasyEDA foi possível desenvolver o circuito elétrico utilizado na unidade. Para facilitar o entendimento, é possível separá-lo em 4 partes: *circuito do Microcontrolador, circuito da balança, circuito externo da Bomba 2,* e *circuito externo das válvulas solenóides.*

A parte *circuito do Microcontrolador* é a mais complexa visto que envolve vários componentes indispensáveis no funcionamento da unidade, como: *Microcontrolador, Driver de Motor, Fluxímetros, Sonda de Nível, Buzzer Ativo* e botões de navegação. O primeiro é alimentado com uma tensão de 12Vdc, o segundo com 16Vdc partindo de um fonte de 19Vdc e utilizando um módulo *Step-Down* para regular a tensão de saída, e os demais operam a 5Vdc. Foi utilizada uma protoboard como barramento para facilitar as conexões +5Vdc e GND. Do microcontrolador são utilizadas 3 *entradas digitais* para os botões, 2 *entradas digitais* para o *Conversor AD,* 1 *pino de interrupção externa* para cada fluxímetro, 1 *entrada digital* para a sonda e 1 *saída digital* para o buzzer ativo. Finalmente, para a comunicação com o driver de motor são utilizadas 2 *saídas digitais*, para o controle de sentido e freio do motor, e *1 saída PWM,* conforme demonstrado na Figura 3.9.

Para cada pino que está configurado como entrada existe um resistor do tipo *pull-up* de 10KΩ, internos do arduino, para evitar um valor de entrada flutuante enquanto o pino estiver desconectado. Soma-se a isso a utilização da biblioteca debounce2.h do autor Thomas Fredericks, publicado no fórum GitHub, para contornar o efeito de *bouncing* quando os botões são acionados. Ademais, na alimentação das *bombas* e válvulas solenóides foram instalados diodos do tipo IN4007, para proteger o circuito contra as correntes reversas; este é ligado em paralelo.

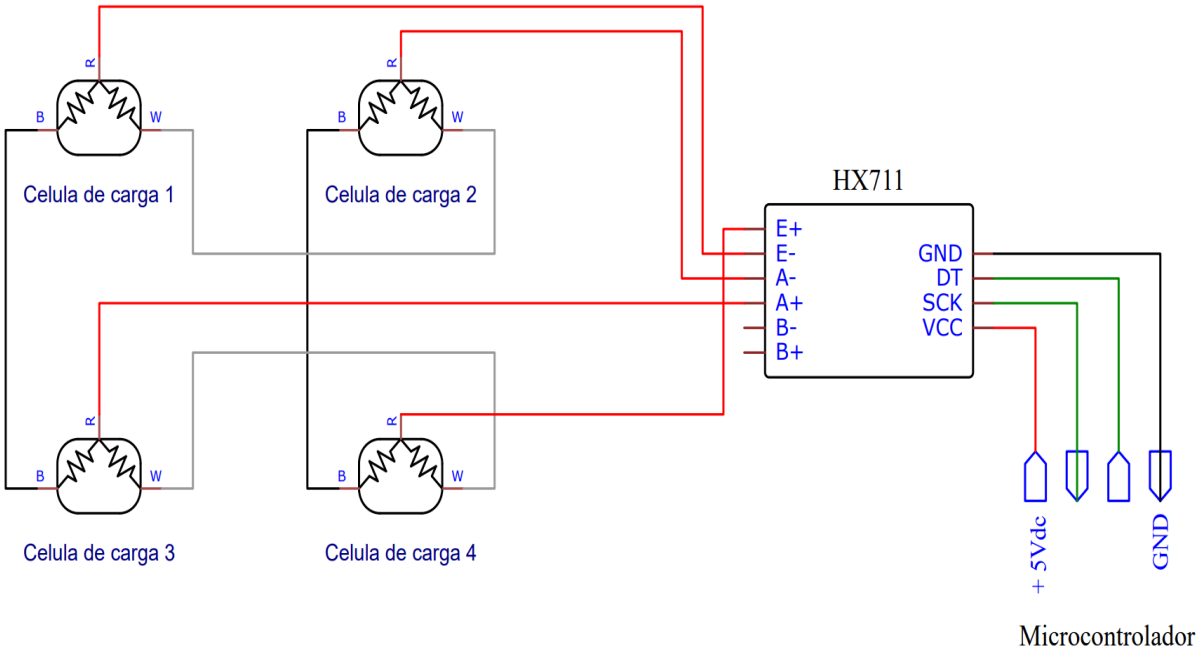
Figura 3.9 – Circuito do Microcontrolador.



Fonte: Elaborado pelo autor.

O conjunto *circuito da balança* conta com o *Conversor A/D* e as *Células de Carga.* O módulo HX711 é alimentado pelo próprio *microcontrolador* a 5Vdc, sendo necessária 1 *entrada digital* e 1 *saída digital* para seu funcionamento, e este faz a ligação com as células. A Figura 3.10 demonstra as ligações feitas entre as partes.

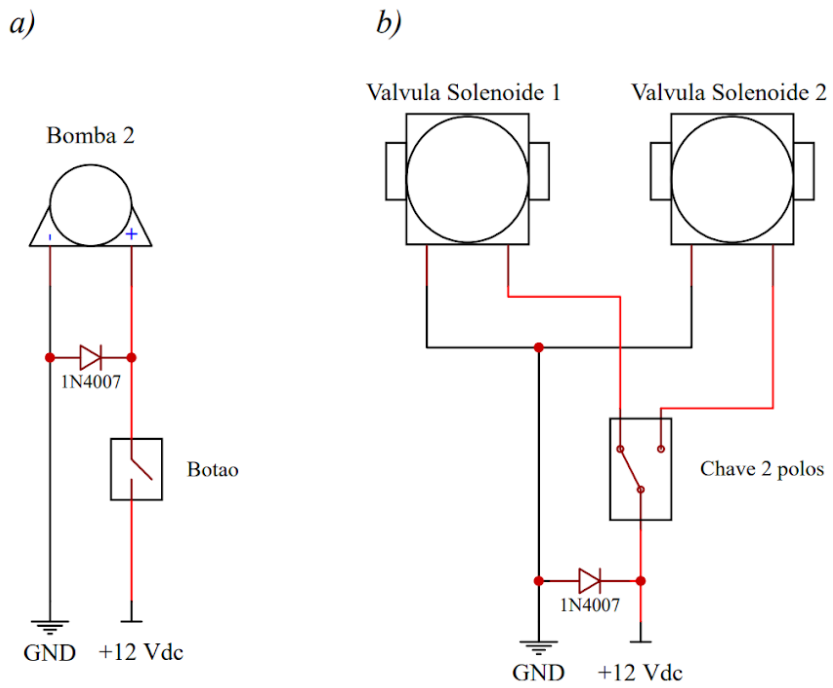
Figura 3.10 – Circuito da balança.



Fonte: Elaborado pelo autor.

Os dois outros circuitos encontrados na unidade operam de maneira externa ao *Sistema de Controle*, para que estes não interfiram em seu funcionamento. O *circuito externo da Bomba 2* possui uma fonte de 12Vdc onde o GND está conectado diretamente na *Bomba 2* e o +12Vdc está conectado a uma chave de 2 polos, que ao ser pressionada, permite o caminho positivo até a bomba. O *circuito externo das válvulas solenóides* possui uma fonte 12Vdc cujo GND é compartilhado e conectado diretamente nas duas válvulas e o +12Vdc está conectado ao pólo central de um interruptor alavanca tripolar, logo, qualquer posição da chave, A ou B, sempre garante que uma válvula estará fechada e outra estará aberta.

Figura 3.11 – Circuitos externos. a) Bomba 2. b) Válvulas solenoides.



Fonte: Elaborado pelo autor.

3.3 Projeto em Software

Este tópico visa explicitar o funcionamento do algoritmo e suas funções, assim como, demonstrar a interação entre este e os instrumentos utilizados na unidade. Para tanto, será descrita cada uma das funções internas do código fonte.

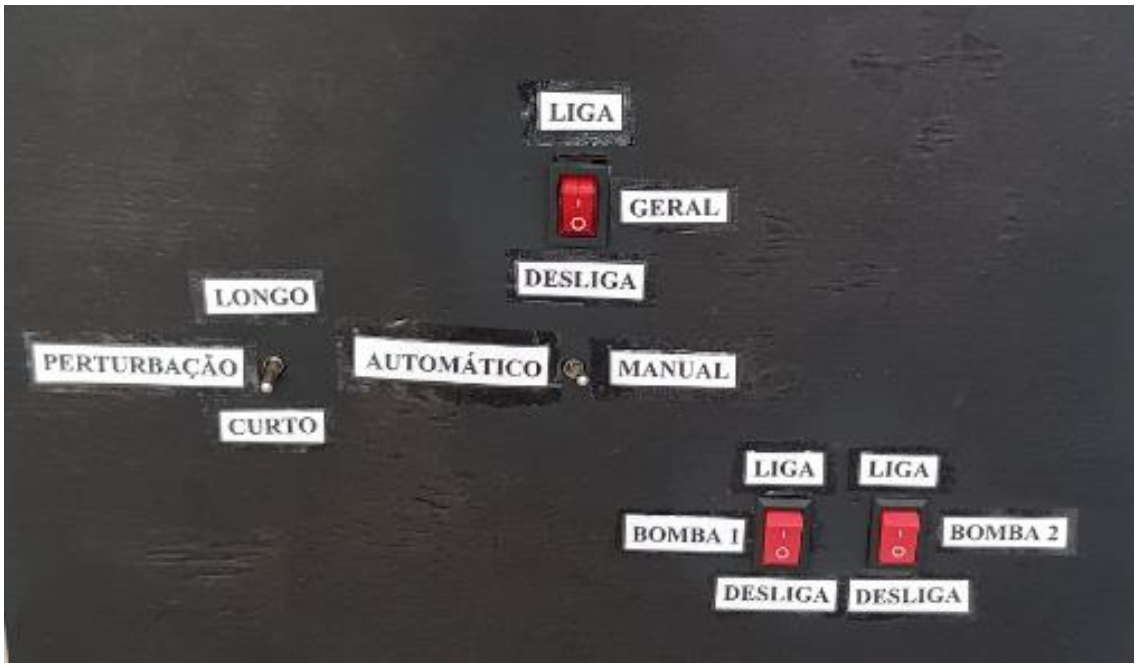
3.3.1 Painel de Controle

A utilização do painel de controle é muito simples e intuitiva. A posição dos botões e seus títulos foram propostos de modo que o usuário possa seguir de maneira linear suas possíveis escolhas.

O painel elétrico possui em sua lateral 5 botões, divididos em 2 grupos, responsáveis pela interface com o usuário, todas as decisões são tomadas através deles, cada qual com um finalidade única, conforme demonstrado na Figura 3.12. O primeiro grupo, com 4 botões, representa o controle do sistema. Seu primeiro botão é chamado *LD* e tem a finalidade de ligar ou desligar todas as funções do algoritmo, podendo ser utilizado como desligamento de emergência. O segundo é chamado *Automático/Manual* e tem como fim permitir que o usuário escolha a operação das bombas entre o modo automático PID, quando a chave está para a esquerda, e o modo de acionamento manual, quando a chave aponta para a direita. Os botões 3

e 4 são derivados do modo manual, onde o usuário opta por ligar ou desligar a *bomba 1* e *bomba 2*, respectivamente. O segundo grupo possui apenas 1 botão chamado *Alternar* que pode ser interpretado como um gerador de distúrbio no sistema, quando a chave está apontada para cima o fluido percorre um caminho com maior perda de carga, caso contrário, percorre um caminho com menos perda de carga.

Figura 3.12 – Painel elétrico.



Fonte: Elaborado pelo autor.

Parte da operação da unidade didática se dá pelo uso de um computador, isto inclui a definição das variáveis, que pode ser realizada através de alterações no código fonte, e a apresentação das variáveis medidas, através de comunicação serial com formato CSV (valores separados por vírgula). A seção “Inputs do Usuário” é a única parte que deve ser manipulada no algoritmo, sendo de responsabilidade do usuário se atentar às regras descritas a seguir. É possível alterar os três parâmetros do controle PID, *Kp, Ti e Td*, que recebem números com até 1 casa decimal, ademais o *Setpoint* também pode ser alterado, com a mesma regra.

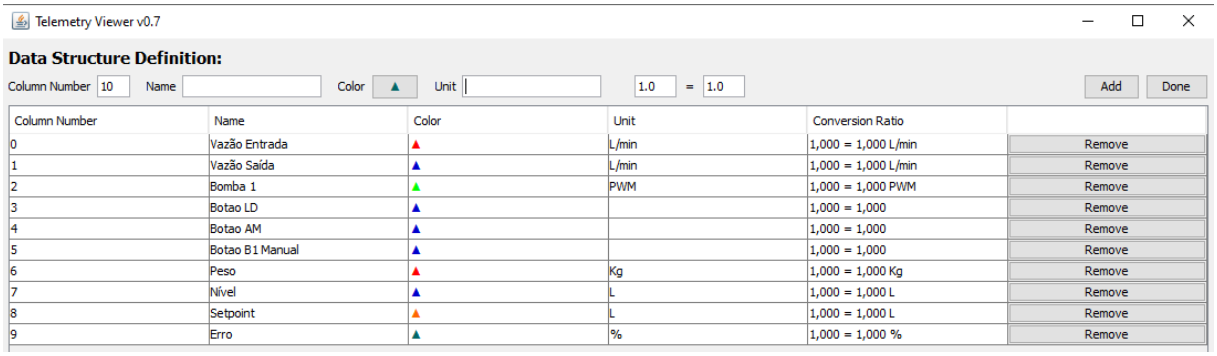
Para facilitar a experiência do usuário é sugerido o uso do software gratuito TelemetryViewer v0.7 do autor Farrell Farahbod (Figura 3.13), que é capaz de plotar em tempo real os dados CSV obtidos por comunicação serial.

Figura 3.13 – Software TelemetryViewer v0.7.

Fonte: Elaborado pelo autor.

No primeiro momento é necessário conectar o programa com a porta USB (UART:COM) que está conectada ao microcontrolador e determinar a velocidade de transmissão (*baud rate*), em seguida o software detecta quantos valores estão separados por vírgula e pede ao usuário que seja determinado seu nome e unidade de medida (Figura 3.14). É possível alterar a subdivisão da área de trabalho e selecionar determinados tipos de gráficos disponíveis para escolha.

Figura 3.14 – Inicialização dos dados CSV.



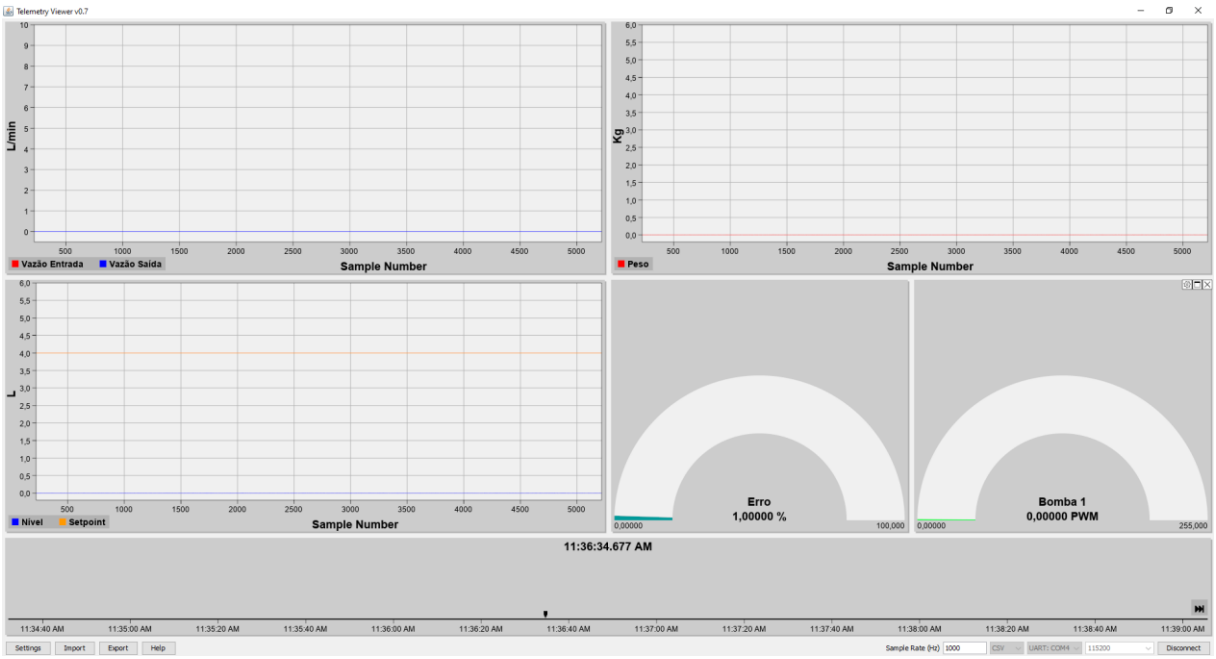
Column Number	Name	Color	Unit	Conversion Ratio	
0	Vazão Entrada	▲	L/min	1,000 = 1,000 L/min	Remove
1	Vazão Saída	▲	L/min	1,000 = 1,000 L/min	Remove
2	Bomba 1	▲	PWM	1,000 = 1,000 PWM	Remove
3	Botao LD	▲		1,000 = 1,000	Remove
4	Botao AM	▲		1,000 = 1,000	Remove
5	Botao B1 Manual	▲		1,000 = 1,000	Remove
6	Peso	▲	Kg	1,000 = 1,000 Kg	Remove
7	Nível	▲	L	1,000 = 1,000 L	Remove
8	Setpoint	▲	L	1,000 = 1,000 L	Remove
9	Erro	▲	%	1,000 = 1,000 %	Remove

Fonte: Elaborado pelo autor.

Para facilitar o uso do programa um arquivo de configuração dos gráficos pode ser encontrado no Apêndice D que deve ser salvo como arquivo de texto (extensão .txt), este utiliza 4 gráficos do tipo *Time Domain* onde o eixo x representa o número da amostra e o eixo y a

unidade de medida da variável selecionada, 1 gráfico do tipo *Timeline* que apresenta a data e hora atual, e 2 gráficos do tipo *Dial* para indicar o *erro* e o *PWM* da *bomba 1*.

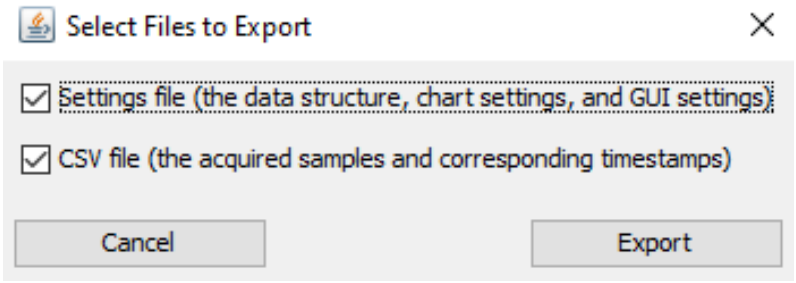
Figura 3.15 – Layout gráfico proposto.



Fonte: Elaborado pelo autor.

A utilização do TelemetryViewer v0.7 também permite a reprodução de testes e experimentos previamente gravados, e a exportação dos dados CSV para fácil comunicação com outros softwares, como indica a figura a seguir:

Figura 3.16 – Tipos de exportação de dados.



Fonte: Elaborado pelo autor.

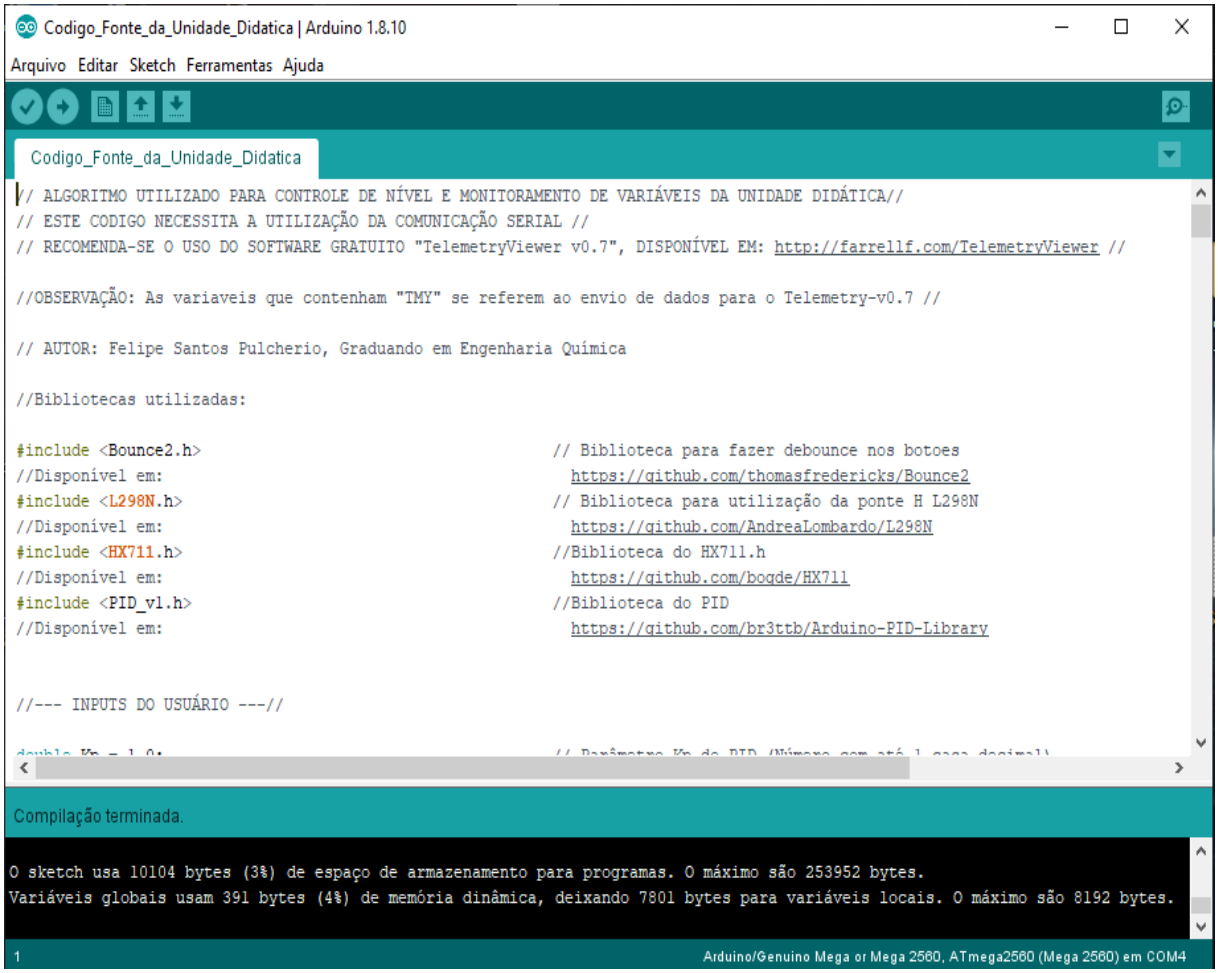
3.3.2 Código Fonte da Unidade

O algoritmo do projeto foi estruturado em uma rotina *loop* a qual é executada novamente após sua finalização. Esta tem como objetivo chamar e executar pequenas rotinas, chamadas de *funções*. Dividir a execução do programa em partes menores garante maior clareza, pois cada sub rotina é responsável por uma etapa específica do funcionamento do programa, e

flexibilidade, permitindo que alterações feitas no interior de um *função* não interfira no funcionamento das outras.

Para desenvolver o código fonte foi utilizado o ambiente de desenvolvimento integrado (IDE) disponibilizado gratuitamente no site oficial do Arduino, apresentado na Figura 3.17. Este tem como vantagem a possibilidade do usuário editar, compilar e depurar, em um único software, seu programa em *linguagem C++*.

Figura 3.17 – IDE Arduino.



Fonte: Elaborado pelo autor.

Durante a criação do programa, o processo de compilar o código e testar a unidade era constante, principalmente para encontrar erros e aferir as subrotinas do projeto, como por exemplo: a calibração dos fluxímetros e balança, teste de funcionamento dos botões, sonda de nível, etc.

O código fonte foi dividido em grandes blocos para manter a linearidade de sua leitura e interpretação. A primeira parte chamada *Inputs do Usuário* é a única que possui variáveis

globais passíveis de alteração pelo usuário. Em seguida, têm-se a *Inicialização de Variáveis* que, como o nome indica, são definidas todas as variáveis globais, determinados os pinos que serão utilizados, e criados *objetos*. A terceira etapa, *Inicialização do Sistema*, é responsável por iniciar a comunicação serial e interrupções, indicar o tipo dos pinos (*Entrada* ou *Saída*), manter a *Bomba 1* no estado parado, tarar a balança, iniciar o PID no modo manual (Desligado) e impor limites no seu range de saída. O modelo de Arduino selecionado possui excursão de *PWM* em 8 bits ($2^8 = 256$), garantindo uma variação de 12Vdc/256 = 0,046875 Vdc/unidade, no entanto, para não danificar a bomba o range do PID está entre 5Vdc e 12Vdc ou 107 e 255 unidades.

Serão descritas a seguir todas as funções que são chamadas pela rotina principal *loop*, assim como o seu funcionamento e finalidade. O algoritmo completo pode ser encontrado no Apêndice E.

- ***CalculoVazao( ):*** Esta é a primeira função que é chamada pelo código fonte. Tem como objetivo coletar amostras de variação de pulso dos *fluxímetros 1* e *2*, executar o cálculo que determina a vazão volumétrica atual e imprimir esses valores. Durante os cálculos os pinos interrompíveis são momentaneamente desligados. Ao utilizar o pino interrompível é necessário utilizar uma *função* extra que será chamada a cada variação de pulso, esta foi chamada de *ContaPulso1( )* e *ContaPulso2( )*, sendo uma para cada *fluxímetro*. Com o intuito de manter a precisão dos cálculos elevada, o período de coleta de amostras selecionado foi 1 segundo, enquanto este tempo não for atingido a *função* imprime o último valor calculado.
- ***ContaPulso1( ) e ContaPulso2( ):*** Chamada diretamente pelos pinos interrompíveis, esta *função* tem como único objetivo armazenar quantos pulsos foram realizados entre o período de 1 segundo, após os cálculos da vazão seu valor é zerado.
- ***QualBotao(byte pBotoesClicados[ ]):*** Esta é capaz de perceber as mudanças entre os botões acionados, utilizando uma rede de *if* e *else*, e acionar uma resposta pertinente a cada situação. Aqui é dada a permissão para ligar a unidade ou desligá-la de modo emergencial, alternar entre o modo automático e manual, assim como, determinar a velocidade inicial da *bomba 1* nestas situações, podendo ser 0 ou caso esteja no modo manual e desligada, ao ser acionada chama-se outra *função* para iniciá-la com softstart. Ao final de sua execução imprime a velocidade atual da *bomba 1* e o estado dos botões.
- ***SoftStart(byte a):*** Quando a velocidade da *Bomba 1* é zero e o usuário deseja acioná-la esta *função* é chamada para dar início ao softstart da mesma. Após atingir a velocidade

máxima que é 255, a *função* é encerrada e retorna 0 indicando a não necessidade de utilizar novamente.

- ***QualNivel( ):*** Serve para periodicamente coletar os valores de peso medidos na balança e convertê-los em nível com o auxílio de uma equação de conversão, que será apresentada com mais detalhes na *Seção 4 Resultados e Discussões.* Ademais, calcula o *Erro* que é igual ao *Nivel* menos o *Setpoint*, dividido pelo *Setpoint*, este representa o quão distante o *Nivel* está do *Setpoint*. Ao final, imprime os valores do *Peso*, *Nivel*, *Setpoint* e *erro*.
- ***AvisoEmergencia( ):*** Quando a sonda de nível detecta a presença de água, esta *função* altera os valores do *pinoBuzzerAtivo* entre ligado e desligado, criando um sinal sonoro para alertar o usuário de que o *reservatório 2* está perto da região de transbordamento.
- ***Loop_PID( ):*** Rotina específica que executa todo o modo automático da unidade, sendo uma das *funções* mais importantes. Primeiramente, altera o valor do *objeto myPID* para automático e atualiza a variável *Input* com o valor mais recente do *Nivel*, em seguida o comando .*Compute( )* é chamado alterando o valor da variável *Output.* Verifica-se, em seguida, se o *Output* está realmente entre o limite previamente definido do PID que é entre 107 e 255, caso esteja menor que 107 ou maior que 255, a variável recebe o valor do limite. Para finalizar a execução, a *Bomba 1* recebe o valor de *Output*.

4. ANÁLISE DE RESULTADOS

O capítulo sobre análise de resultados contempla toda a integração do projeto, a fim de demonstrar a aplicabilidade da teoria e instrumentação trabalhadas na real execução do controle de nível PID na unidade didática. A *Seção 4.1* introduz o capítulo partindo de um *Balanço Dinâmico e Elaboração do Modelo*, sendo estes dados fundamentais para realizar a *Aplicação e Sintonização do Controlador PID* (*Seção 4.1.2)* no código fonte.

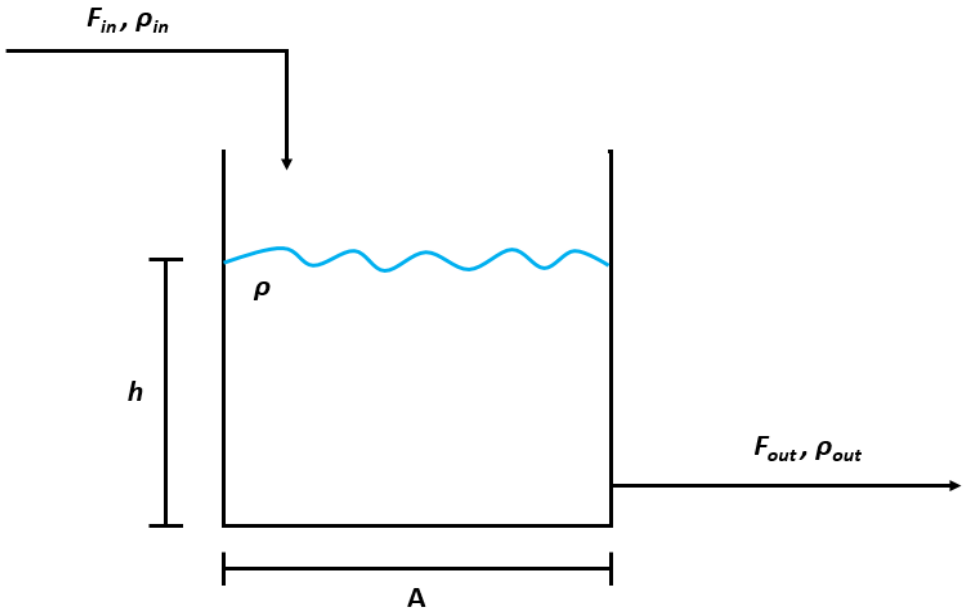
4.1 Balanço Dinâmico e Elaboração do Modelo

Para que seja possível realizar uma ótima *sintonização* do *controlador* do processo em questão, faz-se necessário entender o comportamento do *sistema*, seja no *estado estacionário (EE)* ou no *estado transiente (ET).* Isto pode ser alcançado partindo de uma análise das variáveis envolvidas no processo, assim como, estabelecendo hipóteses acerca de seu funcionamento.

Como hipóteses serão consideradas a ausência de reação química, presença somente do componente água, e que não há variação na temperatura entre o ponto de entrada e saída do

tanque de controle. Com base nisso a figura a seguir foi proposta para modelar o sistema e caracterizar as variáveis:

Figura 4.1 – Modelagem do tanque de controle.



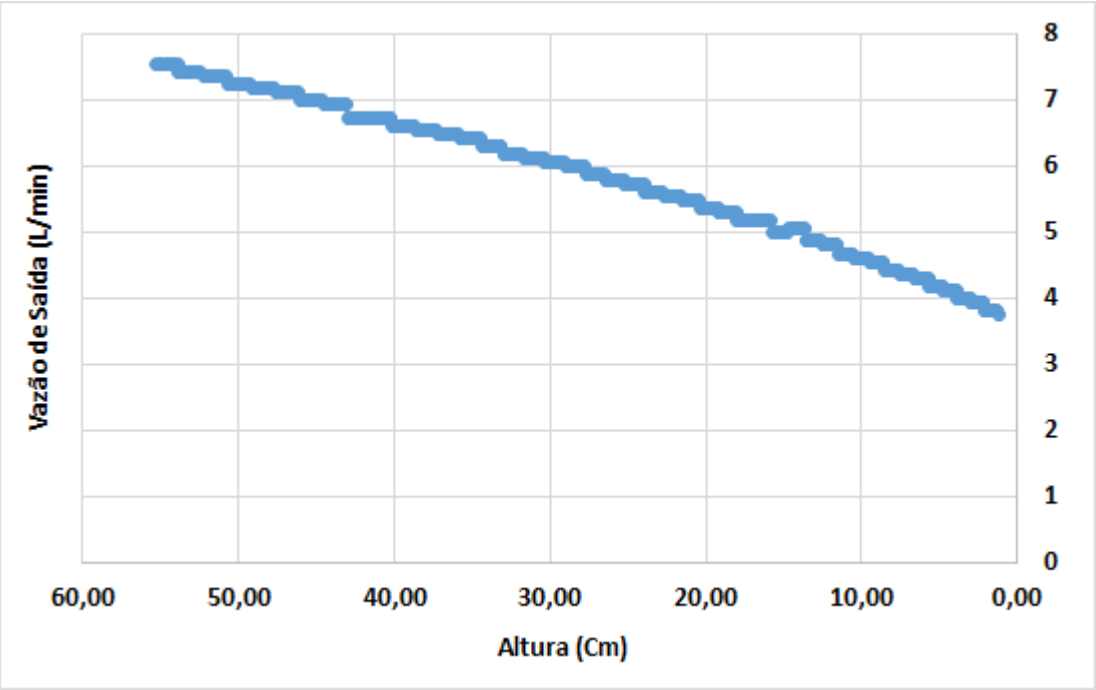
Fonte: Elaborado pelo autor.

Observando a Figura 4.1, e considerando *M* como o *massa do fluido* no tanque, F_{in} como a *vazão de entrada*, e F_{out} como a *vazão de saída*, é possível realizar o balanço de massa obtendo a Equação 4.1, que modela a *variação instantânea do peso* no *tanque de controle*.

$$\frac{dM}{dt} = F_{in} - F_{out} \tag{4.1}$$

Sabe-se que a vazão de saída F_{out} depende da *altura do fluido* no tanque. Para conhecer a natureza desta relação, foi plotado o gráfico entre essas variáveis, usando a válvula na saída em 60% aberto. O gráfico obtido, com o auxílio do software Microsoft Excel, é mostrado na Figura 4.2, sendo que os pontos obtidos foram gerados dentro do código fonte pela função *CalculoVazao( ).*

Figura 4.2 – Altura do fluido no tanque x Vazão de saída, válvula 60% aberta.



Fonte: Elaborado pelo autor.

Partindo dos 483 pontos amostrais obtidos, fez-se possível realizar uma regressão para encontrar uma função que representasse estes dados, a esta foi dado o nome Equação 4.2. Sua análise permite concluir que a *vazão de saída* F_{out} possui relação suficientemente linear com a *altura do fluido* no tanque.

$$F_{out} = \mathbf{0,8719}.h + 3{,}9231 \qquad (r^2 = 0{,}9916) \qquad (4.2)$$

Vale salientar que outras aberturas para a válvula foram testadas, porém apresentaram comportamento diferente do linear, o que aumentaria a complexidade do *sistema* a ser modelado. Deve-se interpretar a válvula como elemento de resistência fixo, cuja única função é limitar a vazão de saída para que esteja de acordo com a capacidade da bomba instalada na entrada.

A variável *h* apresentada na Equação 4.2 é resultado da substituição da Equação 4.4 na 4.3. Como dito anteriormente, a leitura realizada pelo *sensor de peso* está em kg e precisa ser convertida em centímetros.

$$M = V.\rho \qquad (4.3)$$

$$V = A.h \qquad (4.4)$$

Estas equações também podem ser usadas para alterar a Equação 4.1:

$$\frac{dh}{dt}\rho\, A \;=\; F_{in}\,.\,\rho_{in} \;-\; F_{out}\,.\,\rho_{out} \tag{4.5}$$

Considerando que o fluido usado é água, e não é esperado variação de temperatura do mesmo, pode-se assumir a conversão direta pela densidade de 1 Kg/L. Ademais, como o *tanque de controle* possui dimensões fixas a área de sua seção transversal é aproximadamente 78,54 cm².

Substituir a Equação 4.2 na Equação 4.5 permite obter o modelo teórico final do *tanque de controle* que é uma *EDO* de *1ª Ordem*, como visto no tópico de revisão.

$$\frac{dh}{dt} = \frac{F_{in} - (\mathbf{0{,}8719}.h + 3{,}9231)}{78{,}54} \tag{4.6}$$

Ao aplicar a transformada de Laplace na Equação 4.6, considerando sua condição inicial nula (y(0)=0), obtém-se a *função transferência teórica* do *tanque de controle*:

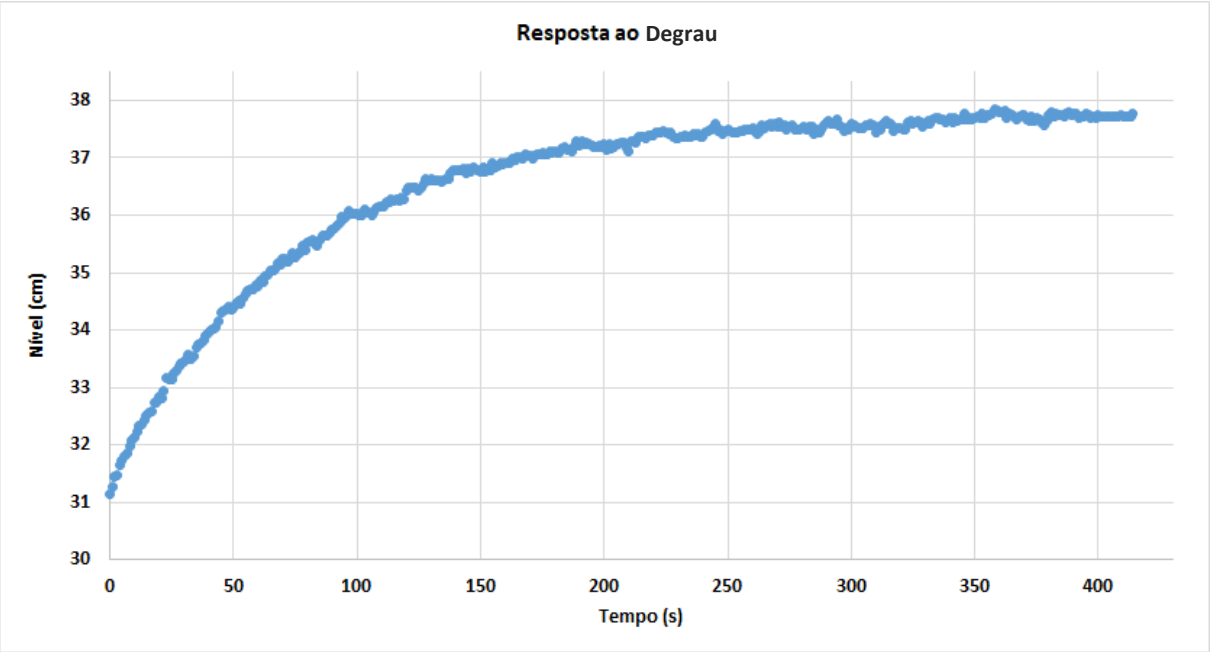
$$G_{tc}(s) = \frac{0{,}0127}{s + 0{,}0112} \tag{4.7}$$

4.2 Sintonização do Controlador PID

A obtenção dos parâmetros do controlador PID será realizada pelo procedimento *método da curva de reação,* logo, um experimento de *perturbação degrau* foi realizado com o *sistema* no modo *malha-aberta*. Foram realizados vários testes para encontrar a *voltagem* mínima de operação da *bomba* que fosse capaz de deixar o *nível* no *tanque de controle* estabilizado no *EE*. Em seguida, foi aplicado um degrau na entrada do *sistema*.

Inicialmente o valor do *pwm* era 200, o que representa uma *tensão* de saída no pino do driver de motor de 9,37Vdc, para esta configuração o *nível* no EE_1 estabilizou em 31,1 cm. O salto escolhido foi de 55 pwm, e assim que acionado, o nível no *tanque de controle* se altera, atingindo nova estabilidade EE_2 de 37,7 cm, após 389 segundos. Para facilitar a visualização da resposta ao salto, o software Microsoft Excel foi novamente usado para plotar um gráfico, como demonstrado na Figura 4.3, sendo que os pontos obtidos foram gerados dentro do código fonte pela função *QualNivel( )*.

Figura 4.3 – Curva de resposta a perturbação no sistema em malha-aberta.



Fonte: Elaborado pelo autor.

Como descrito na *Seção 2.3.4*, o interesse é a obtenção de um valor preliminar dos parâmetros *ganho proporcional K*, *tempo integral* T_i e *tempo derivativo* T_d. Pela Figura 4.4, os valores obtidos foram:

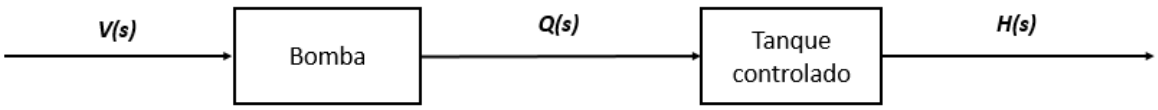
$$K = 1{,}2\frac{T}{L} = 1{,}2\frac{89}{0{,}5} = 213{,}6 \tag{4.9}$$

$$T_i = 2L = 2\ x\ 0{,}5 \ = 1 \tag{4.10}$$

$$T_d = \frac{L}{2} = \frac{0{,}5}{2} = 0{,}25 \tag{4.11}$$

A Equação 4.7 contempla somente o *tanque de controle*, no entanto o *sistema* deve ser modelado de forma a incluir a *bomba*. Esta será considerada uma constante, visto que o seu comportamento dinâmico é mais rápido que o *sistema* como um todo. Um novo diagrama de blocos é apresentado na Figura 4.4, para contemplar essas modificações.

Figura 4.4 – Diagrama de blocos do sistema real.



Fonte: Elaborado pelo autor.

A figura anterior demonstra que a *bomba* recebe o *sinal de tensão V(s)* e o converte em *vazão Q(s)*, enquanto o *tanque de controle* converte a vazão em *nível*. Logo a *função transferência do sistema* deve relacionar *volts* por *centímetros*. A representação matemática, aplicando a propriedade multiplicativa, está demonstrada a seguir na Equação 4.8.

$$G(s) = K_{bomba} \cdot G_i(s) = \frac{0{,}0127.K_{bomba}}{s + 0{,}0112} = \frac{K_{sistema}}{s + 0{,}0112} \quad (4.8)$$

Com a intenção de estabelecer o *ganho estático do sistema*, foi necessário encontrar o EE referente à operação da unidade no limite inferior do PID, ou seja, a bomba foi operada com pwm 107 (5Vdc) e o nível estabilizado foi registrado. Assim, a Equação 4.9 foi preenchida:

$$K_{sistema} = \frac{h_{max} - h_{min}}{V_{max} - V_{min}} = \frac{37{,}7 - 1{,}5}{12 - 5} = 5{,}17\,\frac{cm}{V} \quad (4.9)$$

Finalmente a *função transferência do sistema* pode ser dada conforme a Equação 4.10:

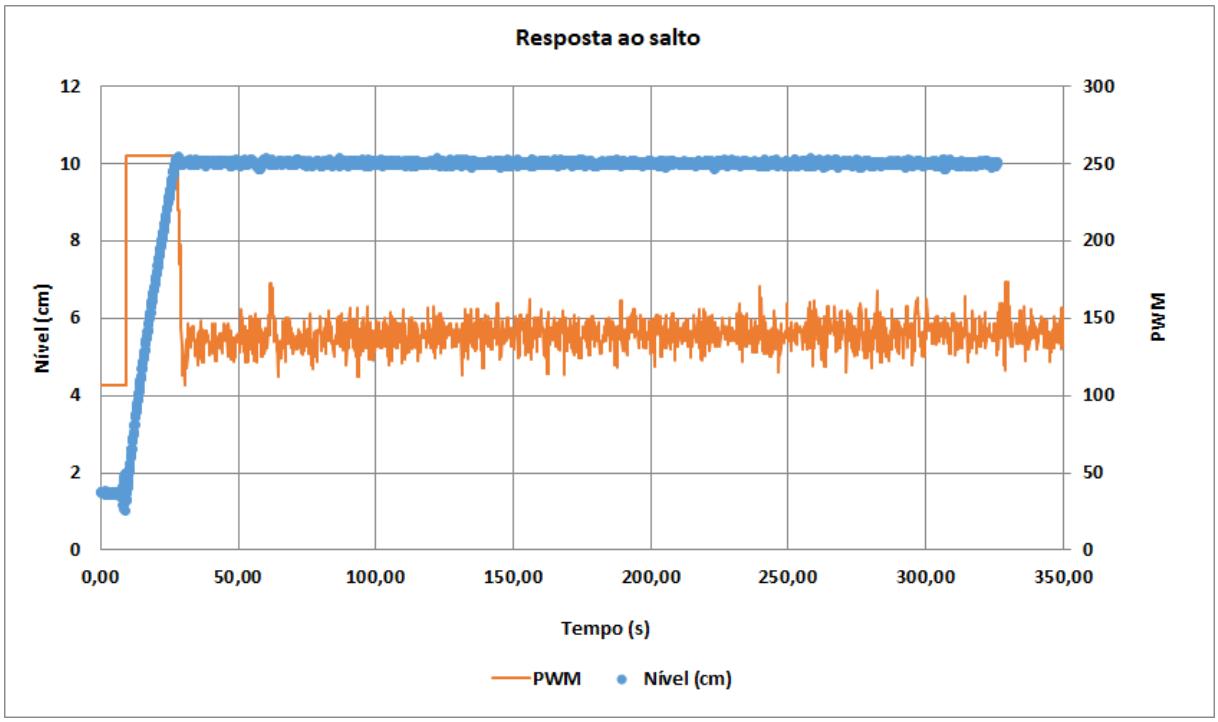
$$G(s) = \frac{5{,}17}{s + 0{,}0112} \quad (4.10)$$

4.3 Comportamento do Controlador PID

Em segundo momento, realizou-se teste dos parâmetros *K*, T_i e T_d previamente obtidos. Para tanto, um novo experimento de salto foi realizado, de forma que houvesse uma mudança no *setpoint* com valor inicial 1,5 cm para valor final 10 cm. O primeiro EE foi atingido ao fixar o *pwm* em 107 no modo *manual* de operação da unidade, enquanto que o segundo EE foi configurado no modo *automático*, dessa forma a mudança no *setpoint* ocorre ao acionar o botão no *painel elétrico* para operar em modo *automático.*

Assim como no tópico anterior, um gráfico foi plotado no software Microsoft Excel, evidenciando os dados de nível, pwm e tempo:

Figura 4.5 – Curva de resposta ao salto em malha-fechada.



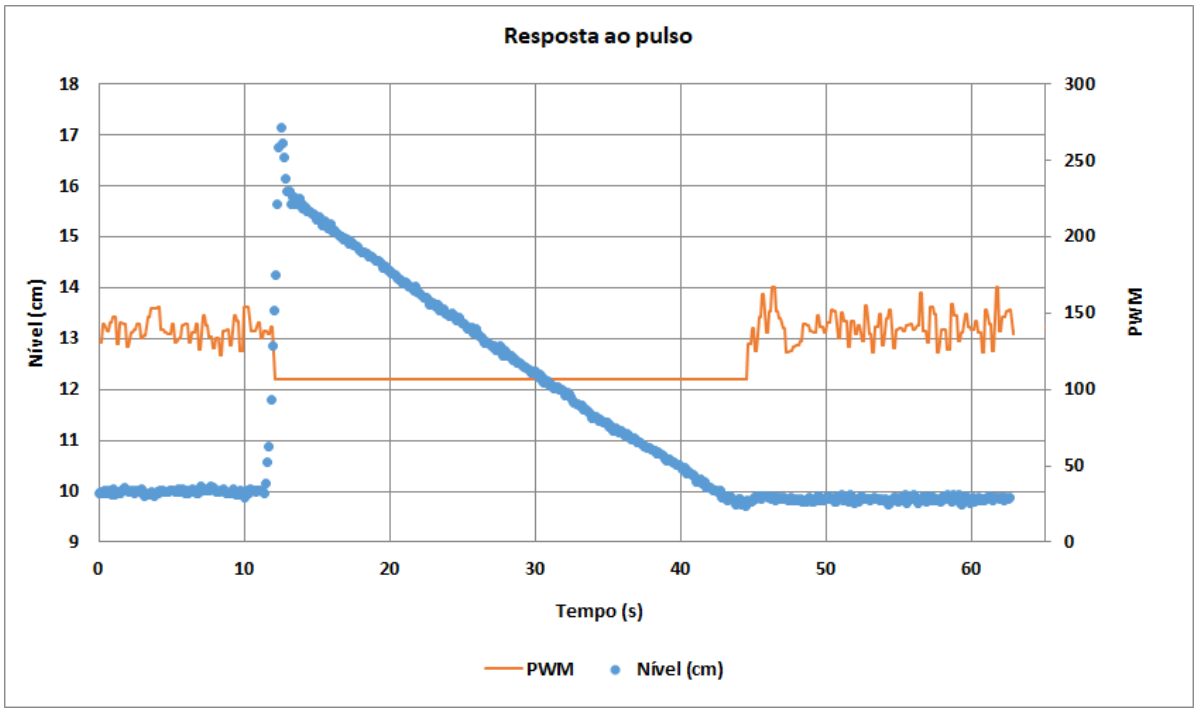
Fonte: Elaborado pelo autor.

De acordo com o registro dos dados, a troca de *setpoint* ocorreu em 8,69 segundos, e logo em seguida, no segundo 9,19, o PID altera o valor do PWM. Neste caso o *overshoot* foi de 1,27% e o tempo de acomodação durou 18,87 segundos. Durante os 584 segundos que a unidade funcionou no novo *setpoint*, o erro máximo registrado foi 1,7%.

Ainda de acordo com a Figura 4.5, percebe-se que nos primeiros segundos houve uma variação anormal na leitura da variável *nível*. Isto é explicado pelo acionamento físico do botão que se encontra no *painel elétrico* gerando certo ruído nos sensores de peso.

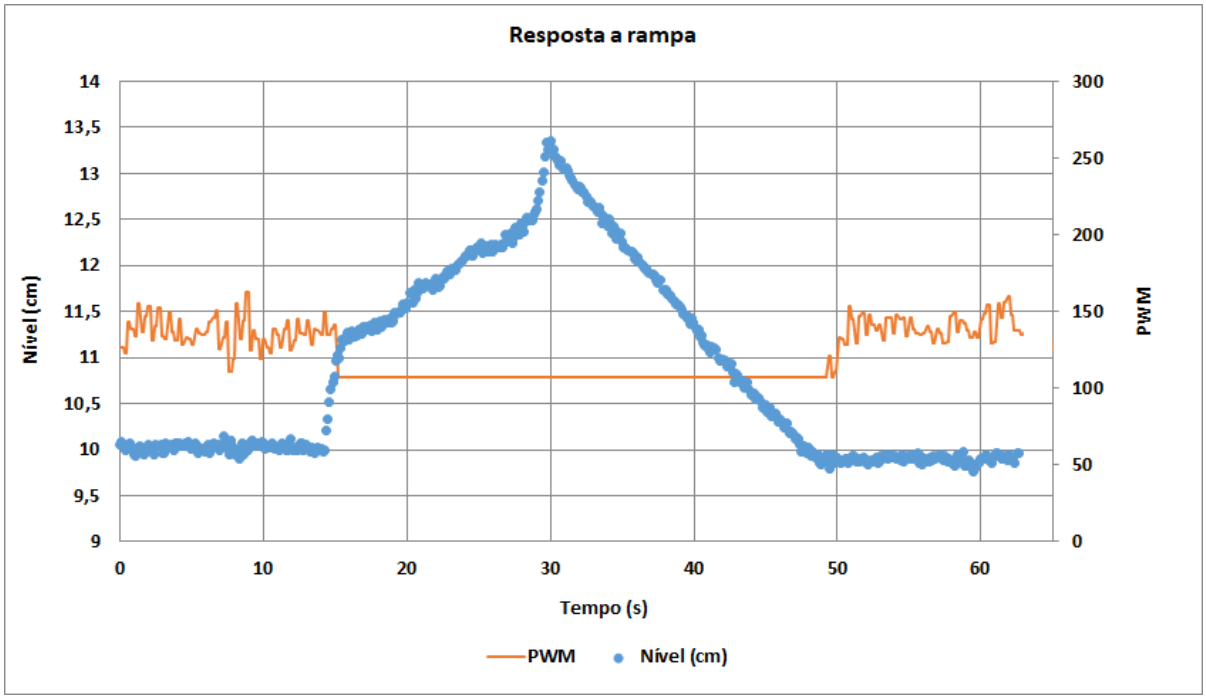
A intenção inicial era realizar uma afinação manual, na forma de mapa de sintonização, no entanto, como os parâmetros iniciais do controlador se demonstraram eficazes, esta afinação não foi necessária. Ao invés disto, uma nova abordagem foi proposta para testar o comportamento do PID por meio da adição de 0,5 L de água no *tanque de controle*. Esta ocorreu de duas formas, como em uma perturbação pulso, no menor intervalo de tempo possível, e como em uma perturbação rampa, ao longo de 15 segundos. As figuras a seguir, mostram a resposta do *sistema* para cada caso:

Figura 4.6 – Curva de resposta do sistema a perturbação pulso.



Fonte: Elaborado pelo autor.

Figura 4.7 – Curva de resposta do sistema a perturbação rampa.



Fonte: Elaborado pelo autor.

O Quadro 4.1 foi gerado para avaliar o comportamento do *sistema* perante as perturbações testadas. Assim, o *tempo morto*, *overshoot*, *tempo de acomodação* e *erro de regime* foram comparados.

Quadro 4.1 – Comparativo das respostas geradas para diferentes perturbações.

Perturbação	**Tempo Morto (s)**	**Overshoot (%)**	**Tempo de acomodação (s)**	**Erro (%)**	**Amostras**
Mudança no setpoint	0,5	1,27	18,87	1,70	2820
Pulso	0,12	2,90	31,42	2,77	551
Rampa (15 s)	0,23	2,04	33,70	2,34	551

Fonte: Elaborado pelo autor.

Ao fixar a mudança no *setpoint* como perturbação de referência, é possível notar que os parâmetros obtidos pelo método de Ziegler-Nichols resultaram em um PID capaz de atuar em diferentes situações com *overshoot* e *erros* extremamente baixos (< 3%). Apesar de existir interferência e ruídos no acionamento do modo automático, estes não impediram que o *tempo morto* se elevasse à grandeza 10^1 segundos.

5. CONCLUSÃO

O trabalho de um modo geral possui uma complexidade elevada, que pôde ser mitigada ao subdividir os tópicos em seções menores, tornando seu entendimento mais intuitivo. Trabalhar com quatro grandes partes permitiu estruturar o progresso e desenvolvimento de maneira lógica, sendo elas: a parte *estrutural*, a parte de *hardware*, *software* e a parte da elaboração do *modelo do sistema*.

A parte *estrutural* envolveu maior dificuldade na execução. Nesta etapa fez-se necessário terceirizar o corte das tábuas de madeira compensada, visto que a dimensão original da chapa é 1,2 x 2,2 m. Com as tábuas menores em mãos foi necessário fazer novos recortes e furos, que com instrumentos comuns e não especializados, demandam elevado tempo para manufaturar. Ademais, a instalação dos tanques necessitou de testes de funcionamento constante para detectar vazamentos e testar a integridade da estrutura como um todo. Finalizado a instalação das partes que envolvem o uso da água, iniciou-se a montagem dos componentes eletrônicos (etapa de *hardware*), que também necessitou de testes de funcionamento, tanto na parte elétrica quanto em códigos, sendo que o principal desafio foi elaborar os esquemas de ligação elétrica e aplicar solda nas peças fixas.

O *software* do projeto foi elaborado na intenção de ser adaptável a diferentes usos da unidade didática, sendo que uma de suas funcionalidades é o controle de nível PID. Aqui foram desenvolvidos inúmeros algoritmos teste, um para cada *hardware*, capaz de calibrar e testar seu funcionamento. Uma vez que seu uso era validado, o código era reescrito de forma mais simples, e adaptado para atuar como uma subrotina do código principal da unidade didática. O uso de bibliotecas disponíveis em fóruns na internet, permitiu agilizar a elaboração destes algoritmos, assim como, reduziu o tempo necessário para implementá-los.

Por fim, a parte de elaboração do *modelo do sistema* teve seu progresso de maneira mais ágil, principalmente por envolver conceitos e teorias já vistos durante a graduação. A abordagem escolhida para gerar o *modelo* e obter os parâmetros do controlador PID foi uma das mais simples disponíveis na literatura. Soma-se a isso o fato da *estrutura* ser baseada no cotidiano de diversas indústrias e em exercícios simples para o ensino de tópicos como balanço de massa e estado transiente.

É possível finalizar este trabalho afirmando que a unidade didática de controle de nível PID representa a consolidação dos conhecimentos teóricos aplicados na prática. Este projeto, seja no desenvolvimento ou uso da unidade, envolve um grande aprendizado que engloba cursos como *Algoritmos e Programação de computadores, Processos Químicos Industriais I, Fenômenos de Transporte I, Operações Unitárias I* e *Controle de Processos Químicos I e II*, sendo que todas estas compõem a grade curricular do curso de *Engenharia Química* da *UFMT*. Considerando o uso da unidade em si, foi possível obter um excelente controle e desenvolver um sistema intuitivo e modular capaz de aceitar novas funcionalidades.

REFERÊNCIAS BIBLIOGRÁFICAS

1 - OGATA, Katsuhiko. **Engenharia de Controle Moderno**. 5. ed. São Paulo: Pearson Education do Brasil, 2014. 822 f.

2 - PLACKO, Dominique (ed.). **Fundamentals of Instrumentation and Measurement**. Londres: Iste, 2007. 555 f.

3 - GARCIA, Claudio. **Modelagem e Simulação de Processos Industriais e de Sistemas Eletromecânicos**. São Paulo: Edusp, 1997. 466 f.

4 - ÅSTRÖM, Karl J.; HÄGGLUND, Tore. **Control PID avanzado**. Madri: Pearson Prentice Hall, 2009. 488 p. 501 f.

5 - GONÇALVES, Marcelo Giglio. **Monitoramento e Controle de Processos**. Brasília: Senai, 2003. 100 p. 99 f.

6 - ÅSTRÖM, Karl J.; HÄGGLUND, Tore. **PID Controllers: theory, design and tuning**. 2. ed. Durham: Instrument Society Of America, 1995.

7 - MORAES JÚNIOR, Deovaldo de. **Transporte de Líquidos: bombas**. São Carlos: Gráfica da Universidade Federal de São Carlos, 1991. 148 p.

8 - PATIENCE, Gregory S. **Experimental Methods and Instrumentation for Chemical Engineers**. 2. ed. Montéal: Elsevier, 2018. 426 p. 426 f.

9 - FELDER, Richard M.; BRENT, Rebecca; PRINCE, Michael J.. **Engineering Instructional Development: programs, best practices, and recommendations**. Journal Of Engineering Education. Raleigh, p. 89-122. jan. 2011.

10 - CARDOSO, Daniel. **PONTE H L298N:** controlando a velocidade de um motor dc com pwm. Disponível em: https://portal.vidadesilicio.com.br/ponte-h-l298n-controle- velocidade-motor/. Acesso em: 07 dez. 2020.

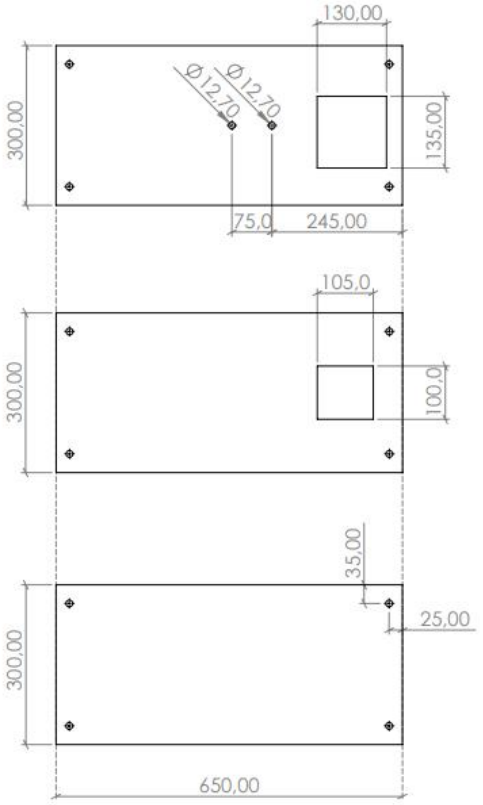
11 - CIRCUIT, Theory. **Water Flow Sensor YF-S201 Arduino Interface**. 2018. Disponível em: http://www.theorycircuit.com/water-flow-sensor-yf-s201-arduino-interface/. Acesso em: 20 out. 2020.

12 - LOCATELLI, Caroline. **BALANÇA COM CÉLULA DE CARGA E HX711**. 2019. Disponível em: https://www.curtocircuito.com.br/blog/balanca-com-celula-de-carga-e-hx711. Acesso em: 12 jan. 2021.

APÊNDICE A - Medidas das tábuas da estrutura

ESTRUTURA:

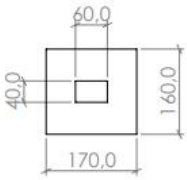
- Vista superior;
- Peças em madeira compensada de 10mm;
- Tábuas 1, 2 e 3: 650mm x 300mm;
- Quando não determinado, furos circulares são de 1/2 in;
- Todas as medidas em mm.



TÁBUA 1

TÁBUA 2

TÁBUA 3

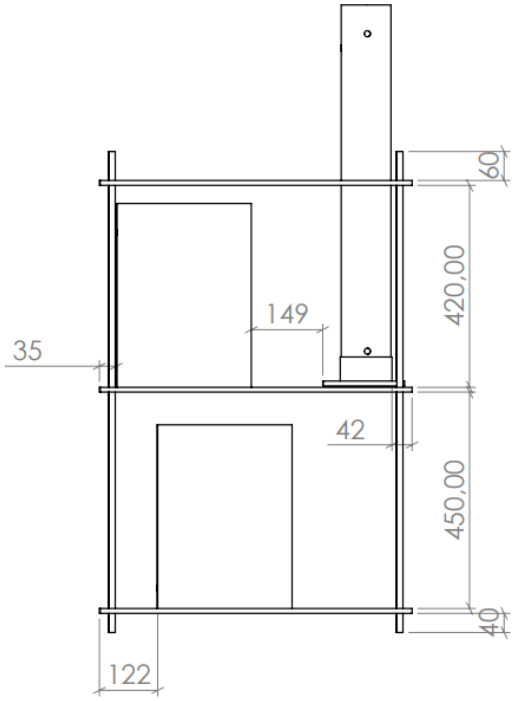


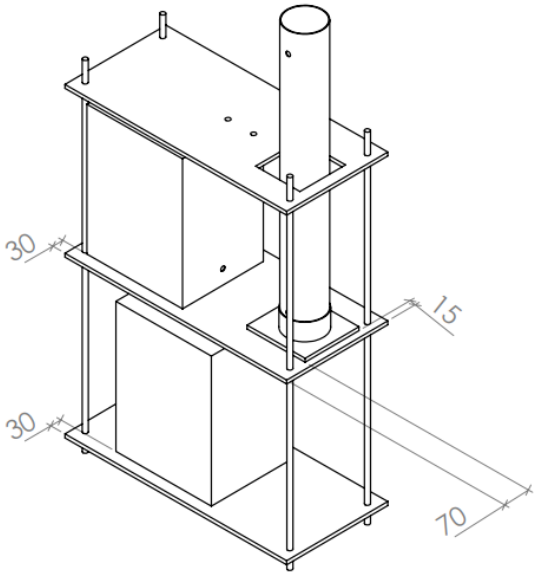
SUPORTE DO TANQUE CONTROLADO

APÊNDICE B - Disposição dos objetos na estrutura

POSICIONAMENTO:

- Todos os reservatórios estão centralizados em relação a profundidade;
- Todas as medidas em mm.





APÊNDICE C - Algoritmo de calibração da balança

// ALGORITMO UTILIZADO PARA CALIBRAR A PONTE DE WHEATSTONE (4 CÉLULAS DE CARGA 50KG + MÓDULO HX711) COM AUXÍLIO DE PESO CONHECIDO

// Adaptado de: Curto Circuito
// Original disponível em: https://www.curtocircuito.com.br/blog/balanca-com-celula-de-carga-e-hx711
// Biblioteca HX711.h disponível em: https://github.com/bogde/HX711

```
#include <HX711.h>                                          //Biblioteca do HX711.h

HX711 balanca;                                              // SCK= pino 42 e DT= pino 43

float fator_balanca = -23270.40;                            // Fator de calibração para ajuste balança
float peso;                                                 // Variável peso

void setup() {                                              // Rotina de configurações
  Serial.begin(115200);                                     // Baud rate da comunicação
  balanca.begin(43,42);
  Serial.println("Remova todos os pesos da balança");       // Printa "Remova todos os pesos da balança"
na COM
  delay(500);                                               // Atraso de 500ms = 0.5s
  Serial.println("Após estabilização das leituras, coloque o peso conhecido na balança");   // Printa "Após
estabilização das leituras, coloque o peso conhecido na balança" na COM
  delay(500);                                               // Atraso de 500ms = 0.5s
  Serial.println("Pressione + para incrementar o fator de calibração");          // Printa "Pressione + para
incrementar o fator de calibração" na COM
  Serial.println("Pressione - para decrementar o fator de calibração");          // Printa "Pressione - para
decrementar o fator de calibração" na COM
  delay(500);                                               // Atraso de 500ms = 0.5s
  balanca.set_scale();                                      // Seta escala
  balanca.tare();                                           // Escala da tara

  long zero_factor = balanca.read_average();                // Realizando a leitura
}

void loop() {                                               // Chama função de loop

  balanca.set_scale(fator_balanca);                         // A balança está em função do fator de
calibração

  peso = balanca.get_units();                               // Variável recebe o peso medido

  if (peso < 0) {                                           // Se o peso for negativo será considerado 0
    peso = 0.00;
  }

  Serial.print("Peso: ");                                   // Printa "Peso:" na COM
  Serial.print(peso, 3);                                    // Printa o peso na serial, com 3 casas decimais
  Serial.print(" kg");                                      // Printa "kg" na serial
  Serial.print(" | Fator de calibração: ");                 // Printa "Fator de calibração:" na serial
  Serial.print(fator_balanca);                              // Printa o fator de calibração na serial
  Serial.println();                                         // Pula linha no serial

  if(Serial.available())                                    // Caso sejam inseridos caracteres no serial
  {
    char temp = Serial.read();
    if(temp == '+')                                         // Se o "+" for pressionado
```

```
    fator_balanca += 0.25;                    // Incrementa 0.25 no fator de calibração
   else if(temp == '-')                       // Caso o "-" seja pressionado
    fator_balanca -= 0.25;                    // Decrementa 0.25 do fator de calibração
  }
}
```

APÊNDICE D - Algoritmo de calibração do fluxímetro

```
// ALGORITMO UTILIZADO PARA CALIBRAR O FLUXÍMETRO yf-s201 (1 - 30 L/min)

// Autor: Felipe Santos Pulcherio

#define VariavelDoEquipamento 7.5          // Valor que deve ser alterado

const byte InterruptPin1 = 2;              // pino digital 2

volatile float TempoEvento1 = 1000;        // Intervalo de tempo para calcular a vazão
float InicioTempoCalculoVazao;             // Inicio da contagem de tempo para calculo da vazao
float ContadorPulsos;                      // Contador de Pulsos
float LPM;                                 // Litros por Minuto

void setup() {
  Serial.begin(115200);                    // Inicialização do serial
  ContadorPulsos = 0;
  LPM = 0;

  attachInterrupt(digitalPinToInterrupt(InterruptPin1), ContaPulso, CHANGE);  // Chama a função ContaPulso
toda vez que o pino mudar seu estado
   InicioTempoCalculoVazao = millis();
  }

void loop() {
  CalculoVazao();                          // Chama a funcao que calcula a vazao
  }

void CalculoVazao() {                      // Funcao do calculo da vazao
  if (millis()- InicioTempoCalculoVazao >= TempoEvento1) {
    detachInterrupt(digitalPinToInterrupt(InterruptPin1));
    LPM = ContadorPulsos / (2 * VariavelDoEquipamento * (TempoEvento1 / 1000));

    Serial.print(LPM, 2);                  // Printa a String no serial

    InicioTempoCalculoVazao = millis();
    ContadorPulsos = 0;
    attachInterrupt(digitalPinToInterrupt(InterruptPin1), ContaPulso, CHANGE);
    }
  }

void ContaPulso() {
  ContadorPulsos++;                        // Conta quantos pulsos (troca de estado) teve
  }
```

APÊNDICE E - Parâmetros dos gráficos no software TelemetryViewer v0.7

Telemetry Viewer v0.7 Settings

GUI Settings:

tile column count = 12
tile row count = 12
time format = Only Time
show 24-hour time = false
show plot tooltips = true
smooth scrolling = true
show fps and period = false
chart index for benchmarks = -1
antialiasing level = 16

Communication Settings:

port = UART: COM4
uart baud rate = 115200
tcp/udp port number = 8080
packet type = CSV
sample rate = 1000

11 Data Structure Locations:

location = 0
binary processor = null
name = Vazão Entrada
color = 0xFF0000
unit = L/min
conversion factor a = 1.0
conversion factor b = 1.0

location = 1
binary processor = null
name = Vazão Saída
color = 0x0000FF
unit = L/min
conversion factor a = 1.0
conversion factor b = 1.0

location = 2
binary processor = null
name = Bomba 1
color = 0x00FF33
unit = PWM
conversion factor a = 1.0
conversion factor b = 1.0

location = 3
binary processor = null
name = Botao LD
color = 0x0000FF
unit =
conversion factor a = 1.0
conversion factor b = 1.0

location = 4

binary processor = null
name = Botao AM
color = 0x0000FF
unit =
conversion factor a = 1.0
conversion factor b = 1.0

location = 5
binary processor = null
name = Botao B1 Manual
color = 0x0000FF
unit =
conversion factor a = 1.0
conversion factor b = 1.0

location = 6
binary processor = null
name = Peso
color = 0xFF0000
unit = Kg
conversion factor a = 1.0
conversion factor b = 1.0

location = 7
binary processor = null
name = Nível
color = 0x0000FF
unit = cm
conversion factor a = 1.0
conversion factor b = 1.0

location = 8
binary processor = null
name = Setpoint
color = 0xFF9900
unit = cm
conversion factor a = 1.0
conversion factor b = 1.0

location = 9
binary processor = null
name = Erro
color = 0x009999
unit = %
conversion factor a = 1.0
conversion factor b = 1.0

location = 10
binary processor = null
name = Tempo Decorrido
color = 0x009999
unit = seg
conversion factor a = 1.0
conversion factor b = 1.0

Checksum:

location = -1
checksum processor = null

6 Charts:

chart type = Time Domain
top left x = 0
top left y = 0
bottom right x = 5
bottom right y = 4
normal datasets = 0,1
bitfield edge states =
bitfield level states =
duration type = Samples
duration = 5000
x-axis = Sample Count
autoscale y-axis minimum = false
manual y-axis minimum = -0.5
autoscale y-axis maximum = false
manual y-axis maximum = 10.0
show x-axis title = true
show x-axis scale = true
show y-axis title = true
show y-axis scale = true
show legend = true
cached mode = false

chart type = Time Domain
top left x = 6
top left y = 0
bottom right x = 11
bottom right y = 4
normal datasets = 6
bitfield edge states =
bitfield level states =
duration type = Samples
duration = 5000
x-axis = Sample Count
autoscale y-axis minimum = false
manual y-axis minimum = -0.2
autoscale y-axis maximum = false
manual y-axis maximum = 6.0
show x-axis title = true
show x-axis scale = true
show y-axis title = true
show y-axis scale = true
show legend = true
cached mode = false

chart type = Time Domain
top left x = 0
top left y = 5
bottom right x = 5
bottom right y = 9
normal datasets = 7,8
bitfield edge states =
bitfield level states =
duration type = Samples
duration = 5000
x-axis = Sample Count
autoscale y-axis minimum = false
manual y-axis minimum = -0.2
autoscale y-axis maximum = false

manual y-axis maximum = 70.0
show x-axis title = true
show x-axis scale = true
show y-axis title = true
show y-axis scale = true
show legend = true
cached mode = false

chart type = Dial
top left x = 6
top left y = 5
bottom right x = 8
bottom right y = 9
normal datasets = 9
bitfield edge states =
bitfield level states =
autoscale dial minimum = false
manual dial minimum = 0.0
autoscale dial maximum = false
manual dial maximum = 100.0
sample count = 5000
show dataset label = true
show reading label = true
show min/max labels = true
show statistics = false

chart type = Dial
top left x = 9
top left y = 5
bottom right x = 11
bottom right y = 9
normal datasets = 2
bitfield edge states =
bitfield level states =
autoscale dial minimum = false
manual dial minimum = 0.0
autoscale dial maximum = false
manual dial maximum = 255.0
sample count = 5000
show dataset label = true
show reading label = true
show min/max labels = true
show statistics = false

chart type = Timeline
top left x = 0
top left y = 10
bottom right x = 11
bottom right y = 11
show = Timeline and Time
normal datasets =
bitfield edge states =
bitfield level states =

APÊNDICE F - Código Fonte da Unidade Didática

```
// ALGORITMO UTILIZADO PARA CONTROLE DE NÍVEL E MONITORAMENTO DE VARIÁVEIS DA UNIDADE DIDÁTICA//
// ESTE CODIGO NECESSITA A UTILIZAÇÃO DA COMUNICAÇÃO SERIAL //
// RECOMENDA-SE O USO DO SOFTWARE GRATUITO "TelemetryViewer v0.7", DISPONÍVEL EM: http://farrellf.com/TelemetryViewer //

//OBSERVAÇÃO: As variaveis que contenham "TMY" se referem ao envio de dados para o Telemetry-v0.7 //

// AUTOR: Felipe Santos Pulcherio, Graduando em Engenharia Química

//Bibliotecas utilizadas:

#include <Bounce2.h>                    // Biblioteca para fazer debounce nos botoes
//Disponível em:                        https://github.com/thomasfredericks/Bounce2
#include <L298N.h>                      // Biblioteca para utilização da ponte H L298N
//Disponível em:                        https://github.com/AndreaLombardo/L298N
#include <HX711.h>                      //Biblioteca do HX711.h
//Disponível em:                        https://github.com/bogde/HX711
#include <PID_v1.h>                     //Biblioteca do PID
//Disponível em:                        https://github.com/br3ttb/Arduino-PID-Library


//--- INPUTS DO USUÁRIO ---//

double Kp = 213.6;                      // Parâmetro Kp do PID (Número com até 1 casa decimal)
double Ti = 1.0;                        // Parâmetro Ti do PID (Número com até 1 casa decimal)
double Td = 0.25;                       // Parâmetro Td do PID (Número com até 1 casa decimal)
double Setpoint = 10.0;                 // Setpoint desejado em cm (Número com até 1 casa decimal)
float Densidade = 1.0;                  // Densidade da água (ou do fluido usado) em Kg/L
float Raio_TC = 5.0;                    // Raio do Tanque de Controle em cm

//--- INICIALIZAÇÃO DE VARIÁVEIS ---//

// Fluxímetros
#define VariavelDoEquipamento1 7.000    // Valor encontrado experimentalmente para o fluximetro 1
#define VariavelDoEquipamento2 7.500    // Valor encontrado experimentalmente para o fluximetro 2

const byte InterruptPin1 = 21;          // pino digital 21 Vazão entrada
const byte InterruptPin2 = 20;          // pino digital 20 Vazão Saída
float TempoEvento1 = 1000;              // Intervalo de tempo para calcular a vazão em ms
float InicioTempoCalculoVazao = 0;      // Inicio da contagem de tempo para calculo da vazao
float VariacaonoTempo = 0;              // Calcula a variação no tempo
volatile unsigned int ContadorPulsos1 = 0;  // Contador de Pulsos
volatile unsigned int ContadorPulsos2 = 0;  // Contador de Pulsos
float LPM1 = 0;                         // Litros por Minuto do fluxímetro 1
float LPM2 = 0;                         // Litros por Minuto do fluxímetro 2


// Botões e debounce dos botões
#define NumeroBotoes 3                  // Define o numero de botoes que o usuario precisa
const uint8_t pinoBotoes[NumeroBotoes] = {30, 33, 34};    // Pinos que possuem botoes conectados (Pino 30 = Chave LD_Geral; Pino 33 = Chave AM; Pino 34 = B1 Manual)

int EstadoLD = 1;                       // Estado inicial da chave seletora L/D Geral (Ligado = 0 / Desligado = 1)
int EstadoAM = 0;                       // Estado inicial da chave seletora Automático/Manual (Automático = 1 / Manual = 0)
```

```
int EstadoB1Manual = 1;                         // Estado inicial do botao que controla a Bomba 1 manual (Ligado = 0 / Desligado = 1)
byte EstavanoManual = 0;                        // Indica que inicialmente não estava no manual
byte BotoesClicados[NumeroBotoes] = {0, 0, 0};          // Cria um array que agrupa os estados (Ligado/Desligado) de todos os botões

Bounce * botoes = new Bounce[NumeroBotoes];             // Inicializacao do vetor botoes

// Ponte H L298N

const byte pinoPWMBomba1 = 2;                   // Pino PWM para Bomba 1 (ENA)
const byte pinoIN1Bomba1 = 3;                   // Pino IN1 para a Bomba 1 (IN1)
const byte pinoIN2Bomba1 = 4;                   // Pino IN2 para a bomba 1 (IN2)

L298N Bomba1(pinoPWMBomba1, pinoIN1Bomba1, pinoIN2Bomba1); // Cria o objeto Bomba 1

// Bombas

unsigned short VelB1Manual = 0;                 // Velocidade da Bomba 1 quando a chave Manual está ligada
unsigned short VelB1Automatico = 0;             // Velocidade da Bomba 1 quando a chave Automática está ligada
byte B1PrecisadeSoftStart = 1;                  // Variável auxiliar
byte a;                                 // Variável auxiliar

// Células de carga (Balança) e Nível

HX711 balanca;                                  // SCK = pino 42; DT = pino 43; Azul(4) = E+; Roxo(3) = A+; Verde(2) = A-; Laranaja(1) = E-;
float fator_balanca = -23270.40;                // Fator de calibração para ajuste balança
float Peso = 0;                                 // Variável peso
float Nivel = 0;                                // Variável Nível

// Buzzer

const byte pinoNivelEmergencia = 26;
const byte pinoBuzzerAtivo = 27;

//PID

double Input;                                   // Variável medida (Nível)
double Output;                                  // Variável manipulada (Velocidade Bomba)
double Erro;                                    // Erro calculado entre o Nível e o Setpoint
double Ki = (1 / Ti);                           // Conervsão de Tempo Integral para Ganho Integral
double Kd = Td;                                 // Conervsão de Tempo Derivativo para Ganho Derivativo
PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT); //Cria o Objeto myPID

//GERAL

float pi = 3.1415;

//--- INICIALIZAÇÃO DO SISTEMA ---//

void setup() {
  Serial.begin(115200);

  // Fluximetros
  attachInterrupt(digitalPinToInterrupt(InterruptPin1), ContaPulso1, CHANGE);   // Chama a função ContaPulso toda vez que o pino mudar seu estado
  attachInterrupt(digitalPinToInterrupt(InterruptPin2), ContaPulso2, CHANGE);   // Chama a função ContaPulso toda vez que o pino mudar seu estado
```

```
  InicioTempoCalculoVazao = millis();


  // Botões e debounce dos botões
  for (int i = 0; i < NumeroBotoes; i++) {
   botoes[i].attach(pinoBotoes[i], INPUT_PULLUP);        // Define cada entrada do vetor (os pinos) como
INPUT_PULLUP
   botoes[i].interval(25);                          // Intervalo do debounce em ms
  }

  // Bombas
  Bomba1.stop();                                  // Bomba 1 inicia parado

  // Células de carga (Balança) e Nível
  balanca.begin(43,42);
  balanca.set_scale(fator_balanca);                    // A balança está em função do fator de calibração
  balanca.tare();                                 // Tara a balança
  long zero_factor = balanca.read_average();             // Tipo da leitura

  // Buzzer
  pinMode(pinoNivelEmergencia, INPUT_PULLUP);
  pinMode(pinoBuzzerAtivo, OUTPUT);

  //PID
  Input = Nivel;
  Output = VelB1Automatico;
  myPID.SetMode(MANUAL);                           // PID inicia desligado
  myPID.SetOutputLimits(107, 255);                 // Variação de voltagem máxima das bombas 5V - 12V
}

//--- CORPO PRINCIPAL DO ALGORITMO ---//

void loop() {
  CalculoVazao();
  QualBotao(BotoesClicados);
  QualNivel();
  AvisoEmergencia();
  Loop_PID();
}


//--- SUBROTINAS ---//

// Fluxímetro 1
void ContaPulso1() {
  ContadorPulsos1++;                              // Conta quantos pulsos (troca de estado) teve
}

// Fluxímetro 2
void ContaPulso2() {
  ContadorPulsos2++;                              // Conta quantos pulsos (troca de estado) teve
}

// Cálculo da vazão    ******CHECAR SE A ALTERAÇÃO DEU CERTO******
void CalculoVazao() {
  if (millis()- InicioTempoCalculoVazao >= TempoEvento1) {
    VariacaonoTempo = millis()- InicioTempoCalculoVazao;
    detachInterrupt(digitalPinToInterrupt(InterruptPin1));
    detachInterrupt(digitalPinToInterrupt(InterruptPin2));
```

```
    LPM1 = ContadorPulsos1 / (2 * VariavelDoEquipamento1 * (VariacaonoTempo / 1000));
    LPM2 = ContadorPulsos2 / (2 * VariavelDoEquipamento2 * (VariacaonoTempo / 1000));

    // caso queira saber os litros até o momento:
    // Litros =+ (LPM / 60);

    Serial.print(LPM1, 2);                    // TMY: Printa os L/min do fluximetro 1 no serial para que o
Telemetry possa ler
    Serial.print(",");                        // TMY: Coloca virgula (CSV)
    Serial.print(LPM2, 2);                    // TMY: Printa os L/min do fluximetro 2 no serial para que o
Telemetry possa ler
    Serial.print(",");                        // TMY: Coloca virgula (CSV)

    InicioTempoCalculoVazao = millis();
    ContadorPulsos1 = 0;
    ContadorPulsos2 = 0;

    attachInterrupt(digitalPinToInterrupt(InterruptPin1), ContaPulso1, CHANGE);
    attachInterrupt(digitalPinToInterrupt(InterruptPin2), ContaPulso2, CHANGE);
  }

  else {
    Serial.print(LPM1, 2);                    // TMY: Printa os L/min do fluximetro 1 no serial para que o
Telemetry possa ler
    Serial.print(",");                        // TMY: Coloca virgula (CSV)
    Serial.print(LPM2, 2);                    // TMY: Printa os L/min do fluximetro 2 no serial para que o
Telemetry possa ler
    Serial.print(",");                        // TMY: Coloca virgula (CSV)
  }

}

// Quais botões estão pressionados?
void QualBotao(byte pBotoesClicados[]) {


  for (int i = 0; i < NumeroBotoes; i++) {
    botoes[i].update();                       // Atualiza todas as entradas do vetor botoes
  }

  EstadoLD = botoes[0].read();                // Lê o valor atualizado da primeira entrada do vetor e armazena
na variavel
  EstadoAM = botoes[1].read();                // Lê o valor atualizado da segunda entrada do vetor e armazena
na variavel
  EstadoB1Manual = botoes[2].read();          // Lê o valor atualizado da terceira entrada do vetor e
armazena na variavel

  switch (EstadoLD) {
    case 1:                                   // EstadoLD Desligado
      Bomba1.stop();

      Serial.print(0);                        // TMY: Printa a Velocidade da bomba 1 (PWM) no serial para que o
Telemetry possa ler
      Serial.print(",");                      // TMY: Coloca virgula (CSV)
    break;

    case 0:                                   // EstadoLD Ligado

      switch (EstadoAM) {
        case 1:                               // Chave está no Automático
```

```
        if (EstavanoManual == 1) {                  // Se estava no manual e mudou para o automático, as bombas
mantém o estado do manual
          VelB1Automatico = VelB1Manual;
          Bomba1.forward();
          Bomba1.setSpeed(VelB1Automatico);          // A velocidade da Bomba 1 continua igual a do Manual
caso haja mudança AM
          EstavanoManual = 0;
        }

        else {
          // Se estava no automático e manteve no automático, não altera valores (Não interferir no PID)
        }

        Serial.print(VelB1Automatico);                // TMY: Printa a Velocidade da bomba 1 (PWM) no serial para
que o Telemetry possa ler
        Serial.print(",");                          // TMY: Coloca virgula (CSV)
       break;

       case 0:                                     // Chave está no Manual
        EstavanoManual = 1;

        if (EstadoB1Manual == 0) {                  // Botao Bomba 1 Manual Ligado
          if (B1PrecisadeSoftStart == 1) {
            VelB1Manual = 107;
            B1PrecisadeSoftStart = SoftStart(1);
          }

          else if (B1PrecisadeSoftStart == 0) {
            if(Serial.available()) {                // Caso sejam inseridos caracteres no serial
              char temp = Serial.read();
              if (temp == '+')                      // Se o "+" for pressionado aumenta 55 no PWM
                VelB1Manual += 55;
              else if (temp == '-')                 // Se o "-" seja pressionado diminui 55 no PWM
                VelB1Manual -= 55;
              else if (temp == '*')                 // Se o "," seja pressionado PWM = 107
                VelB1Manual = 107;

            }
            Bomba1.forward();
            Bomba1.setSpeed(VelB1Manual);
          }
        }

        else {                                      // Botao Bomba 1 Manual Desligado
          Bomba1.stop();
          VelB1Manual = 0;
          B1PrecisadeSoftStart = 1;
        }

        Serial.print(VelB1Manual);                  // TMY: Printa a Velocidade da bomba 1 (PWM) no serial para
que o Telemetry possa ler
        Serial.print(",");                          // TMY: Coloca virgula (CSV)
       break;
     }

   break;
  }

  pBotoesClicados[0] = EstadoLD;
```

```
  pBotoesClicados[1] = EstadoAM;
  pBotoesClicados[2] = EstadoB1Manual;

  Serial.print(EstadoLD);                    // TMY: Printa o estado do botao LD no serial para que o
Telemetry possa ler
  Serial.print(",");                         // TMY: Coloca virgula (CSV)
  Serial.print(EstadoAM);                    // TMY: Printa o estado do botao AM no serial para que o
Telemetry possa ler
  Serial.print(",");                         // TMY: Coloca virgula (CSV)
  Serial.print(EstadoB1Manual);              // TMY: Printa o estado do botao B1Manual no serial para que
o Telemetry possa ler
  Serial.print(",");                         // TMY: Coloca virgula (CSV)
}

void QualNivel(){
  Peso = balanca.get_units();                // Variável recebe o peso medido

  if (Peso < 0) {                            // Se o peso for negativo (leitura do sensor) será considerado 0
    Peso = 0.00;
  }

  Nivel = (Peso / (Densidade * pi * sq(Raio_TC))) * 1000;  // Equação de conversão de peso em Kg para Nível
em Centimetros

  double aux = ((Nivel-Setpoint)/Setpoint)*100;          // Cálculo do erro em relaçãoao setpoint
  Erro = abs(aux);
  double Tempo = millis() / 1000;

  Serial.print(Peso, 3);                     // TMY: Printa o peso no serial para que o Telemetry possa ler
  Serial.print(",");                         // TMY: Coloca virgula (CSV)
  Serial.print(Nivel, 3);                    // TMY: Printa o nível no serial para que o Telemetry possa ler
  Serial.print(",");                         // TMY: Coloca virgula (CSV)
  Serial.print(Setpoint, 1);                 // TMY: Printa o nível no serial para que o Telemetry possa ler
  Serial.print(",");                         // TMY: Coloca virgula (CSV)
  Serial.print(Erro, 2);                     // TMY: Printa o nível no serial para que o Telemetry possa ler
  Serial.print(",");                         // TMY: Coloca virgula (CSV)
  Serial.print(Tempo, 2);                    // TMY: Printa os seg decorridos até o momento no serial para que
o Telemetry possa ler
  Serial.print(",");                         // TMY: Coloca virgula (CSV)
  Serial.println();                          // TMY: Passa para linha de baixo
}

byte SoftStart(byte a) {
  if (a == 1) {                              // O SoftStart é para a Bomba 1
    for (int i = 107; i < 200; i = i + 10) {          // Inicia em 5V e Ternmina em 12v
      Bomba1.forward();
      Bomba1.setSpeed(i);
      delay(200);
    }
    Bomba1.forward();
    Bomba1.setSpeed(200);
  }

  else {
  }

  return 0;
}

void AvisoEmergencia() {                     // Executa um efeito sonoro para indicar sobre-nivel no Tanque 2
```

```
  /*Serial.println(digitalRead(pinoNivelEmergencia));
  Serial.println();
  */
  if (digitalRead(pinoNivelEmergencia) == 0) {
    digitalWrite(pinoBuzzerAtivo, HIGH);
    delay(30);
    digitalWrite(pinoBuzzerAtivo, LOW);
  }

  else {
    digitalWrite(pinoBuzzerAtivo, LOW);
  }
}

void Loop_PID(){
  if (BotoesClicados[1] == 1) {                    // Chave está no Automático
    myPID.SetMode(AUTOMATIC);                      // PID Ligado
    Input = Nivel;                                 // Atualiza valores da variável medida
    myPID.Compute();                               // Roda a biblioteca do PID

    if (Output < 107) {                            // Verifica se Output está dentro do limite permitido
      Output = 107;
    }

    else if (Output > 255) {                       // Verifica se Output está dentro do limite permitido
      Output = 255;
    }

    Bomba1.forward();
    Bomba1.setSpeed(Output);                       // A Bomba 1 recebe o valor modificado pela biblioteca PID
    VelB1Automatico = Output;                      // Atualiza a variável VelB1Automatico para que seja printada
com novo PWM
  }

  else {                                           // Chave está no Manual
    myPID.SetMode(MANUAL);                         // PID Desligado
  }

}
```