LEONARDO COUTO MOLINAR HENRIQUE

DESENVOLVIMENTO DE SISTEMA AUTOMÁTICO DE CONTROLE DE NÍVEL UTILIZANDO ARDUINO

UNIVERSIDADE FEDERAL DE UBERLÂNDIA

FACULDADE DE ENGENHARIA MECÂNICA

2022

LEONARDO COUTO MOLINAR HENRIQUE

DESENVOLVIMENTO DE SISTEMA AUTOMÁTICO DE CONTROLE DE NÍVEL UTILIZANDO ARDUINO

Projeto de fim de curso apresentado ao curso de Graduação em Engenharia Mecânica da Universidade Federal de Uberlândia, como parte dos requisitos para obtenção do título de **Bacharel em Engenharia Mecânica**.

Orientadora: Elaine Gomes Assis

Uberlândia – MG

2020

AGRADECIMENTOS

Gostaria de agradecer, primeiramente, à minha família, por todo suporte e apoio durante todas as etapas da minha vida, desde meu nascimento até o presente momento.

Agradeço especialmente aos meus pais, Humberto e Vilma, por toda a ajuda durante o desenvolvimento do projeto.

Também agradeço minha orientadora, Prof. Dra. Elaine Gomes Assis, pela confiança depositada e auxílio durante esses meses.

E por fim, à Faculdade de Engenharia Mecânica (FEMEC) e seus professores e técnicos juntamente da Universidade Federal de Uberlândia (UFU) por toda educação investida em mim.

RESUMO

O presente trabalho visou desenvolver um protótipo de um controlador de nível para colocar em prática aprendizados adquiridos durante o curso e analisar a eficácia da plataforma Arduino juntamente com *softwares* desenvolvidos em Python para a montagem de um controlador eficiente.

A linguagem de programação Python foi utilizada para desenvolver as interfaces gráficas do protótipo e estabelecer a comunicação com o *hardware*. A plataforma de prototipagem eletrônica Arduino (*open source*) foi utilizada para desenvolver a parte eletrônica das montagens e realizar a aquisição/controle dos dados experimentais. Além de abordar as questões teóricas e conceituais do Arduino, o presente trabalho apresenta descritivos das montagens com foco na operação automática das unidades experimentais.

Além disso, foi feito um comparativo de diferentes métodos matemáticos teóricos e heurísticos para avaliar qual teria a melhor aplicabilidade para este projeto. Os principais resultados demostraram que a montagem teve plena capacidade de realizar o controle e cumpriu com seu objetivo. A análise qualitativa e quantitativa dos resultados pôde demonstrar uma melhor exatidão e estabilidade do controlador quando utilizado métodos que fornecerem uma constante de ganho proporcional (K_C) menor.

Palavras-chave: Arduino, PID, Python, Controlador, Interface Arduino/Python, Engenharia.

ABSTRACT

The present work aimed at the development of a level controller prototype to put into practice the knowledge acquired during graduation course. The Python programming language along with some of its own libraries was used to develop the graphical interfaces of the prototype and establish communication with the hardware. The Arduino electronic prototyping platform (open source) was used to develop the electronic part of the assemblies and to carry out the acquisition/control of the experimental data. In addition to addressing the theoretical and conceptual issues of Arduino, the present work presents descriptions of the assemblies focusing on the automatic operation of the experimental units. In addition, a comparison of different theoretical and heuristic mathematical methods was made to evaluate which one would have the best applicability for this project. The main results showed that the assembly was fully capable of performing the control and it fulfilled its main objective. The qualitative and quantitative analysis of the results could demonstrate a better accuracy and stability of the controller when using methods that provide a smaller proportional gain constant (K_C)

Keywords: Arduino, PID, Python, Controller, Arduino/Python Interface, Engineering.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

CAPÍTULO 1

1 INTRODUÇÃO

Um *hardware* é um termo usado para descrever qualquer um dos componentes físicos de um computador analógico ou digital. Alguns exemplos de *hardware* são processadores, placas de vídeo, placa-mãe, placas embarcadas, placas lógicas, entre outros. Todos eles funcionam em sistema binário, ou seja, a informação apresentada, recebida e processada é toda representada pelos dígitos 0 ou 1. Em um dado instante, em um circuito digital de um *hardware*, a presença de um impulso elétrico representa o dígito 0, enquanto que a ausência de um impulso elétrico representa o número 0. Os impulsos elétricos são chamados de *bits* e são processados por circuitos lógicos (portas lógicas), capazes de associar um determinado pulso de saída para os impulsos elétricos na entrada. A combinação de portas lógicas pode criar circuitos lógicos combinatórios para executar funções e cálculos extremamente complexos de forma rápida e eficiente.

O Arduino é um *hardware*, ou mais especificamente, uma plataforma de prototipagem eletrônica muito versátil e amplamente utilizada por estudantes e profissionais das mais diversas áreas. O objetivo principal do Arduino é tornar o acesso à prototipagem eletrônica mais fácil, mais barata e flexível. Ele é capaz de captar e processar impulsos elétricos, traduzi-los em *bits*, transformá-los e gerar uma saída diferente da captada através de instruções recebidas.

As instruções recebidas pelo *hardware* são transmitidas através do que chamamos de *software*. Um s*oftware* é uma sequência de instruções escritas com o objetivo de executar tarefas específicas. Também pode ser definido como os programas que comandam o funcionamento de um computador.

O Ambiente de Desenvolvimento Integrado do Arduino (IDE) contém um editor de texto para escrita de código em linguagem C/C++. Ele se conecta ao *hardware* do Arduino para fazer upload de programas e se comunicar com eles. Ou seja, essa IDE nada mais é do que o *software* usado para se comunicar com o *hardware* Arduino.

O presente trabalho visa desenvolver um *kit* experimental para laboratórios de controle e automação utilizando *hardware* baixo custo e *software* próprio, que se constituem em uma plataforma de fácil utilização e aquisição por instituições de ensino e pesquisa. O *software* utilizado foi construído baseado em linguagem Python e foi usado para desenvolver os códigos fonte e a interface gráfica do usuário. O *hardware* utilizado

foi a plataforma de prototipagem eletrônica de código aberto Arduino, que foi usada para adquirir os dados da unidade experimental. O trabalho apresenta descrições da configuração experimental, dos controladores PID de tempo real usados e das questões teóricas/conceituais do Arduino. Para a operação da planta experimental foram utilizados algoritmos de programação em C/C++ e Python, aplicando a técnica de controle PID (proporcional integral derivativo). Os controladores PID com base nos métodos Cohen-Coon, minimização da integral do erro absoluto ponderado no tempo (ITAE) e Ziegler-Nichols foram sintonizados para mudanças de *setpoint* e de carga do sistema e execuções em tempo real foram realizadas a fim de fazer uso da teoria de controle em tempo real. Os resultados mostraram que a plataforma desenvolvida se provou adequada para uso em configurações experimentais, permitindo aos usuários comparar suas ideias e expectativas com as evidências experimentais de forma real e com baixo custo. Além disso, a instrumentação é simples de configurar com nível de ruído aceitável e particularmente útil para aprendizagem de controle automação com fins educacionais

Em suma, foi desenvolvida uma plataforma (*software*+*hardware*) para o controle automático do nível de água em um tanque, utilizando minibombas de água, sensor de pressão, placa Arduino e outros componentes eletrônicos.

CAPÍTULO 2

2 REVISÃO BIBLIOGRÁFICA

2.1 Introdução ao Arduino

Arduino é uma plataforma de desenvolvimento de projetos eletrônicos *open source*, ou seja, de código aberto, cujo propósito é tornar o acesso e aprendizagem à prototipagem eletrônica mais fácil, flexível e menos custosa. A placa é capaz de ler valores de entrada (*inputs*) acionados por botões, potenciômetros, células de carga, sensores, etc, e transformá-los em valores de saída (*outputs*) que podem ser usados para acionar e controlar rotores, leds, entre outros componentes eletrônicos. Seu *hardware* é composto por diversas placas de circuito impresso com um microcontrolador que executam tarefas programáveis pelo *software* de interface ARDUINO IDE, que utiliza linguagem C/C++ de programação para se comunicar com a placa e possibilitando o utilizador a transformar os dados e sinais de entrada utilizando operadores lógicos para fornecer instruções e retornar sinais de saída específicos para desenvolver uma tarefa. A interface ARDUINO IDE foi desenvolvida em linguagem Java e permite a escrita e upload de códigos em C/C++ para as plataformas Arduino. Ela pode ser baixada gratuitamente de *https://www.arduino.cc/en/Main/Software*.

O Arduino é, portanto, um tipo básico de computador programável capaz de receber e enviar sinais elétricos para um dispositivo, podendo se tornar uma poderosa ferramenta nas mãos de programadores experientes. Suas principais vantagens são: baixo custo, multiplataforma (podendo ser utilizada em Windows, macOS e Linux), de código aberto e extensível (podendo ser expandida através de bibliotecas C++, que podem ser usufruídas e/ou criadas por qualquer um).

A plataforma foi lançada na Itália em 2005, no *Interaction Design Institute Ivrea*, pelos seus criadores Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca Martino e David Mellis. Ao longo dos anos o Arduino tem sido o núcleo de milhares de projetos, desde objetos e automações de uso cotidiano até complexos instrumentos científicos. Uma comunidade inteira de desenvolvedores, estudantes, entusiastas, profissionais, e programadores se formou em torno dessa plataforma, somando contribuições em um

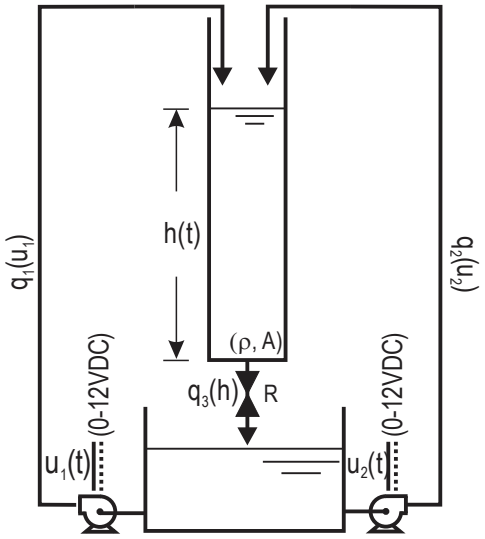
expressivo número de conhecimento gratuito e acessível que ajuda desde profissionais não qualificados até experientes.

É importante salientar que, desde seu lançamento, muitas versões da placa já foram lançadas, cada uma com suas especificidades, capacidades e funções particulares, mas todas possuindo o mesmo propósito geral: tornar o conhecimento da prototipagem eletrônica acessível (é possível comprar placas Arduino entre US$5 a US$40) e de fácil utilização. Além disso, por ser de código aberto, até mesmo outras empresas também lançaram seus próprios modelos compatíveis com Arduino.

2.2 O modelo matemático do processo

O procedimento usual para descrever o comportamento dinâmico de um sistema é usar os princípios fundamentais de conservação de massa, energia e quantidade de movimento do sistema em estudo.

Figura 1. O diagrama do processo



Fonte: Autor

As variáveis do processo consideradas foram:

- $q_1(u_1)$: vazão volumétrica da bomba 1. [m^3/s];
- $u_1(t)$: sinal enviado pelo computador proporcional à tensão aplicada no terminal da minibomba 1. [bytes];
- $q_2(u_2)$: vazão volumétrica da bomba 2. [m^3/s];

- $u_2(t)$: sinal enviado pelo computador proporcional à tensão aplicada no terminal da motobomba 1. [bytes];
- h(t): Nível de líquido dentro do tanque. [m];
- $q_3(t)$: vazão volumétrica de descarga do tanque. [m^3/s];
- A: área de seção transversal do tanque. [m^2];
- ρ: densidade do líquido. [kg/m^3];

Para o desenvolvimento do modelo, algumas considerações foram feitas, tais como: densidade do líquido constante, área de seção transversal do tubo constante, vazão de descarga do tanque diretamente proporcional à raiz da altura do nível de líquido, pois a velocidade de um fluido escoando em um orifício ao fundo segue a equação de Bernoulli ($\sqrt{2 \times g \times h}$), vazões de entrada q_1 e q_2 que variam linearmente com a rotação da motobomba, que por sua vez, depende da tensão nos terminais do motor da mesma, isto é, de 0 a 12 VDC. Este sinal em tensão de 0 a 12 V entregue nos terminais da minibomba depende do sinal PWM enviado pela placa Arduino. Este sinal PMW é gerado através dos comandos md.setM1Speed(velocidade) e md.setM2Speed(velocidade) da biblioteca *'DualVNH5019MotorShield.h'*. O parâmetro "*velocidade*" é codificado na biblioteca com os valores de [0 a 400], sendo cada unidade representada por 12 V / 400 unidades, resultando em um valor de 0,03 V por unidade e [-400 a 0] para rotação da bomba do valor máximo para o zero no sentido anti-horário. Como as minibombas têm sentido de rotação horário, os valores de velocidade de rotação usados estão no intervalo [0 a 400].

Entretanto, devido à altura do tanque e a força necessária para superar a inércia das minibombas e levar água ao topo do tanque, foi observado experimentalmente que as minibombas só conseguem bombear o líquido a partir de um sinal enviado de 200 [$3 \times 10^{-2}\ V$] pelo Arduino.

Também chamado de controle retroativo, o controle em malha fechada é uma operação que reduz a diferença entre a saída de um sistema e um valor externo previamente estabelecido. Ele capta as informações da saída do controlador através de elementos sensores ou transdutores, compara o sinal da saída com o *set-point* e corrige a saída caso a mesma esteja desviando-se dos parâmetros programados. Desta maneira, o controle em malha fechada mantém a variável do processo no seu valor desejado, compensando as perturbações externas e as possíveis não-linearidades do sistema. A Figura 7 mostra esquematicamente uma malha de controle realimentado.

2.3 O controlador Proporcional–Integral–Derivativo (PID)

Controlador proporcional integral derivativo, ou simplesmente PID, é uma técnica de controle de processos que une as ações derivativa, integral e proporcional, fazendo assim com que o sinal de erro seja minimizado pela ação proporcional, zerado pela ação integral e obtido com antecipadamente pela ação derivativa.

Esses três controladores são combinados de forma a produzir um sinal de controle. Por ser um controlador de *feedback*, ele fornece a saída de controle nos níveis desejados. Antes da invenção dos microprocessadores, o controle PID era implementado pelos componentes eletrônicos analógicos. Atualmente, todos os controladores PID são processados pelos microprocessadores. Os controladores lógicos programáveis também possuem as instruções do controlador PID embutidas. Devido à flexibilidade e confiabilidade dos controladores PID, eles são tradicionalmente usados em aplicações de controle de processos industriais.

A ação de controle calculada pelo controlador PID assume a forma temporal da equação (7) em que os parâmetros K_c, τ_I e τ_D são ajustáveis (sintonia do controlador):

$$u(t) = K_c\left(e(t) + \frac{1}{\tau_I}\int_0^t e(t)dt + \tau_D\frac{de(t)}{dt}\right) + u_0 \qquad (7)$$

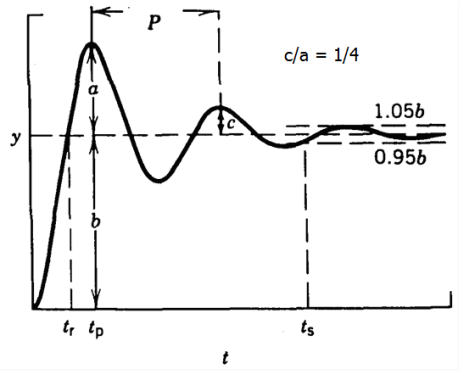
No domínio de Laplace a equação (7) assume a forma da equação (8):

$$G_c = K_c\left(1 + \frac{1}{\tau_I s} + \tau_D s\right) \qquad (8)$$

2.4 O método Ziegler-Nichols

O método de ajuste de Ziegler–Nichols é um método heurístico de ajuste de um controlador PID. Foi desenvolvido por Ziegler e Nichols (1942). Eles usaram um modelo do processo tipo primeira com tempo morto (FOPTD – First Order PlusTime Delay) numa numa malha de controle usando um controlador PID. Eles correlacionaram os parâmetros do PID (K_c, τ_I e τ_D) com os parâmetros do modelo (K, τ e θ) de forma que a resposta temporal do sistema frente a uma perturbação degrau no *setpoint* tivesse uma razão de decaimento de aproximadamente ¼ (ver figura 8).

Figura 2. O método Ziegler-Nichols



Fonte: *Seborg and Mellichamp (1989)*

A figura 1 mostra as fórmulas utilizadas para o ajuste do controlador usando este método.

Tabela 1. O método Ziegler-Nichols

Controlador	Ziegler-Nichols
P	$\boldsymbol{KK_C = \left(\tau/\theta\right)}$
PI	$\boldsymbol{KK_C = 0,9\left(\tau/\theta\right)}$ $\boldsymbol{\frac{\tau_I}{\tau} = 3,33\left(\theta/\tau\right)}$
PID	$\boldsymbol{KK_C = 1,2\left(\tau/\theta\right)}$ $\boldsymbol{\frac{\tau_I}{\tau} = 2,0\left(\theta/\tau\right)}$ $\boldsymbol{\frac{\tau_D}{\tau} = 0,5\left(\theta/\tau\right)}$

Fonte: *Ziegler and Nichols (1942).*

2.5 O método Cohen-Coon

As regras de ajuste Cohen-Coon são adequadas para uma variedade maior de processos quando comparadas ao método Ziegler-Nichols. As regras de Ziegler-Nichols funcionam bem apenas em processos onde o tempo morto é menor que a metade da constante de tempo. Já o método Cohen-Coon funciona bem em processos em que o tempo morto é menor que duas vezes a constante de tempo. Este método também é baseado num processo FOPDT e utiliza os três parâmetros de processo: ganho do processo (K), tempo morto (θ) e constante de tempo (τ).

O método Cohen-Coon corrige a resposta lenta e estável dada pelo método Ziegler-Nichols quando há um grande tempo morto (atraso de processo) em relação à constante de tempo da malha aberta. Um grande atraso de processo é necessário para tornar esse método prático porque, de outra forma, ganhos de controlador excessivamente grandes serão previstos. Este método é usado apenas para modelos de primeira ordem com atraso de tempo, devido ao fato de que o controlador não responde instantaneamente à perturbação.

Para o método Cohen-Coon há um conjunto de configurações pré-determinadas para obter o deslocamento mínimo e a taxa de decaimento padrão de ¼. Uma razão de decaimento de ¼ refere-se a uma resposta que tem oscilações decrescentes de tal maneira que a segunda oscilação terá 1/4 da amplitude da primeira oscilação. O método de Cohen-Coon é recomendado para sistemas de primeira ordem com grande tempo morto ($\theta \gg \tau$). A tabela 2 mostra as fórmulas utilizadas para o ajuste do controlador usando este método.

Tabela 2. O método Cohen-Coon

Controlador	**Cohen-Coon**
P	$\boldsymbol{KK_C = \left(\tau/\theta\right) + 1/3}$
PI	$\boldsymbol{KK_C = 0.9\left(\tau/\theta\right) + 0,083}$ $\boldsymbol{\tau_I = \frac{\theta\left[30 + 3\left(\theta/\tau\right)\right]}{9 + 20\left(\theta/\tau\right)}}$
PID	$\boldsymbol{KK_C = 1{,}33\left(\tau/\theta\right) + 0{,}25}$ $\boldsymbol{\tau_I = \frac{\theta\left[30 + 3\left(\theta/\tau\right)\right]}{9 + 20\left(\theta/\tau\right)}}$ $\boldsymbol{\tau_D = \frac{4\left(\theta/\tau\right)}{11 + 2\left(\theta/\tau\right)}}$

Fonte: *Cohen and Coon (1953).*

2.6 O método ITAE

Os métodos de Ziegler e Nichols e de Cohen-Coon foram desenvolvidos para se obter razão de decaimento de ¼. Respostas com ¼ de decaimento são julgadas na prática industrial com muito oscilatórias. Estes métodos levam em consideração somente dois

pontos da resposta do laço fechado. Um método alternativo é desenvolver relações de projeto baseado na minimização das integrais do erro. Um método popular é o método da minimização da integral do erro absoluto ponderado pelo tempo. Erro aqui é tomado com a diferença entre o *setpoint* e o valor medido da variável controlada.

$$ITAE = \int_0^\infty t|e(t)|dt \tag{9}$$

Esse método também considera que a dinâmica do processo é representada por um modelo FOPDT com os parâmetros do modelo (K, τ e θ). Os parâmetros de sintonia do controlador são, então, determinados de forma a minimizar o valor do $ITAE$ para um problema servo (mudança de *setpoint*) e, distintamente, para um problema regulador (variação na carga do sistema). As soluções do problema de otimização anterior (K_c, τ_I e τ_D) foram correlacionadas aos diferentes valores de (K, τ e θ) para os problemas servo e regulador, levando as relações de projeto dadas na figura 9 a seguir.

Figura 3. O método ITAE

Controller Design Relations Based on the ITAE Performance Index and a First-Order plus Time-Delay Model

Type of Input	*Type of Controller*	*Mode*	*A*	*B*
Load	PI	P	0.859	−0.977
		I	0.674	−0.680
Load	PID	P	1.357	−0.947
		I	0.842	−0.738
		D	0.381	0.995
Set point	PI	P	0.586	−0.916
		I	1.03[b]	−0.165[b]
Set point	PID	P	0.965	−0.85
		I	0.796[b]	−0.1465[b]
		D	0.308	0.929

[a]Design relation: $Y = A(\theta/\tau)^B$ where $Y = KK_c$ for the proportional mode, τ/τ_I for the integral mode, and τ_D/τ for the derivative mode.
[b]For set-point changes, the design relation for the integral mode is $\tau/\tau_I = A + B(\theta/\tau)$. [8]

Fonte: *Seborg and Mellichamp (1989)*

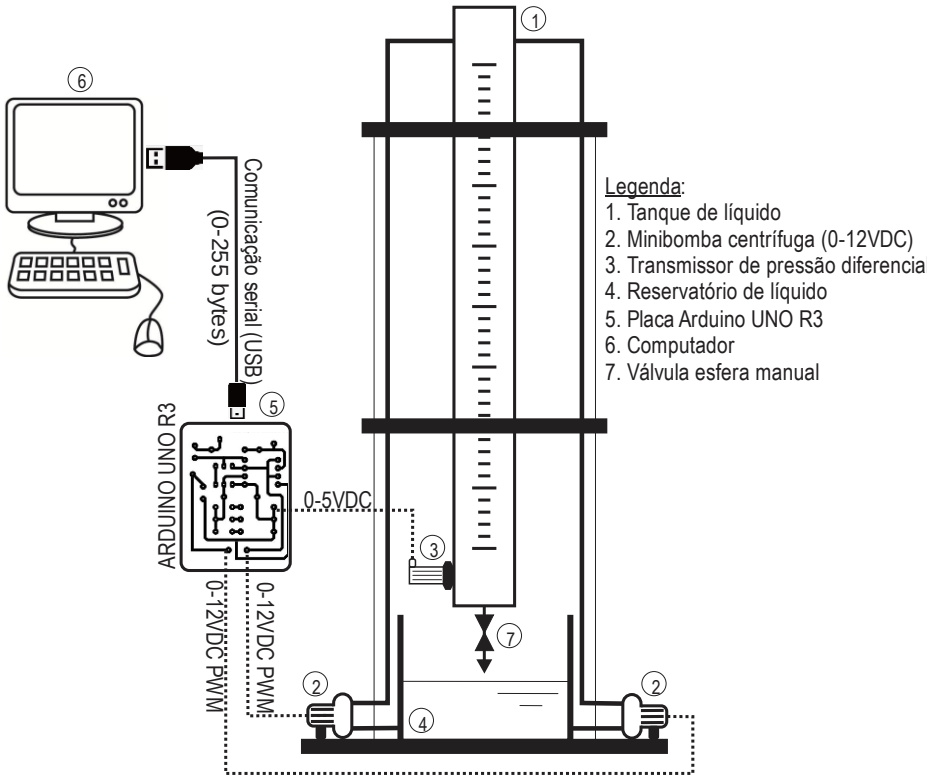
CAPÍTULO 3

3 METODOLOGIA E PROCEDIMENTOS EXPERIMENTAIS

3.1 A construção da unidade experimental

O processo consiste num tanque de armazenamento de líquido que recebe duas vazões de entrada provenientes de duas minibombas centrífugas de 12 VDC. O tanque é construído em acrílico e tem altura de 1 m e diâmetro nominal de 0,05 m. Além das duas vazões de entrada existe uma terceira vazão de saída localizada no fundo do tanque onde existe uma válvula esfera manual. No fundo do tanque existe um sensor/transmissor de pressão diferencial que transmite em tempo real as medidas do nível do tanque para um computador através de um microcontrolador baseado no ATmega328 modelo Arduino UNO R3. O computador roda sob plataforma Windows 10 e o software de aquisição e controle de dados que faz a interface com a unidade experimental é feito com linguagem de programação Python.

Figura 4. Representação esquemática do processo com detalhes da instrumentação



Fonte: Autor

Figura 5. A medição do nível de água

Fonte: Autor

As peças e instrumentos utilizados foram:

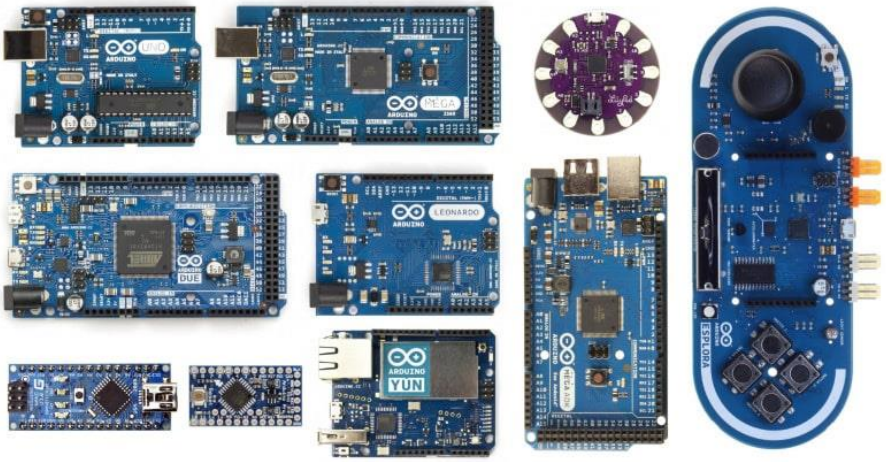
- Sensor/transmissor de pressão diferencial modelo SS302+ piezoresistivo.
- Duas minibombas hidráulicas submersas 12 VDC, 19 W;

- Válvula esfera manual;
- Tanque cilíndrico de acrílico de 2” de diâmetro nominal e 100 cm de altura dotado de visor de nível.
- Microcontrolador baseado no ATmega328 modelo Arduino UNO R3 para aquisição de dados;
- Computador rodando sob plataforma Windows 10;
- Tubulações e acessórios;
- Cabos e conexões para interligação dos instrumentos.

3.2 Placas Arduino

Há uma diversidade de modelos de placas Arduino e, dependendo de cada necessidade do usuário, existirá uma mais recomendada. Basicamente, elas se diferenciam pelo número de canais de entrada e saída, fator de forma e microprocessador. Os modelos são: UNO, Pro Mini, Fio, Nano, Micro, Leonardo, Mega, Lilypad, Duo, Yun, Esplora, Robot, Ethernet, Pro, Tre, entre outras. A figura 1 mostra algumas dessas placas e a figura resume suas características.

Figura 6. Foto de alguns modelos de placas



Fonte: *https://protto.com.br/2020/09/28/como-escolher-uma-placa-eletronica-para-o-seu-projeto/*

Figura 7. Características de placas

Arduino	Microcontrolador	Memória *	Clock	Portas	Tensão de entrada	Tensão de operação	Corrente Pinos Dig.
Pro Mini	ATmega328	Flash: 32KB SRAM: 2KB EEPROM:1KB	16 MHz	Digital: 14 Analógico: 6 PWM: 6	5 - 12 V	5 V	40 mA
Fio	ATmega328P	Flash: 32KB SRAM: 2KB EEPROM:1KB	8 MHz	Digital: 14 Analógico: 8 PWM: 6	3.3 - 12 V	3.3 V	40 mA
Nano	ATmega328	Flash: 32KB SRAM: 2KB EEPROM:1KB	16 MHz	Digital: 22 Analógico: 8 PWM: 6	7-12 V	5 V	40 mA
Micro	ATmega32U4	Flash: 32KB SRAM: 2.5KB EEPROM:1KB	16 MHz	Digital: 20 Analógico: 12 PWM: 7	7 - 12 V	5 V	20 mA
Uno	ATmega328P	Flash: 32KB SRAM: 2KB EEPROM:1KB	16 MHz	Digital: 14 Analógico: 6 PWM: 6	7 - 12 V	5 V	20 mA
Leonardo	ATmega32U4	Flash: 32KB SRAM: 2.5KB EEPROM:1KB	16 MHz	Digital: 20 Analógico: 12 PWM:7	7-12	5V	40 mA
Mega	ATmega2560	Flash: 256KB SRAM: 8KB EEPROM:4KB	16 MHz	Digital: 54 Analógico: 16 PWM:15	7-12	5 V	40 mA
Lilypad	ATmega168 ou AT mega328V	Flash: 16KB SRAM: 1KB EEPROM: 512bytes	8 MHz	Digital: 14 Analógico: 6 PWM:6	2.7 - 5.5 V	2.7 - 5.5 V	40 mA
Duo	AT91SAM3X8E	Flash: 512KB SRAM: 96KB EEPROM:	84 MHz	Digital: 54 Analógico: 12 PWM:12	7-12	3.3V	130 mA
Yun	ATmega32U4	Flash: 32KB SRAM: 2.5KB EEPROM:1KB	16 MHz	Digital:20 Analógico: 12 PWM: 7	5V	5V	40 mA
Esplora	ATmega32U4	Flash: 32KB SRAM: 2.5KB EEPROM:1KB	16 MHz	Digital: - Analógico:- PWM:-	5 V	5 V	
Robot	ATmega32U4	Flash: 32KB SRAM: 2.5KB EEPROM:1KB	16 MHz	Digital: 5 Analógico:4 PWM:6	5 V	5 V	40 mA
Ethernet	ATmega328P	Flash: 32KB SRAM: 2KB EEPROM:1KB	16 MHz	Digital: 14 Analógico: 6 PWM:4	7-12	5	40 mA
Pro	ATmega328	Flash: 32KB SRAM: 2KB EEPROM:1KB	16 MHz	Digital: 14 Analógico: 6 PWM:6	7 - 12 V	5 V	40 mA
Tre	ATmega32U4	Flash: 32KB SRAM: 2.5KB EEPROM:1KB	16 MHz	Digital: 14 Anal: 6 PWM:7		5 V	

Fonte: *https://blog.smartkits.com.br/tipos-de-arduino/*

3.2.1 Arduino UNO

Arduino UNO (Figura 3) é o modelo mais utilizado no mundo. O resultado do seu sucesso se deve pelo baixo custo (aproximadamente Us$22 a original e Us$2.75 as compatíveis) e excelente funcionalidade. A placa Arduino Uno é a primeira de muitas versões do Arduino. É baseada no microcontrolador Atmega328p, sendo uma de suas principais vantagens a atualização do firmware do USB. Com isso, o Arduino aparece como outros dispositivos no computador utilizado. Possui 14 pinos de entrada e saída digitais e 6 entradas analógicas de 10 *bits*, sendo que, em alguns casos, os pinos analógicos podem ser usados como digitais. Dos 14 pinos digitais, 6 podem ser usados

como saídas PWM (*pulse width modulation*) de 8 *bits*. Além disso, também conta com entrada USB, 6 pinos ICSP (usados para programar diretamente os microcontroladores da placa usando o protocolo serial SPI). Possui memória flash de 32KB (0,5KB usado no *bootloader*), memória SRAM de 2KB, memória EEPROM de 1KB e velocidade de clock de 16MHz.

A placa pode ser alimentada diretamente pela conexão USB ou por um adaptador AC/DC. A tensão não deve exceder 12V, sendo recomendado entre 6V e 12V.

Figura 8. Arduino e seus componentes

Fonte: *https://www.arduino.cc/en/uploads/Products/Uno.jpg*

3.2.1.1 Pinos Analógicos, Digitais, PWM e AREF

Pinos digitais: Ao todo são 14 pinos digitais que podem ser configurados como entrada ou saídas digitais conforme a necessidade de seu projeto. Estes pinos são numerados de 0 a 13 e podem ser utilizados para ligar ou desligar um circuito digital. Possuem dois estados, *HIGH* e *LOW*, que produzem, respectivamente, tensões de 5V E 0V nos pinos. Cada um deles pode fornecer no máximo 40mA e possuem resistores internos de 20-50 kΩ.

Pinos PWM: PWM é uma técnica utilizada por sistemas digitais para variação do valor médio de uma forma de onda periódica. A técnica consiste em manter a frequência de uma onda quadrada fixa e variar o tempo que o sinal fica em nível lógico alto, gerando,

assim, uma tensão de saída média que pode variar dentro do intervalo de 0V a 5V. Estão indicados com o símbolo "~" na placa.

Pino AREF: É utilizado para definir uma referência de tensão externa entre 0V e 5V como o limite superior para os pinos de entrada analógicos.

Pinos Serial RX e Serial TX: São os pinos responsáveis pela comunicação serial entre o Arduino e um computador para transferir programas da IDE do Arduino para a placa Arduino.

Pinos analógicos: São os pinos A0 – A5. Recebem um sinal analógico dos sensores e o convertem através do conversor Analógico digital, ADC ou conversor A/D em um valor digital que pode ser lido por computadores via comunicação USB.

3.2.1.2 Pinos Fonte de Energia Elétrica (5V, 3.3V, GND)

A placa também pode ser utilizada como forma de fornecer energia elétrica para *shields* ou circuitos externos que necessitem de tensão. Para isso servem os pinos fonte de energia elétrica, que podem fornecer 3.3V (corrente máxima de 50 mA) e 5V. Além disso, existem 3 pinos de aterramento denominados GND na placa Arduino UNO e todos eles podem ser utilizados para aterrar seu circuito.

3.2.1.3 Botão de Reset

Botão de reinicialização cuja função é desligar a placa e a ligar de novo caso seja necessário. Também vale ressaltar que existe um pino com a mesma funcionalidade do botão *reset*, ou seja, é capaz de forçar a reinicialização ao se aplicar uma voltagem nele. Isso é especialmente útil quando a placa está montada dentro de alguma caixa ou dispositivo e seja necessário forçar a reinicialização de tempos em tempos. Para isso, basta acoplar um acionador externo ligado ao pino *reset*.

3.2.1.4 Microcontrolador ATMEGA328

O microcontrolador, componente principal da placa, funciona como se fosse o "cérebro" do Arduino. É ele o responsável por realizar todas as tarefas e processamentos de maneira eficaz. Possui 8 bits, 32 KB de memória *Flash*, 2 KB de RAM e 1 KB de EEPROM

3.2.1.5 Regulador de Tensão

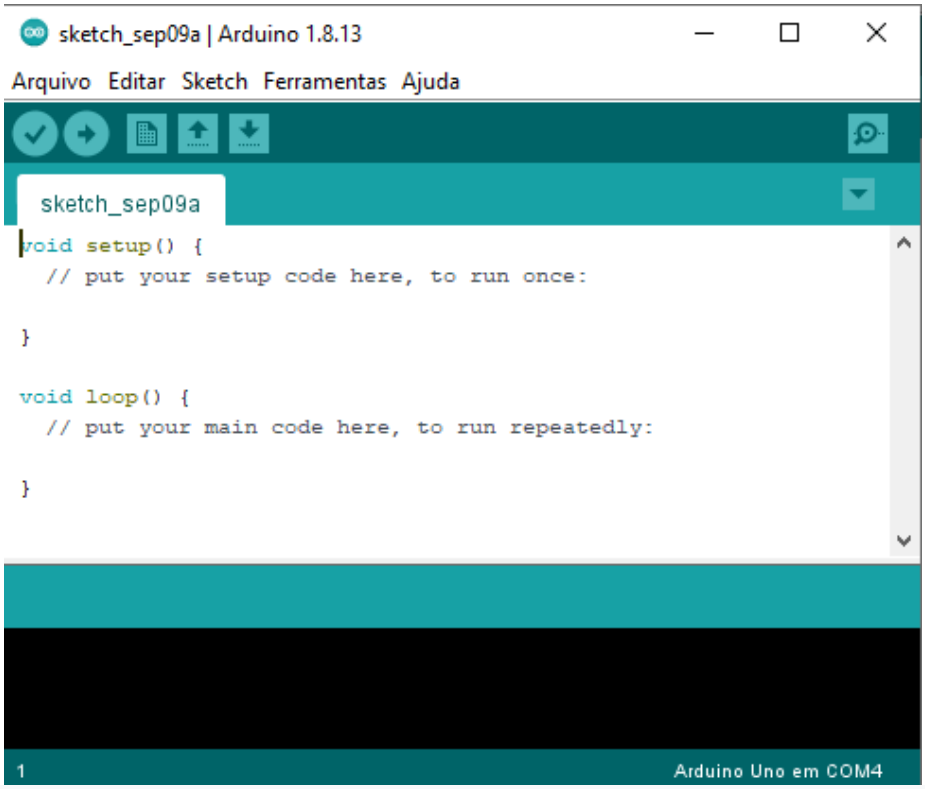
É um dispositivo cuja função é controlar a tensão fornecida à placa, eliminando qualquer quantidade extra que possa danificar o circuito. Possuem um limite de controle, ou seja, não são capazes de controlar a tensão acima de sua capacidade. Por isso, não se deve conectar o Arduino em fontes que forneçam tensão superior à 20 V.

3.3 A IDE Arduino

O ambiente de desenvolvimento integrado (IDE do inglês *Integrated Development Environment*) é uma ferramenta utilizada para criação e edição de códigos, execução de script, depuramento e compilação em um único ambiente. É um programa repleto de funcionalidades que podem ser usadas por muitos aspectos no desenvolvimento de software, que inclui ferramentas de preenchimento de código, plugins, e muitos outros recursos para facilitar o processo de desenvolvimento.

A IDE Arduino contém um editor de texto para escrita de código, uma área de mensagens, um console de texto, uma barra de ferramentas com botões para funções comuns e uma série de menus. Ele se conecta ao hardware do Arduino para fazer upload de programas e se comunicar com eles.

Figura 9. A IDE Arduino



Fonte: Autor

Programas escritos usando o software Arduino (IDE) são chamados de *sketches*. São programas escritos no editor de texto e salvos com a extensão de arquivo .ino. A IDE reconhece apenas a linguaguem de programação C/C++ e, portanto, toda sintaxe e estruturação lógica feita pelo usuário deve seguir esse padrão. O console exibe a saída de texto do software, incluindo mensagens de erro completas e outras informações. O canto inferior direito da janela exibe a placa configurada e a porta serial. Os botões da barra de ferramentas permitem que você verifique e carregue programas, crie, abra e salve esboços e abra o monitor serial.

Os 6 botões encontrados na interface e exibidos na figura 4 são descritos a seguir:


Usado para compilar o *sketch* do Arduino. Ao terminar o código o usuário pode clicar nele uma vez para verificar o código escrito.


Usado para transferir (*upload*) o código compilado para a EEPROM do Arduino. Caso não tenha sido previamente compilado, primeiro ele compila o código e depois o transfere para a EEPROM da placa.


Usado para criar um novo arquivo.


Botão usado para abrir um *sketch* existente.


Botão usado para salvar o sketch atual.


Botão usado para abrir o monitor serial, ferramenta que possibilita o envio e recepção de dados do Arduino para a tela do computador.

Além da compilação de código em C/C++ (é importante ressaltar que muitos recursos do C++ não estão incluídos na IDE), o Arduino possui bibliotecas próprias incluídas no *hardware* da placa que facilitam o controle e manipulação do circuito montado, possibilitando ler e escrever valores em pinos digitais, ler valores analógicos e outras funções específicas. Dentre elas, as mais importantes de serem destacadas são:

- *void setup()*: Executada uma única vez ao compilar o código. É utilizada para definir as configurações iniciais do Arduino.
- *void loop()*: Executada repetidamente de modo iterativo. É aqui que a lógica principal do circuito deve ser escrita.

- *pinMode(pin, mode)*: Define o pino especificado como entrada ou saída.
- *digitalWrite(pin, valor)*: Define um nível de tensão para o pino designado 0 (LOW) ou 5V (HIGH).
- *digitalRead(pin)*: Lê o nível de tensão atual do pino designado 0 (LOW) ou 5V (HIGH).
- *analogRead(pin)*: Lê o nível de tensão atual do pino designado entre de 0 a 1023 (0 a 5V) – 10 bits (2^{10}).
- *analogWrite(pin, Duty_Cycle)*: Envia sinal PWM para o pino designado com ciclo de trabalho de 0 a 255 (0 a 100%) – 8 bits (2^{8}).
- *serial.write, serial.read, serial.print, serial.println*: Comandos usados para interação com o monitor serial. Podem ser usados para definir valores dinamicamente durante a execução da função *loop*, escrever textos ou valores no monitor serial, ler os valores escritos pelo usuário etc.

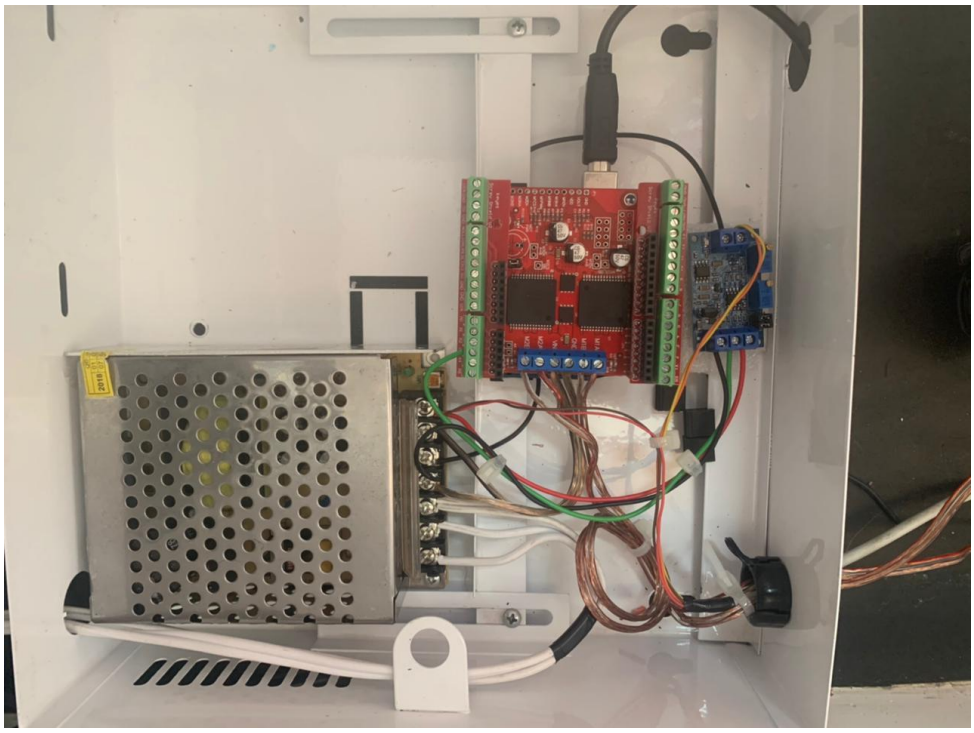
3.4 Introdução aos atuadores, sensores e *shields* utilizados no projeto

Além de seu baixo custo e alta praticidade, uma das grandes vantagens do Arduino é a capacidade de lidar com uma ampla variedade de sensores, atuadores, *shields* etc. Sensores são dispositivos que respondem a um estímulo físico ou químico de maneira específica, produzindo um sinal que pode ser lido e transformado para fins de medição e/ou monitoramento. Atuadores são dispositivos que produzem movimento, convertendo energia pneumática, hidráulica ou elétrica, em energia mecânica. Shields são placas que podem ser acopladas sobre a placa Arduino para ampliar suas capacidades e/ou adicionar funcionalidades, possibilitando criar módulos de hardware com várias funções diferentes.

Neste trabalho, a intenção é utilizar um sensor de pressão para medição de nível d'água e medir, manipular e controlar as vazões de entrada através de bombas hidráulicas de forma a se obter uma altura estável em um sistema em que o fluxo de líquido é variável e dependente do nível de água no recipiente.

A figura a seguir exibe a montagem dos componentes eletrônicos utilizados no projeto (fonte 12 V, shield e placa Arduino).

Figura 10. A montagem dos componentes eletrônicos



Fonte: Autor

3.5 Medição de nível de água

3.5.1 O Hardware

Neste projeto, para a medição do nível de água em tempo real, foi utilizado o transmissor de pressão diferencial SS302+ instalado diretamente no tubo.

Os transmissores de pressão piezoresistivos da série SS302+ possuem maior estabilidade e menor nível de interferência se comparados à série SS302. Com um sistema eletrônico de alta qualidade montado dentro do sensor, fornece sinal de tensão de 4-20mA com grande constância e baixo ruído.

Figura 11. Transmissor de pressão SS302+

Fonte: *http://www.pressuresensorsuppliers.com/image/58f4aae08043f.png*

As especificações do sensor são:

- Modelo: SS302+
- Sinal de saída: 4-20mA / 0-10V
- Voltagem de alimentação: 9-30Vcc / 12-30Vcc
- Tempo de resposta: < 4ms
- Intervale de pressão suportado: 0.1bar até 600bar
- Peso: 150-300g
- Pinos: 1 - alimentação, 2 - saída, 3 - não utilizado, 4 - Terra
- Fios de conexão: Vermelho - alimentação, Preto – saída

3.5.2 O Software

Após a devida conexão dos fios de entrada e saída, o sensor passa a transmitir valores que variam de 0 a 1000 para IDE Arduino. Para a leitura dos valores recebidos, o pino conectado foi definido como analógico e foi criada uma função que lê o valor transmitido a esse pino e os filtra antes de enviá-lo à interface do Python através da comunicação serial. Tal filtragem é importante para diminuir ruídos e possíveis valores *outliers*. Basicamente, a função funciona criando um laço iterativo que é executado 10 vezes e recebe 10 valores de pressão do sensor durante esse ciclo. Após as iterações, é eliminado o menor valor, o maior valor, e a média entre os 8 valores restantes é retornada.

Entretanto, os valores lidos diretamente pelo sensor (0 a 1000) não condizem com a realidade do nível de água no tubo e, portanto, fez-se necessária a realização de uma calibração do sistema. Para isso, diferentes valores de pressão foram obtidos para diferentes valores do nível de água medidos diretamente no tubo através de uma fita métrica e, após a obtenção dos resultados, foi criado um gráfico de dispersão no *software* Excel e encontrado a melhor função que descreve a relação entre o valor medido pelo sensor e o nível real de água no sistema. Os resultados encontrados estão descritos na figura 9.

Por fim, a função que descreve a relação entre os valores foi implementada diretamente na IDE do Python para tratar os dados recebidos pelo Arduino e obter o nível real de água com um bom nível de exatidão.

Figura 13. Função implementada diretamente na IDE

```
def transformToLevel(value):
    if value < 1:
        return 0
    else:
        return float(value)*0.1079 + 1.7544
```

Fonte: Autor

Figura 12. Função de filtragem dos valores de pressão

```
int getFilteredPressure(){
  int sensorValues[10];
  int maxValue = 0;
  int minValue = 0;
  int maxIndex = 0;
  int minIndex = 0;
  for (int i = 0; i < 10; i++){
    sensorValue = analogRead(sensorPin);
    sensorValues[i] = sensorValue;
    if (i==0){
      minValue = sensorValue;
      minIndex = i;
    }
    if (sensorValue > maxValue) {
      maxValue = sensorValue;
      maxIndex = i;
    }
    if (sensorValue < minValue) {
      minValue = sensorValue;
      minIndex = i;
    }

    if (i==9) {
      float arraySum = 0;
      for (int j=0;j<10;j++){
        if (j != minIndex && j != maxIndex){
          arraySum += sensorValues[j];
        }
        if (j==9) {
          float filteredPressure = arraySum/8;
          return filteredPressure;
        }
      }
    }


  }
}
```

Fonte: Autor

3.6 Controle de vazão utilizando minibombas hidráulicas 12V

3.6.1 O Hardware

Para este projeto, é feito o acionamento e controle de rotação de duas minibombas hidráulicas submersíveis 12V de corrente contínua, que bombeiam água diretamente para o topo do tubo. O objetivo é manipular automaticamente a rotação do motor utilizando o Arduino. Para isso, manipula-se a tensão de 0 a 12V nos terminais da minibomba utilizando o *motor shield* VNH5019.

Figura 14. Motor Shield VNH5019

Fonte: *https://i.ebayimg.com/images/g/Q7sAAOSwyi5ceigN/s-l500.jpg*

Figura 15. Minibomba hidráulica

Fonte: *https://ae01.alicdn.com/kf/Bomba-submers-vel-ultra-silenciosa-port-til-do-motor-12v-24v.jpg_Q90.jpg*

As características das minibombas são:

- Material: Plástico
- Tensão: 12V DC
- Corrente Nominal: 1000mA
- Corrente Máxima: 1600mA
- Potência: 19W
- Max Taxa De Fluxo: 800 L/H
- Entrada/Saída: 1/2" macho
- Comprimento da bomba: 7.7cm
- Altura: 6.5cm
- Diâmetro: 4.2cm
- Comprimento do fio: 48cm
- Elevação máxima: 5m (O fluxo diminui a cada metro que se eleva o nível da água).

O *motor shield* utilizado possui 6 conectores na parte direita destacados na cor azul, eles são utilizados para a entrada e saída dos motores e de energia. Além disso, nos lados superior e inferior, existem 14 entradas que possuem pinos abaixo do *shield* que se acoplam às entradas do Arduino, de modo que seja possível conectar dispositivos diretamente no *shield* e ligá-los com a placa. A figura 11 mostra como fica o conjunto Arduino-*shield* após o acoplamento.

Figura 16. Acoplamento da placa com motor shield

Fonte: *https://a.pololu-files.com/picture/0J5214.1200.jpg?4e32ab33ad064dbf408c150d2a216e39*

Para conectar as bombas basta soldar dois fios nos terminais “+” e “-” da minibomba e depois conectá-los aos terminais M1A/M1B e M2A/M2B do *shield*. A velocidade também pode ser variada em incrementos de 0,39% (8 *bits*) usando os pinos PWM do Arduino.

3.6.2 O Software

O *shield* utilizado neste projeto possui uma biblioteca própria feita pelos desenvolvedores para utilização na IDE Arduino, facilitando a escrita do *script* e simplificando o *sketch*. Para a instalação, é necessário seguir o passo a passo abaixo:

1. Na IDE do Arduino, abra o menu "*Sketch*", selecione "Incluir Biblioteca" e depois "Gerenciar Bibliotecas...".
2. Procure por "DualVNH5019MotorShield".

3. Clique na entrada "DualVNH5019MotorShield".
4. Clique em "Instalar".

Feito isso, a biblioteca já se encontrará pronta para o uso na IDE. Para utilizá-la, é necessário importá-la através do comando *"#include 'DualVNH5019MotorShield.h'"*. Com isso, basta definirmos uma variável para ser atribuída a biblioteca (como por exemplo, "md"), inicializarmos através do comando "*md.init()*" e, após isso, podemos então controlar as velocidades das bombas 1 e 2 através dos comandos *md.setM1Speed(velocidade)* e *md.setM2Speed(velocidade)*, respectivamente. Os valores de velocidade definidos devem estar dentro do intervalo -400 até 400, sendo -400 a máxima tensão com corrente fluindo no sentido contrário (invertendo a rotação da bomba) e 400 a máxima tensão com corrente fluindo no sentido padrão.

3.7 INTEGRAÇÃO PYTHON – ARDUINO

3.7.1 Introdução

O Arduino e sua IDE são ferramentas muito poderosas e com possiblidades muito grandes de aplicações, utilizadas amplamente por inúmeros pesquisadores, estudantes, empresas e entusiastas. Entretanto, por utilizar linguagem C/C++, que são linguagens naturalmente mais complexas por serem consideradas de baixo nível (fornecem pouca abstração da arquitetura do conjunto de instruções de um computador), e não possuir compatibilidade com todas as bibliotecas e funcionalidades das respectivas linguagens, pode acabar causando certa objeção de usuários que não estão acostumados com elas.

Por outro lado, a linguagem Python, considerada de alto nível (com forte abstração dos detalhes do computador, ou seja, comandos e sintaxes ligeiramente mais compreensíveis e intuitivos) e de mais fácil aprendizado, além de ser a segunda mais utilizada por desenvolvedores do mundo todo, seria ideal para ser utilizada com o Arduino, cujo intuito é justamente tornar a prototipagem eletrônica mais simples e acessível para a maioria das pessoas. É aqui que a biblioteca *PySerial* entra em cena. Ela possibilita o acesso à porta serial do computador, ou seja, permite que códigos escritos em Python leiam e manipulem os valores que a IDE Arduino utiliza para se comunicar

com a placa, criando uma “ponte” indireta entre o programa em Python e o circuito montado.

3.7.2 A interface Python – Arduino (PySerial)

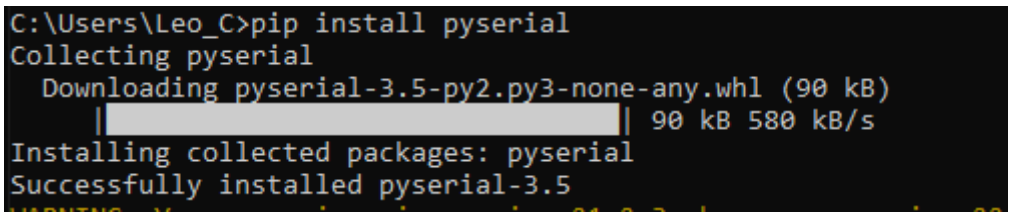
PySerial é uma biblioteca que fornece suporte para conexões seriais ("RS-232") em uma variedade de dispositivos diferentes: portas seriais antigas, *dongles* Bluetooth, portas infravermelhas e assim por diante. Ele também suporta portas seriais remotas via RFC 2217.

Com isso, torna-se possível ler e escrever dados para a placa através de um script desenvolvido em Python, podendo desfrutar, assim, de todo potencial e robustez dessa linguagem, que possui bibliotecas próprias para criação de interface gráfica, geração de gráficos de todos os tipos, além de funções matemáticas próprias para resolução de problemas de alto nível de complexidade.

Para instalar e iniciar a utilização dessa biblioteca, um simples passo a passo deve ser seguido:

- Com o Python instalado no computador, execute o comando “*pip install pyserial*” no prompt de comando
- Em uma IDE de preferência, crie um arquivo com a extensão “*.py*” e importe a biblioteca utilizando o comando “*import serial*”
- Crie uma variável e atribua a ela o comando *serial.Serial(nome, baudrate, timeout)*, sendo os parâmetros utilizados na função: o nome da portal serial utilizada pelo Arduino, o valor do *baudrate* configurado na IDE Arduino e o valor do tempo máximo de conexão escolhido (*timeout*), respectivamente.

Figura 17. Instalação da biblioteca PySerial



Fonte: Autor

Figura 18. Importação e configuração da porta serial utilizada pela placa

```
import serial

serial = serial.Serial('COM3', baudrate=9600, timeout=0.3)
```

Fonte: Autor

Feito isso, já é possível, então, ler, manipular, interpretar e exibir os dados recebidos e escrevê-los na porta serial através dos comandos *serial.write(valor)* e *serial.read()*.

3.7.3 A comunicação Python – Arduino (Handshake)

Na computação, *handshake* é o termo usado para designar a comunicação entre dois dispositivos ou programas e é usado para, por exemplo, coordenar uma aplicação que utiliza dois dispositivos interdependentes, mas com funções diferentes. Este tipo de sincronização pode ser feito de duas formas diferentes, uma comunicação de uma via, em que apenas um dos lados envia informações, enquanto o outro lado as lê. Ou uma comunicação de duas vias, em que ambos os lados leem e enviam informações, sendo certamente mais complexa de ser implementada.

Para este projeto, precisamos que a IDE Arduino receba os sinais do sensor, envie-os para a porta serial, que será lida pelo programa em Python que, então, deve exibi-los na forma de gráficos em uma interface visual interativa e, ao mesmo tempo, deve trabalhar com esses valores em equações matemáticas e numéricas para adquirir as velocidades ideais das minibombas e, por fim, escrever os novos valores na portal serial, que será novamente lida pelo Arduino e os novos valores de velocidade serão enviados para as minibombas em forma de tensão. Logo, tanto o programa escrito na IDE Arduino quanto o programa escrito em Python devem ler e enviar informações pela porta serial e, portanto, deve ser desenvolvida uma comunicação de duas vidas, ou *two-way handshake*, entre as duas interfaces.

Para atingir esse objetivo, foi necessário desenvolver um conjunto de caracteres padrão que pudessem ser interpretados e escritos por ambas as partes na porta serial utilizada. O padrão de comunicação é executado da seguinte maneira:

1. O programa em Python solicita para o Arduino os valores das velocidades e pressão escrevendo na porta serial o caractere “r”.
2. O programa em Arduino detecta que o caractere “r” foi escrito na porta serial e, então, responde o pedido escrevendo os valores na porta serial da seguinte maneira: “V1 V2 P”, sendo V1 a velocidade da bomba 1 (0 a 400), V2 a velocidade da bomba 2 (0 a 400) e P o valor de pressão detectado no sensor (0 a 1000).
3. O programa em Python, então, lê a sequência de caracteres escritos pelo Arduino e os separa, executa os cálculos necessários para calcular os novos valores de velocidade e, então, escreve novamente na porta serial a seguinte sequência de caracteres: “s V1 V2”, sendo V1 a nova velocidade da bomba 1 e V2 a nova velocidade da bomba 2.
4. O programa em Arduino detecta que o caractere “s” foi escrito na porta serial e, então, define os novos valores de velocidade das minibombas 1 e 2 com base no valor lido.

3.7.4 Leitura e escrita de dados na porta serial pela IDE Arduino

Na IDE Arduino, para que a comunicação fosse feita de forma efetiva, foram desenvolvidas três funções que serão explicadas a seguir.

Figura 19. A função *readPort()*

```
String readPort(){
  String conteudo = "";
  char caractere;
  while(Serial.available()>0){
    caractere = Serial.read();
    if (caractere != '\n'){
      conteudo.concat(caractere);
    }
    delay(5);
  }
  return conteudo;
}
```

Fonte: Autor

A função *readPort()* tem o objetivo de obter todos os caracteres escritos na porta serial e agrupá-los em uma *string*. Ela executa um laço que se repetirá iterativamente enquanto houver algum valor na porta e cada caractere encontrado é adicionado à uma variável.

Figura 20. A função *separateString()*

```
String separateString(String data, char separator, int index)
{
    int found = 0;
    int strIndex[] = { 0, -1 };
    int maxIndex = data.length() - 1;

    for (int i = 0; i <= maxIndex && found <= index; i++) {
        if (data.charAt(i) == separator || i == maxIndex) {
            found++;
            strIndex[0] = strIndex[1] + 1;
            strIndex[1] = (i == maxIndex) ? i+1 : i;
        }
    }
    return found > index ? data.substring(strIndex[0], strIndex[1]) : "";
}
```

Fonte: Autor

A função *separateString()* tem o objetivo de receber um conjunto de caracteres, ou seja, uma *string*, e separá-las em *substrings* criadas a partir de um caractere de referência. Ela recebe 3 parâmetros quando é chamada, o primeiro é a *string* que o usuário deseja separar, o segundo é o tipo de separador, ou seja, qual caractere deve ser usado como ponto de separação (no caso deste projeto, por padrão, sempre será o espaço em branco) e o terceiro é o índice da *string* que a função deve retornar, ou seja, qual das *substrings* criadas deve ser retornada pela função.

Com isso, após ler a porta serial, é possível identificar qual desses valores devem ser usados como as novas velocidades das minibombas.

Figura 21. A função *loop()*

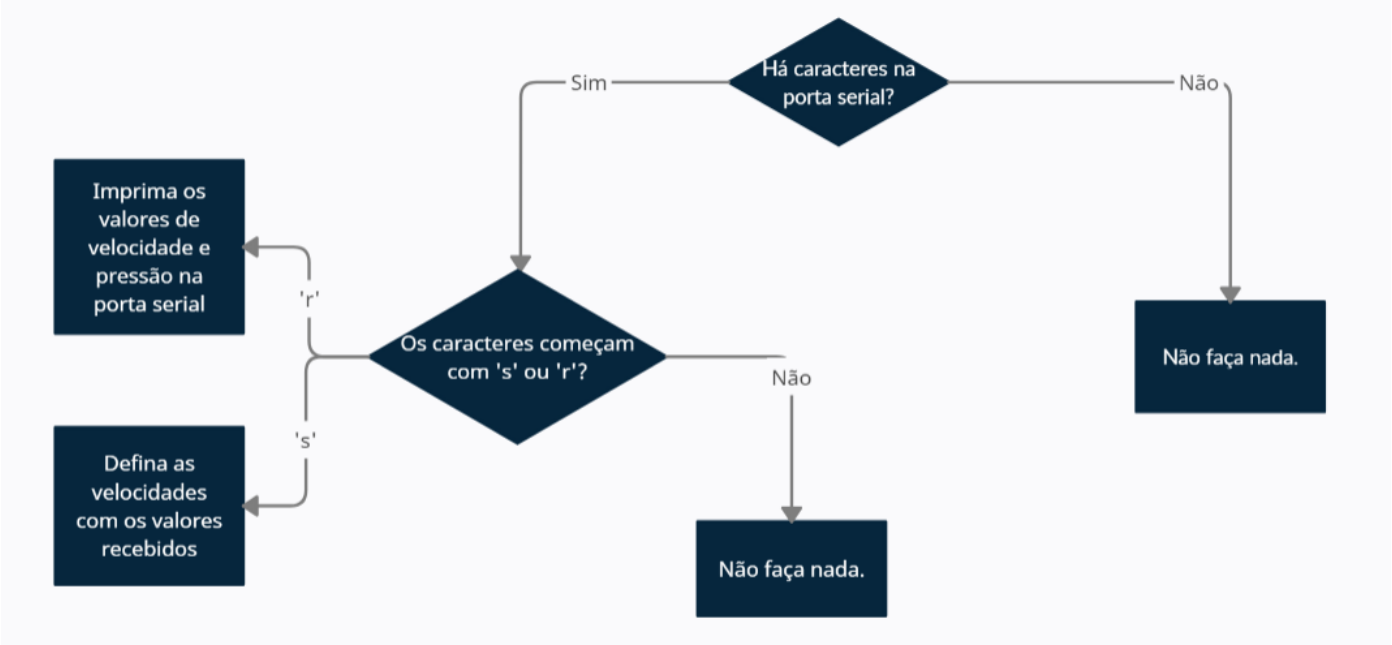
```
void loop() {
  if(Serial.available()>0){
    String value = readPort();
//    int communicate = separateString(value,' ',0).toInt();
    String communicate = separateString(value,' ',0);
    if (communicate.equals(String('r'))){
      pressureValue = getFilteredPressure();
      Serial.print(speedMotor1);
      Serial.print(' ');
      Serial.print(speedMotor2);
      Serial.print(' ');
      Serial.println(pressureValue);
      clearBuffer();
    }
    else if (communicate.equals(String('s'))){
      setSpeeds(value);
      clearBuffer();
    }

  }
}
```

Fonte: Autor

A função *loop()* é a principal função dentro da IDE Arduino. Ela já é própria da interface e tem o objetivo de se repetir indefinidamente. Dentro dela colocamos toda a estrutura lógica que queremos que seja executada em um ciclo contínuo.

Figura 22. Estrutura lógica da função *loop()*



Fonte: Autor

Com isso, conseguimos deixar apenas o indispensável sob responsabilidade da IDE Arduino, que é o controle direto das tensões enviadas para as minibombas e a leitura direta da pressão medida pelo sensor, e transferimos toda responsabilidade de cálculos matemáticos, construção de gráficos e exibição dos dados em uma interface visual para o programa em Python.

3.7.5 Leitura e escrita de dados na porta serial pela IDE Python

Com Python, a leitura e escrita dos dados se torna tarefa bem mais simples por ser uma linguagem naturalmente mais compreensiva e possuir inúmeras funções próprias que facilitam o trabalho de iteração e modificação de *strings*.

Para isso, apenas duas simples funções são necessárias, uma para solicitar e obter os valores atuais, e outra para escrever na porta serial os novos valores de velocidade das minibombas.

Figura 23. A função *getValues()*

```
def getValues():
    serial.write(b'r')
    reading = str(serial.readline()).split("'")[1].split("\\")[0].split(' ')
    while reading[0] == '':
        serial.write(b'r')
        reading = str(serial.readline()).split("'")[1].split("\\")[0].split(' ')
    return reading
```

Fonte: Autor

A função *getValues()* inicialmente imprime o caractere “r” na porta serial (que é o caractere de referência definido para solicitar valores para o Arduino) e, logo após, executa um laço que lê a porta serial a cada iteração até que algum valor seja encontrado (que deverá ocorrer após a IDE Arduino identificar a solicitação dos valores com o caractere “r”).

Figura 24. A função *setSpeed()*

```
def setSpeed(V1,V2):
    sv.V1 = str(V1)
    sv.V2 = str(V2)
    serial.write(b'%s %s %s' % ('s'.encode('utf-8'),sv.V1.encode('utf-8'),sv.V2.encode('utf-8')))
```

Fonte: Autor

A função *setSpeed()* é usada após os cálculos para imprimir os valores das velocidades desejadas. Para isso, é necessário utilizar a função *write* da biblioteca *PySerial* seguindo o padrão de referência estabelecido “s V1 V2”, sendo V1 e V2 as velocidades desejadas para a minibomba 1 e 2, respectivamente.

É importante ressaltar que só é possível escrever na porta serial em formato de bytes. Por isso, utilizamos a função nativa do Python chamada “*encode”*, que transforma a *string* em bytes (no nosso caso, em formato UTF-8)

3.8 O Software em Python

3.8.1 Introdução

Python é uma linguagem de programação que foi concebida no final de 1980 e sua implementação foi iniciada em dezembro de 1989 como um sucessor para a linguagem de programação ABC capaz de manipulação de exceção e interface gráfica. É muito utilizada para criar visualizações de dados e previsões.

Com Python, é possível encontrar uma grande (e crescente) comunidade. Isso é particularmente interessante pois é possível contar com uma vasta gama de especialistas e desenvolvedores que interagem entre si com o propósito de sempre buscar uma solução adequada e otimizada para problemas específicos utilizando a programação.

Além disso, também é possível encontrar uma grande variedade bibliotecas de ciência de dados (como por exemplo: *NumPy*, *SciPy*, *StatsModels*, *scikit-learn*, *pandas* etc.), que estão em crescimento exponencial. Restrições (em métodos de otimização / funções) existentes há algum tempo já não são mais problema atualmente, e a melhora dessas bibliotecas é constante e rápida, pois conta com inúmeros colaboradores pelo mundo todo.

Por fim, em relação a outras linguagens amplamente conhecidas para a ciência de dados (como MatLab, Stata, R) Python é muito mais rápido, além de possuir sintaxe amigável e grande rapidez em sua aprendizagem.

Para este projeto, foram utilizadas 3 bibliotecas essenciais que facilitaram e tornaram o projeto mais interativo e gráfico para o usuário, são elas: NumPy, Tkinter e Matplotlib, que serão descritas a seguir.

3.8.2 A biblioteca NumPy

NumPy é uma biblioteca Python usada para trabalhar com *arrays*, comumente chamados de listas no mundo da computação, que são basicamente matrizes de uma dimensão, mas ela também possui funções para trabalhar no domínio da álgebra linear, transformada de Fourier e matrizes multidimensionais. A biblioteca foi criada em 2005 por Travis Oliphant e é um projeto de código aberto que pode ser usado livremente. NumPy significa *Numerical Python*, que quer dizer Python numérico, e uma de suas

principais vantagens é fornecer um *array* próprio que é até 50x mais rápido que as listas tradicionais do Python.

Para este projeto, foi criada uma função chamada PID, que executa o cálculo da nova velocidade da bomba 1, enquanto a bomba 2 é mantida na mesma velocidade e pode ser alterada apenas pelo usuário para simular uma perturbação no sistema.

Figura 25. A função PID()

```
def PID():
    sv.pt[0] = float(sv.y2Values[-1])
    while not sv.stopExecution:
        if sv.killThread:
            break

        sv.yt[2] = sv.level
        sv.et[1] = sv.setpoint - sv.yt[2]

        sv.pt[1] = sv.pt[0]+sv.Kc*(sv.et[1] - sv.et[0])\
                   + (sv.Kc*sv.deltaT/sv.Ti)*sv.et[1]\
                   - (sv.Kc*sv.Td/sv.deltaT)*(sv.yt[2]-2*sv.yt[1]+sv.yt[0])

        if sv.pt[1] > 400:
            sv.pt[1] = 400
        elif sv.pt[1] < 80:
            sv.pt[1] = 80

        setSpeed(sv.pt[1], sv.V2)
        sv.yt[0] = sv.yt[1]
        sv.yt[1] = sv.yt[2]
        sv.et[0] = sv.et[1]
        sv.pt[0] = sv.pt[1]
```

Fonte: Autor

Os valores das variáveis e seus significados são descritos a seguir:

- sv.setpoint – Constante que representa o *setpoint* inserido pelo usuário na interface.
- sv.pt – Um *array* que armazena os valores de velocidade da bomba, sendo o primeiro valor (sv.pt[0]) a medida anterior e o segundo valor (sv.pt[1]) a medida atual.

- sv.yt - Um *array* que armazena os valores de altura do nível de água, sendo o primeiro valor (sv.yt[0]) a antepenúltima medida, o segundo valor (sv.yt[1]) a penúltima medida e o terceiro valor (sv.yt[2]) a medida atual.
- sv.et - Um *array* que armazena os valores do erro, ou seja, a diferença entre o *setpoint* e a altura atual, sendo o primeiro valor (sv.et[0]) a medida anterior e o segundo valor (sv.et[1]) a medida atual.
- sv.Kc – Constante de ganho proporcional inserida pelo usuário na interface.
- sv.deltaT – Constante que representa o tempo entre cada medição (neste projeto foi definido 1 segundo como o valor do intervalo).
- sv.Td – Constante que representa o valor do ganho derivativo inserido pelo usuário na interface.
- sv.Ti - Constante que representa o valor do ganho integral inserido pelo usuário na interface.

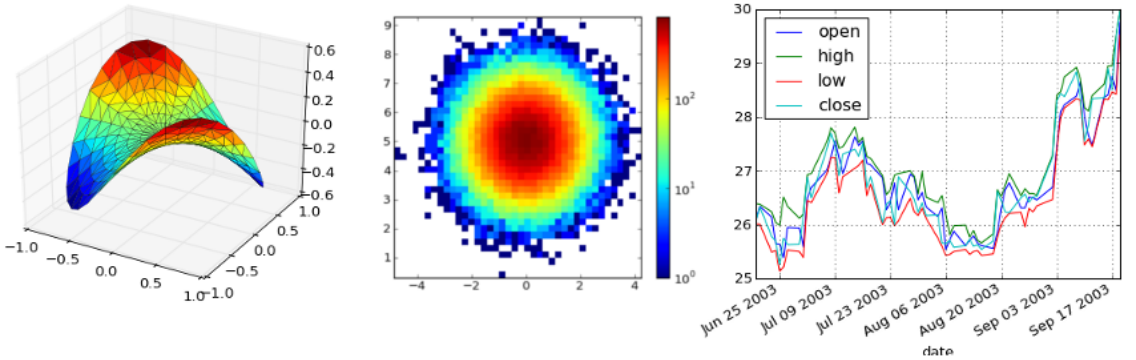
Portanto, resumidamente, a função implementada recebe os valores de velocidade, altura do nível de água, constantes K_c, τ_D e τ_I, o valor do setpoint, calcula o erro e com ele encontra o novo valor de velocidade e, após isso, altera o valor da vazão da bomba 1 automaticamente para o valor calculado.

3.8.3 A biblioteca Matplotlib

Matplotlib é uma biblioteca abrangente para criar visualizações estáticas, animadas e interativas em Python.

Originalmente criada pelo americano John D. Hunter, a biblioteca hoje possui uma comunidade ativa atuando em seu desenvolvimento e é distribuída sob uma licença BSD (*Berkeley Software Distribution*), uma licença de código aberto, assim como a GNU (*General Public License*), porém com menos restrições autorais, colocando-a relativamente próxima do domínio público.

Figura 26. Exemplos de gráficos feitos com a biblioteca Matplotlib



Fonte: *https://itom.bitbucket.io/latest/docs/_images/matplotlib_intro.png*

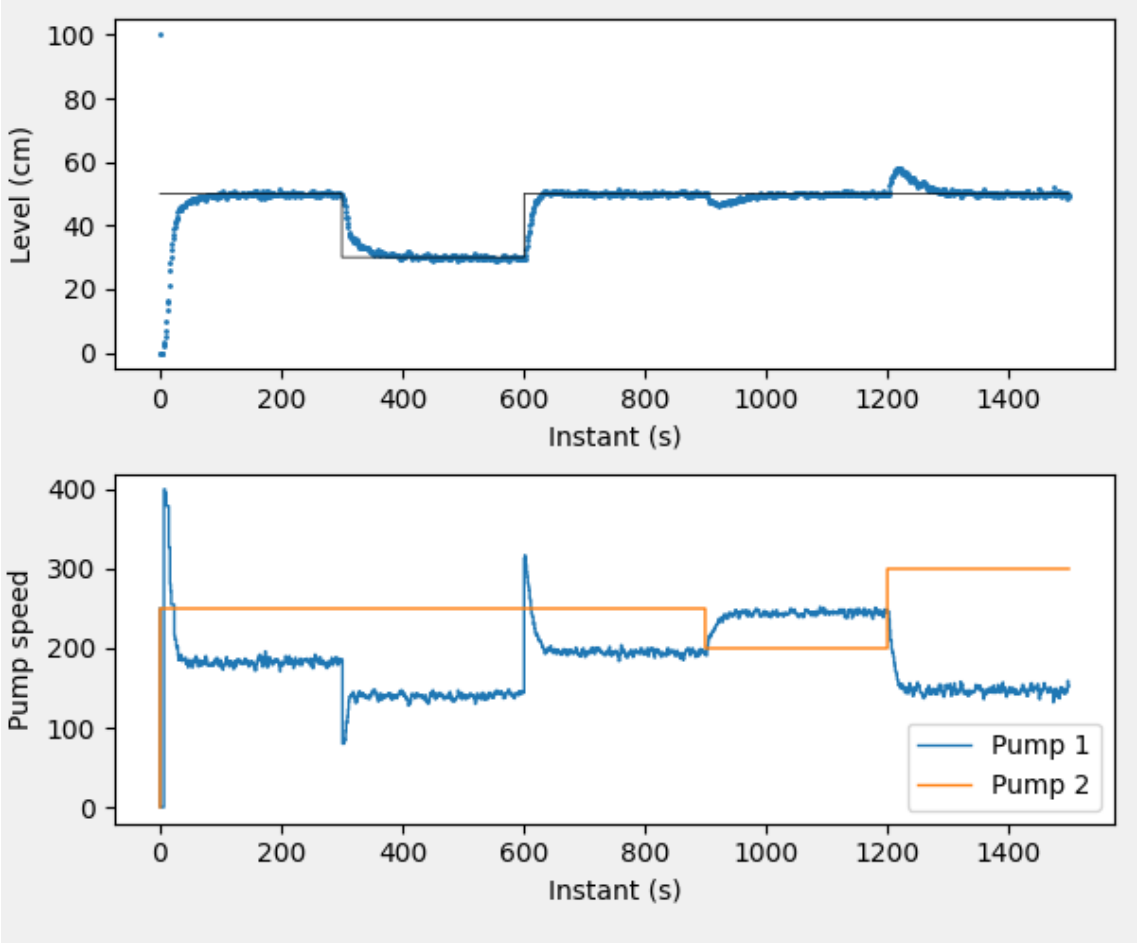
No caso deste projeto, são plotados dois gráficos em tempo de aquisição, ou seja, enquanto os dados estão sendo adquiridos e alterados, os gráficos dos valores são exibidos.

O primeiro gráfico exibe os valores do nível de água em função do tempo. Além disso, uma linha com o *setpoint* é mantida sobre ele para melhor visualização da eficiência do controlador. Para fazê-lo, foram utilizadas as seguintes funções:

- plt.plot(x, y) – Função principal da biblioteca, ela exibe o ponto especificado no gráfico. Neste caso, x representa o eixo das abscissas e possui uma lista de valores que contém o tempo decorrido. Enquanto y representa o eixo das coordenadas e possui uma lista de valores que contempla todos os valores da altura do nível de água medidos segundo a segundo.
- plt.xlim(max, min) – Função que tem o objetivo de delimitar a área de exibição do eixo das abscissas. Para a aquisição em tempo real os valores definidos foram 0 e 100 para melhor visualização. Entretanto, ao fim da medição, o gráfico completo é exibido (com todos os valores de x).
- plt.ylim(max, min) - Função que tem o objetivo de delimitar a área de exibição do eixo das coordenadas. Os valores definidos foram 0 e 100 pois eles representam a altura em centímetros do tanque, não fazendo sentido serem alterados.
- Plt.subplot() – Função que tem o objetivo de definir mais de um gráfico dentro de um espaço determinado, além da posição de cada um (linha, coluna etc)

Já o segundo gráfico exibe as velocidades da bomba 1 e 2 em função do tempo. Para ele, foram utilizadas as mesmas funções, sendo “y” uma lista com todas as velocidades para cada instante e o limite de exibição dos eixos das coordenadas foram definidos como 0 e 400, pois esses são os valores mínimos e máximos de velocidade possíveis para as bombas.

Figura 27. Gráfico gerado pela biblioteca Matplotlib



Fonte: Autor

Além disso, algumas outras funções de estilização do gráfico foram utilizadas:

- plt.xlabel – Define o texto a ser exibido no eixo das coordenadas.
- plt.ylabel – Define o texto a ser exibido no eixo das abscissas.
- plt.legend – Define as legendas a serem exibidas com diferenciação de cores.
- plt.subplots_adjust – Define a posição e dimensões de cada gráfico.

A figura 29 exibe todo o código desenvolvido para a exibição dos gráficos em tempo real:

Figura 28. A função *plotUpdate()*

```
def plotUpdate(y1Values,y2Values,y3Values,xValues,setpoint,all_setpoints):

    if xValues[-1:] != []:
        x = int(xValues[-1])

        plt.subplots_adjust(left=0.1, bottom=0.129, right=0.98, top=0.98, wspace=0.2, hspace=0.3)

        plt.subplot(2,1,1)
        plt.cla()
        try:
            plt.plot(xValues, y1Values,marker='o',markersize=1,linewidth=0)
        except:
            pass
        plt.ylim(ymin=0,ymax=100)
        plt.yticks(numpy.arange(0,110,10))
        plt.ylabel('Level (cm)')
        plt.xlabel('Instant (s)')
        if x < 100:
            plt.xlim(xmax=100, xmin=0)
            all_setpoints = numpy.resize(all_setpoints,101)
            all_setpoints[len(xValues):101] = setpoint
            plt.plot(numpy.arange(0,101),all_setpoints,linewidth=0.5, color='black', drawstyle='steps-pre')
        else:
            try:
                plt.xlim(xmax=x, xmin=x - 100)
                plt.plot(xValues, all_setpoints, linewidth=0.5, color='black', drawstyle='steps-pre')
            except:
                pass

        plt.subplot(2,1,2)
        plt.cla()
        try:
            plt.plot(xValues, y2Values, xValues, y3Values, drawstyle='steps-pre',linewidth=1)
        except:
            pass
        plt.ylim(ymin=80,ymax=400)
        plt.yticks(numpy.arange(80, 401, 40))
        plt.ylabel('Pump speed')
        plt.xlabel('Instant (s)')
        plt.legend(['Pump 1','Pump 2'])
        if x < 100:
            plt.xlim(xmax=100, xmin=0)
        else:
            plt.xlim(xmax=x,xmin=x-100)
```

Fonte: Autor

3.8.4 A biblioteca Tkinter

GUI significa "*Graphical User Interface*" (Interface Gráfica do Usuário), que consiste em um modelo de interface para o usuário que permite a interação com o hardware para execução de funcionalidades através de elementos gráficos como botões, caixas de texto etc. Para a linguagem Python, existem diversos *frameworks* e bibliotecas específicas que permitem a criação dessas interfaces.

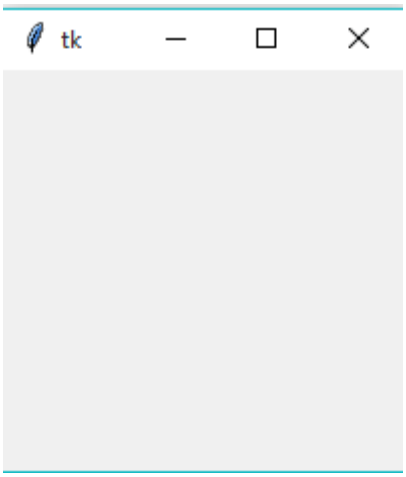
O pacote tkinter é a biblioteca padrão do Python para criação de uma GUI. Ele está disponível na maioria dos sistemas operacionais. Essa biblioteca nos permite desenvolver a nossa própria interface gráfica customizada de acordo com as necessidades do usuário e preferências de estilização, experiências interativas etc. O principal motivo

para utilizá-la é a sua facilidade e recursos disponíveis, além de ser nativa da linguagem Python, precisando apenas importá-la e assim já estará pronta para uso.
Uma GUI aborda muitos conceitos, mas os principais que precisamos entender para este projeto são:

- Container – Tem como objetivo realmente atuar como um container, ou seja, tem a função de armazenamento. Nesse caso, o que é armazenado são os widgets.
- Widget – É um componente da interface, que pode ser um botão, uma barra, uma imagem, uma caixa de texto etc.
- *Event Handler* – Funcionam como tratadores de uma função. Por exemplo, ao clicarmos em um botão para executar uma ação, uma rotina é executada, sendo ela chamada de *event handler*.
- *Event Loop* – O *event loop* funciona de maneira cíclica. Checa constantemente se outro evento foi executado. Caso positivo, ele irá executar a rotina correspondente.

Para inicializar a interface gráfica, após importar a biblioteca, deve-se instanciar a classe Tk() através da variável *root*, que será o "motor" da aplicação. Após isso, para dar início à execução da GUI, executa-se a função *root.mainloop()*, que exibirá a interface até que sua interrupção seja forçada.

Figura 29. Exemplo de uma interface Tkinter vazia



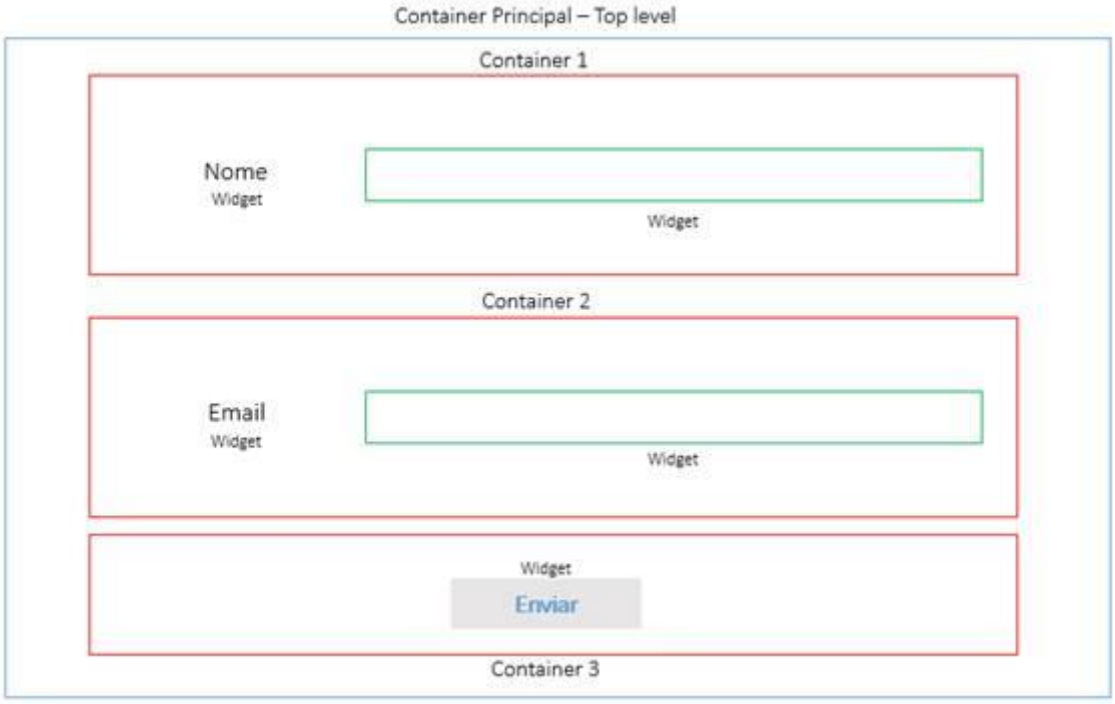
Fonte: Autor

Para poder trabalhar com widgets é necessário entender o conceito de container, que é uma estrutura onde os widgets são colocados. Por questão de organização e para

sua correta criação, define-se os containers, e dentro de cada container, um ou mais widgets que o compõe.

A Figura 31 exibe de forma visual os conceitos de container e widget.

Figura 30. Containers e *Widgets* de uma GUI



Fonte: *https://www.devmedia.com.br/tkinter-interfaces-graficas-em-python/33956*

A biblioteca Tkinter oferece três formas de trabalharmos com geometria e posicionamento: *Pack, Grid* e *Place.* Entretanto, foi abordado somente o método *Place,* que fornece possibilidade de posicionarmos nossos elementos de forma absoluta, ou seja, definindo valores fixos para o posicionamento vertical e horizontal.

Por exemplo, para posicionar um botão dentro da interface, deve-se uma variável *button* e a posicionar utilizando a função *button.place(x, y)*, sendo x a posição no eixo horizontal e y a posição no eixo vertical. Além disso, é possível estilizar e dimensionar todos os *widgets* criados.

As configurações de estilo e dimensionamento utilizadas foram:

- *Width*– Largura do *widget*;
- *Height*– Altura do widget;

- *Text*– Texto a ser exibido no *widget*;
- *Font*– Família da fonte do texto;
- *Fg*– Cor do texto do *widget*;
- *Bg*– Cor de fundo do *widget*;

Tendo esse conhecimento e utilizando o mesmo conceito para caixas de texto, imagens e gráficos, podemos criar uma interface gráfica completa posicionando e dimensionando os elementos criados. A figura 32 mostra o exemplo de alguns botões criados para este projeto. Nela, podemos ver a criação das variáveis e o seu posicionamento através do eixo horizontal e vertical. Além disso, também é possível definir um comando, ou seja, uma função dentro do código que será executada ao se clicar no respectivo botão.

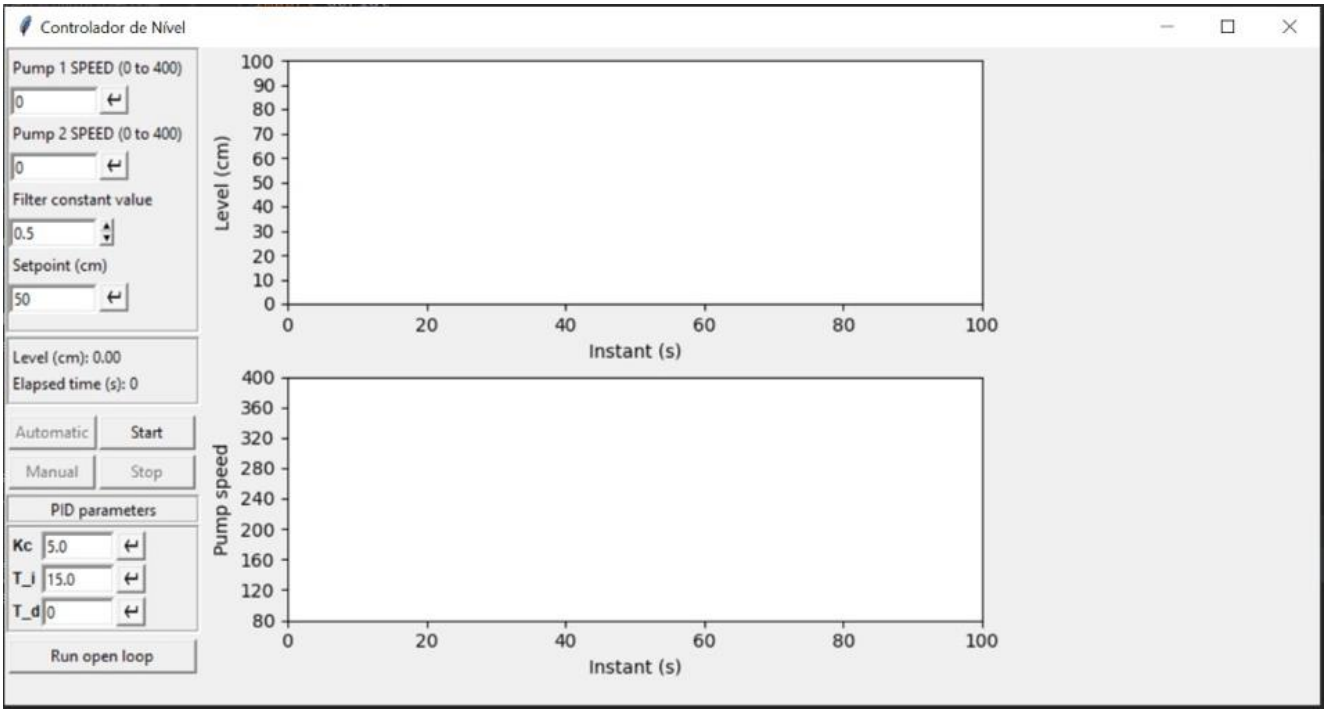
Figura 31. Criação de botões através da biblioteca Tkinter.

```
button3_1 = Button(sv.root, image=buttonImage2, command=clickButton2)
button3_1.place(x=72, y=130)
button3_2 = Button(sv.root, image=buttonImage3, command=clickButton3)
button3_2.place(x=72, y=140)
```

Fonte: Autor

A figura 33 exibe a interface gráfica do projeto criada e devidamente dimensionada e estilizada, com botões, caixas de texto, gráficos e imagens. Em seguida será especificado a funcionalidade de cada um desses *widgets.*

Figura 32. A interface gráfica do projeto.



Fonte: Autor

As funcionalidades da interface serão explicadas a seguir:

- Caixa de texto "*Pump* 1 *SPEED*" – Entrada de dados usada pelo usuário para definir o valor da velocidade da bomba 1 manualmente. O botão ao lado da caixa de texto está vinculado à função "*setSpeed()*" do programa em Python.
- Caixa de texto "*Pump* 2 *SPEED"* – Mesma funcionalidade da primeira caixa de texto, porém é usado para definir a velocidade da bomba 2 manualmente.
- Caixa de texto "*Filter constant value*" – Entrada de dados usada pelo usuário para definir o valor da constante de filtragem usada para obtenção dos dados. Tem a função de filtrar o valor medido pelo sensor a fim de diminuir artificialmente possíveis ruídos e evitar variações bruscas na medição do nível. Ela é utilizada na função *filteredValues()* do código em Python que retorna o valor filtrado da medição do sensor e depois tal valor é transformado em centímetros. Para isso, utiliza-se essa constante como peso para determinar quanto o último valor medido deve ser considerado na nova medição.

Figura 33. A função *filteredValue()*

```
def filteredValue(xf0,currentValue,alpha):
    while True:
        if len(currentValue) == 3:
            xf = alpha*xf0 + (1-alpha)*float(currentValue[2])
            return xf
        else:
            pass
```

Fonte: Autor

- Caixa de texto "*setpoint"* – Entrada de dados usada pelo usuário para definir o *setpoint* do sistema, ou seja, o valor do nível de água que deve ser mantido pelo controlador através do PID. Só é útil quando o programa está em modo automático, que será devidamente explicado nesses tópicos. O botão ao lado desta caixa de texto está vinculado à variável usada pela função PID para definir o valor do erro.
- "*Level (cm)"* – Valor que é atualizado automaticamente a cada segundo de acordo com a leitura filtrada e transformada do sensor de pressão. Representa o valor em centímetros da altura de água.
- "*Elapsed time (s)*" – Valor atualizado automaticamente que representa o tempo, em segundos, decorrido desde o início da corrida (usuário apertou o botão "*start*").
- Botão *"Automatic"* – Botão que está vinculado com a função PID e inicia o processo de controle automático do nível de água através do setpoint definido pelo usuário. Ao ser clicado, este botão desabilita automaticamente a caixa de texto "*Pump 1 SPEED*", que permite o controle manual da vazão da bomba 1, pois a velocidade dessa bomba será definida dinamicamente pelo controlador, permitindo ao usuário apenas o controle manual do *setpoint* e da bomba 2 para simular possíveis perturbações. Além disso, este botão só ficará habilitado para uso depois que o botão "*Start*" estiver pressionado.
- Botão "*Start"* – Botão que tem a função de iniciar a contagem de tempo, medição do nível e plotagem dos gráficos.
- Botão "*Manual"* – Botão que tem a função de desativar o PID, ou seja, ele desabilita o controle automático do nível pelo controlador e permite o usuário definir a velocidade de ambas as bombas manualmente.
- Botão "Stop" – Botão que tem a função de encerrar a corrida. Encerra a plotagem de gráficos, a medição do nível e congela a contagem de tempo.

- Caixa de texto “Kc” – Permite o usuário inserir o valor da constante do ganho proporcional na função PID. O valor só será atribuído à respectiva variável dentro do programa após o usuário pressionar o botão ao lado da caixa de texto.
- Caixa de texto “T_i” – Permite o usuário inserir o valor da constante do ganho integral na função PID. O valor só será atribuído à respectiva variável dentro do programa após o usuário pressionar o botão ao lado da caixa de texto.
- Caixa de texto “T_d” – Permite o usuário inserir o valor da constante do ganho derivativo na função PID. O valor só será atribuído à respectiva variável dentro do programa após o usuário pressionar o botão ao lado da caixa de texto.
- Botão “*Run Open Loop”* – Botão que tem a função de iniciar uma corrida com valores predeterminados das velocidades das bombas que variam durante 15 minutos. Após o fim da corrida, todos os valores das velocidades das bombas e nível de água em função do tempo são salvos em um arquivo de texto e foram utilizados para a determinação da função de transferência do sistema.

CAPÍTULO 4

4 RESULTADOS E DISCUSSÕES

4.1 Calibração do Transmissor de Nível piezoresistivos da série SS302+

O transmissor de pressão diferencial piezoresistivo da série SS302+ foi utilizado aqui para medir e transmitir o nível de líquido dentro do tanque. Este transmissor possui maior estabilidade e menor nível de interferência se comparados à série SS302. Com um sistema eletrônico de alta qualidade montado dentro do sensor, ele fornece sinal de tensão de 4-20mA com baixo nível ruído. O objetivo aqui é obter uma curva de calibração por regressão de dados experimentais e validar estatisticamente a curva de calibração para um nível de confiança estatístico pré-estabelecido.

Após a devida conexão dos fios de entrada e saída, o sensor passa a transmitir um sinal de 4-20 mA, que é convertido para 0V-5V. A saída do sensor foi conectada num dos pinos de entrada analógica do Arduino UNO. Para a leitura dos valores recebidos, o pino conectado foi definido como analógico e foi criada uma função que lê o valor transmitido a esse pino e os filtra antes de enviá-lo à interface do Python através da comunicação serial. Tal filtragem é importante para diminuir ruídos e possíveis valores *outliers*. Basicamente, a função funciona criando um laço iterativo que é executado 10 vezes e recebe 10 valores de pressão do sensor durante esse ciclo. Após as iterações, é eliminado o menor valor, o maior valor e a média entre os 8 valores restantes é retornada.

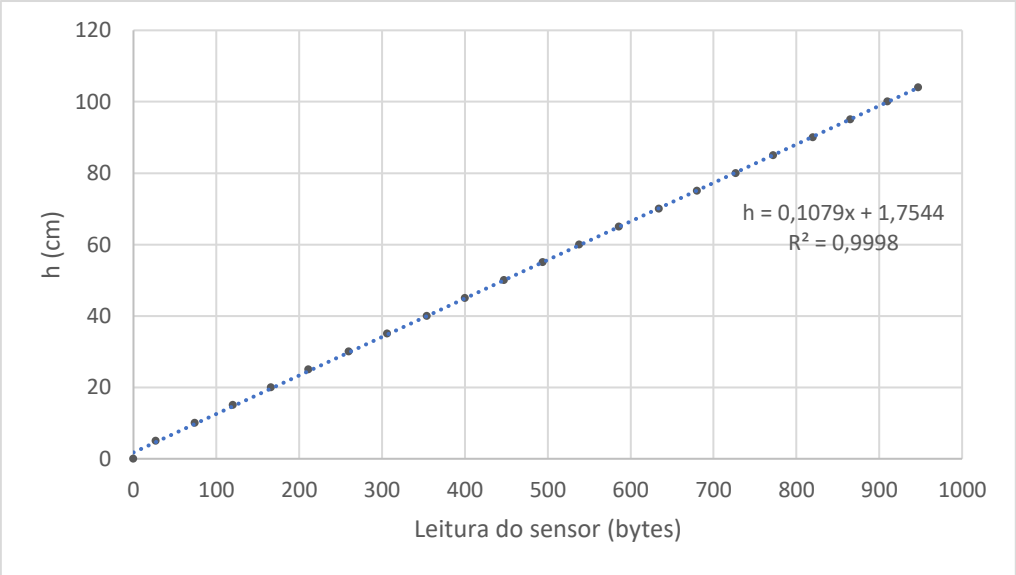
Os valores lidos diretamente na IDE do Arduino são números (bytes) que variam de 0 a 1024 (2^{10}, placa de 10 bytes) e são proporcionais ao nível de água dentro do tubo. Portanto, é necessária a realização de uma calibração do sistema, isto é, determinar a dependência entre sinal medido pelo sensor e transmitido ao Arduino e o nível de líquido dentro do tanque. Para isso, diferentes valores de pressão foram obtidos para diferentes valores do nível de água medidos diretamente no tanque através de uma fita métrica. O dados são mostrados na Tabela XX a seguir.

Tabela 3. Dados experimentais de calibração do sensor de nível.

Leitura sensor (bytes)	Nível (cm)
0	0
27	5
74	10
120	15
166	20
211	25
260	30
306	35
354	40
400	45
447	50
494	55
538	60
586	65
634	70
680	75
727	80
772	85
820	90
865	95
910	100
947	104

Uma regressão linear entre os valores lidos pelo sensor e o nível medido experimentalmente foi realizada e o resultado é mostrado na figura XXI a seguir.

Figura 34. Curva de calibração do sensor de nível.



Fonte: Autor

Observa-se que o sensor/transmissor tem um comportamento estritamente linear entre o sinal medido e o nível do tanque, com baixo nível de ruídos. Esta equação de calibração foi implementada no programa fonte em Python para retornar o nível de líquido dentro do tanque usando leituras enviadas pelo sensor/transmissor.

4.2 Estimação dos Parâmetros do Modelo Linear da Planta Experimental

Para estimação dos parâmetros τ_1, τ_2,K_1, K_2, θ_1 e θ_2, duas corridas em malha aberta foram realizadas. Na primeira corrida um sinal PWM constituído de uma sequência aleatória de degraus com amplitude entre [200 e 400] foi enviado para a minibomba 1, mantendo-se a minibomba 2 desligada. O nível foi medido e adquirido na forma de um arquivo de dados no formato de texto. O sinal PWM enviado para a minibomba 1 e o nível registrado formam o par entrada-saída do sistema, que depois de devidamente transformado em variável desvio, foi usado para a estimação dos parâmetros τ_1, K_1 e θ_1.

Analogamente, o mesmo procedimento foi feito para a minibomba 2, mantendo-se agora a minibomba 1 desligada e enviando um sinal PWM constituído de uma sequência aleatória de degraus com amplitude entre [200 e 400] para a minibomba 2. O nível foi medido e adquirido na forma de um arquivo de dados no formato de texto. O sinal PWM enviado para a minibomba 2 e o nível registrado formam o par entrada-saída do sistema, que depois de devidamente transformado em variável desvio, foi usado para a estimação dos parâmetros τ_2, K_2 e θ_2.

De posse dos dados experimentais em malha aberta, os parâmetros do sistema foram estimados usando a função "*tfest*" do *toolbox* de identificação de sistemas do *software* Matlab versão R2019a. Esta função estima uma função de transferência de tempo contínuo, *sys*, usando dados no domínio do tempo ou da frequência. Sua sintaxe é:

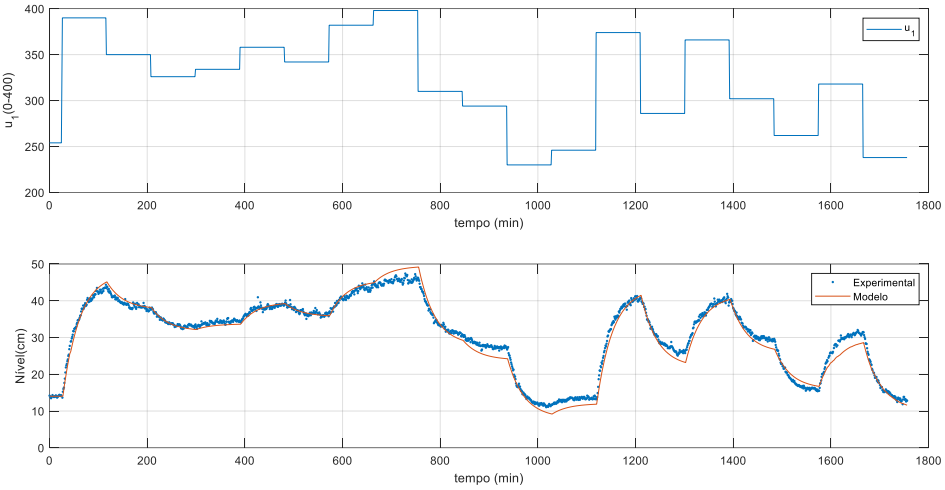
sys = tfest(data,np,nz,iodelay)

Em que "data" contém uma forma adequada dos dados temporais obtidos experimentalmente, "np" é o número de polos da função de transferência, "nz"é o número de zeros da função de transferência e "iodelay" é o atraso de transporte para o par de entrada/saída. Por inspeção das funções de transferência (16)-(17), nota-se que np = 1, nz = 0. Por inspeção dos dados experimentais também foi possível estimar $\theta_1 = \theta_2 = 2\ s$. Então, $G_1(s)$ e $G_2(s)$ foram determinados como:

G1 = tfest(data_1,1,0,2) e G2 = tfest(data_2,1,0,2)

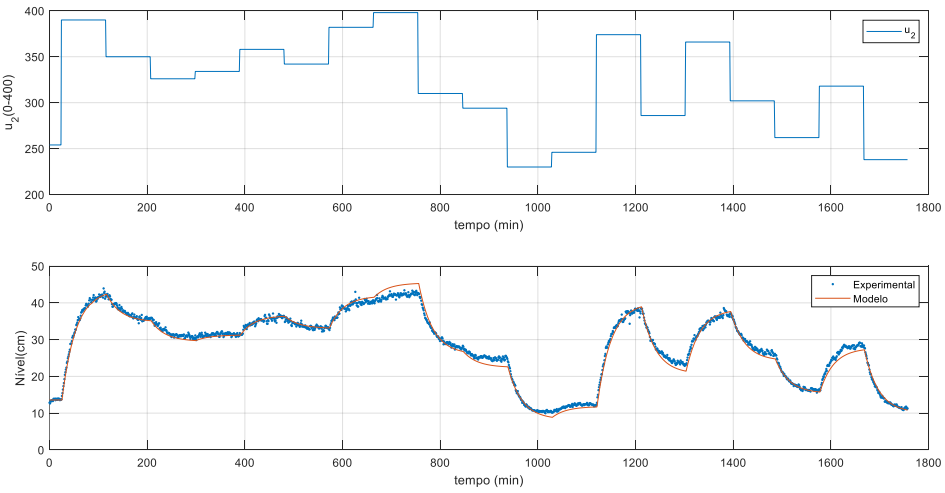
Os resultados de h(t) versus $u_1(t)$ são mostrados na figura XXII e os resultados de h(t) versus $u_2(t)$ são mostrados na figura XXIII.

Figura 35. Dados de malha aberta: h(t) versus u1(t).



Fonte: Autor

Figura 36. Dados de malha aberta: h(t) versus u2(t).



Fonte: Autor

As funções de transferência encontradas foram:

$$G_1(s) = \frac{H_1'(s)}{u_1'(s)} = \frac{0{,}2462e^{-2s}}{66{,}31s + 1}$$

$$G_2(s) = \frac{H_2'(s)}{u_2'(s)} = \frac{0{,}2211e^{-2s}}{56{,}59s + 1}$$

4.3 Validação estatística do modelo matemático

Há diversas formas de validar um modelo usando estatísticas. Escolheu-se aqui a metodologia que se baseia no princípio de que se o modelo ajustado consegue capturar o comportamento do sistema, então, no erro de modelagem estarão presentes somente contribuições devido ao ruído dos dados experimentais. Como esses ruídos são aleatórios por natureza, não são passíveis de previsão. Então, nenhum modelo conseguirá capturá-los. O melhor que se pode obter, então, é um ruído branco para o erro de modelagem. Assim, pode-se colocar a validação do modelo com um teste de hipótese, isto é, testa-se se o erro de modelagem é um ruído branco com média zero e variância finita. Segundo Montgomery e Runger (2003), se $x_1, x_2, \cdots, x_n$ são amostras aleatórias de uma distribuição normal com média μ e variância σ^2 desconhecida (s^2 variância amostral), então a estatística "T" a seguir é uma variável aleatória:

$$T = \frac{\bar{X} - \mu_0}{s/\sqrt{n}}$$

com distribuição "*t-student*" com (n-1) graus de liberdade. Logo, pode-se testar se a média amostral $\bar{x}$ é estatisticamente igual a média verdadeira μ_0. Então, usa-se o teste:

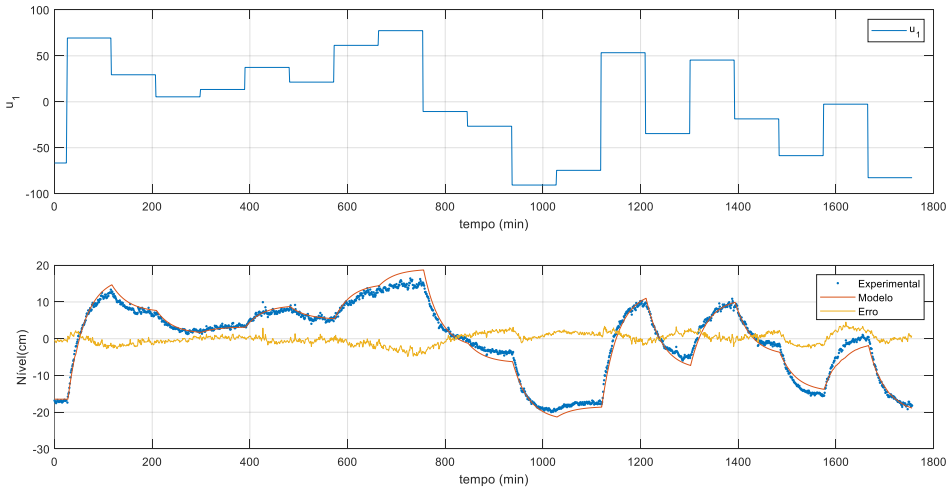
$$H_0: \mu = \mu_0 \ (hipótese\ nula)$$
$$H_1: \mu \neq \mu_0 \ (hipótese\ alternativa)$$

Se a hipótese nula é verdadeira (H_0), então T tem uma distribuição "*t-student*" com (n-1) graus de liberdade e a média amostral é estatisticamente igual a média verdadeira, o que equivale a dizer que o erro de modelagem é um ruído branco. Se a hipótese nula for rejeitada, então a hipótese alternativa é verdadeira e a média amostral não é igual à média verdadeira. Então, o erro de modelagem não é um ruído branco o que equivale a dizer que o modelo é tendencioso (existem termos não modelados presentes no erro de modelagem). Se $|T| \geq t_{\alpha/2,n-1}$ então $H_0: \mu = \mu_0$ é rejeitada e se $|T| \leq$

$t_{\alpha/2,n-1}$ então $H_0: \mu = \mu_0$ é aceita. "α" é nível de significância do teste, tipicamente 1% a 5%.

Através dos dados experimentais e dos resultados preditos pelo modelo ajustado, foi possível calcular os erros de modelagem, que são mostrados nas figura XXIV e na figura XXV. Nestas figuras os dados experimentais e simulados foram deslocados da média dos dados experimentais para permitir colocá-los de maneira adequada na mesma figura que os erros de modelagem.

Figura 37. Erro de modelagem do modelo G1 (s).



Fonte: Autor

Validação de $G_1(s)$:

Média do erro de modelagem : $\bar{x} = 1{,}0897.10^{-14}\ cm$

Variância do erro modelagem : $s^2 = 2.2942\ cm^2$

Média a ser testada : $\mu_0 = 0\ cm$

n (número de dados) : 1717

Estatística T : $T = \frac{\bar{X}-\mu_0}{S/\sqrt{n}} = \frac{1{,}0897.10^{-14}-0}{\sqrt{2.2942/1717}} = 2{,}9811.10^{-13}$

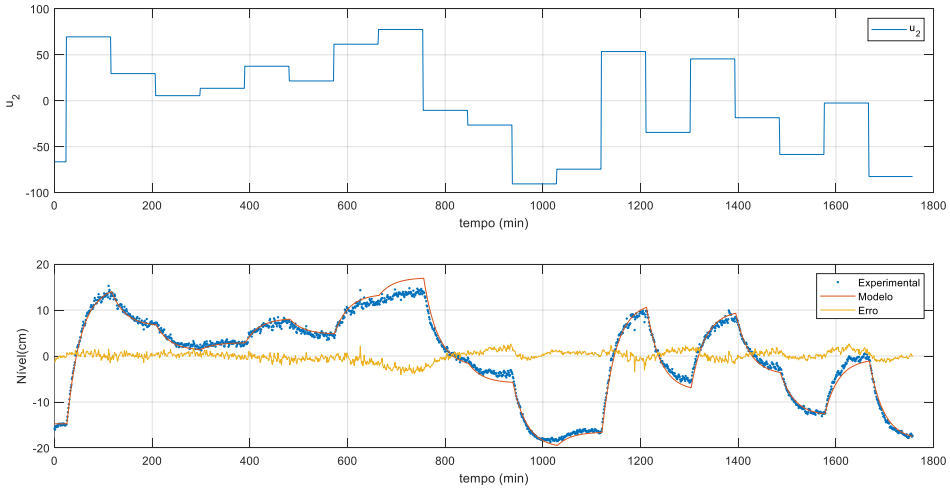
Nível de significância (□) : $\alpha = 5\%$

$t_{\alpha/2,n-1}$ (Tabela t-Student) : $t_{0{,}025,\infty} = 1{,}96$

Como $|T| \leq t_{\alpha/2,n-1}$, então $H_0: \mu = \mu_0$ $(hipótese\ nula)$ é aceita e a média do erro de modelagem é estatisticamente zero. Portanto, o erro de modelagem é um ruído branco e o modelo $G_1(s) = \frac{H_1'(s)}{u_1'(s)} = \frac{0{,}2462e^{-2s}}{66{,}31s+1}$ é não tendencioso.

Figura XXV.

Figura 38. Erro de modelagem do modelo G2 (s).



Fonte: Autor

Validação de $G_2(s)$:

Média do erro de modelagem : $\bar{x} = 4{,}9573.10^{-14}\ cm$

Variância do erro modelagem : $s^2 = 1{,}1298\ cm^2$

Média a ser testada : $\mu_0 = 0\ cm$

n (número de dados) : 1732

Estatística T : $T = \frac{\bar{X}-\mu_0}{S/\sqrt{n}} = \frac{4{,}9573.10^{-14}-0}{\sqrt{1{,}1298/1732}} = 1{,}9410.10^{-12}$

Nível de significância (□) : $\alpha = 5\%$

$t_{\alpha/2,n-1}$ (Tabela t-Student) : $t_{0{,}025,\infty} = 1{,}96$

Como $|T| \leq t_{\alpha/2,n-1}$, então $H_0: \mu = \mu_0$ $(hipótese\ nula)$ é aceita e a média do erro de modelagem é estatisticamente zero. Portanto, o erro de modelagem é um ruído branco e o modelo $G_2(s) = \frac{H_2'(s)}{u_2'(s)} = \frac{0{,}2211e^{-2s}}{56{,}59s+1}$ é não tendencioso.

4.4 Os Ensaios

Para o presente projeto, foram feitos ensaios de 25 minutos divididos em 5 etapas, ou *steps*, utilizando os parâmetros PI dos métodos Ziegler-Nichols, ITAE, Cohen-Coon e também foi feito um ensaio utilizando parâmetros próprios através de observações empíricas do sistema.

Vale ressaltar que para este trabalho o ganho derivativo (τ_D) não foi utilizado pois apresentou grandes inconsistências devido aos ruídos inerentes do sistema.

O esquema utilizado foi:

De 0 a 5 minutos: *setpoint* mantido em 50 centímetros e velocidade da bomba 2 mantida em 250.

De 5 a 10 minutos: *setpoint* reduzido para 30 centímetros e velocidade da bomba 2 mantida em 250.

De 10 a 15 minutos: *setpoint* retornado para 50 centímetros e velocidade da bomba 2 mantida em 250.

De 15 a 20 minutos: *setpoint* mantido em 50 centímetros e velocidade da bomba 2 reduzida para 200.

De 20 a 25 minutos: *setpoint* mantido em 50 centímetros e velocidade da bomba 2 aumentada para 300.

Para a comparação quantitativa do melhor resultado para o processo em questão foram analisados 2 valores: tempo de assentamento (t) e desvio padrão (DP) após estabilização.

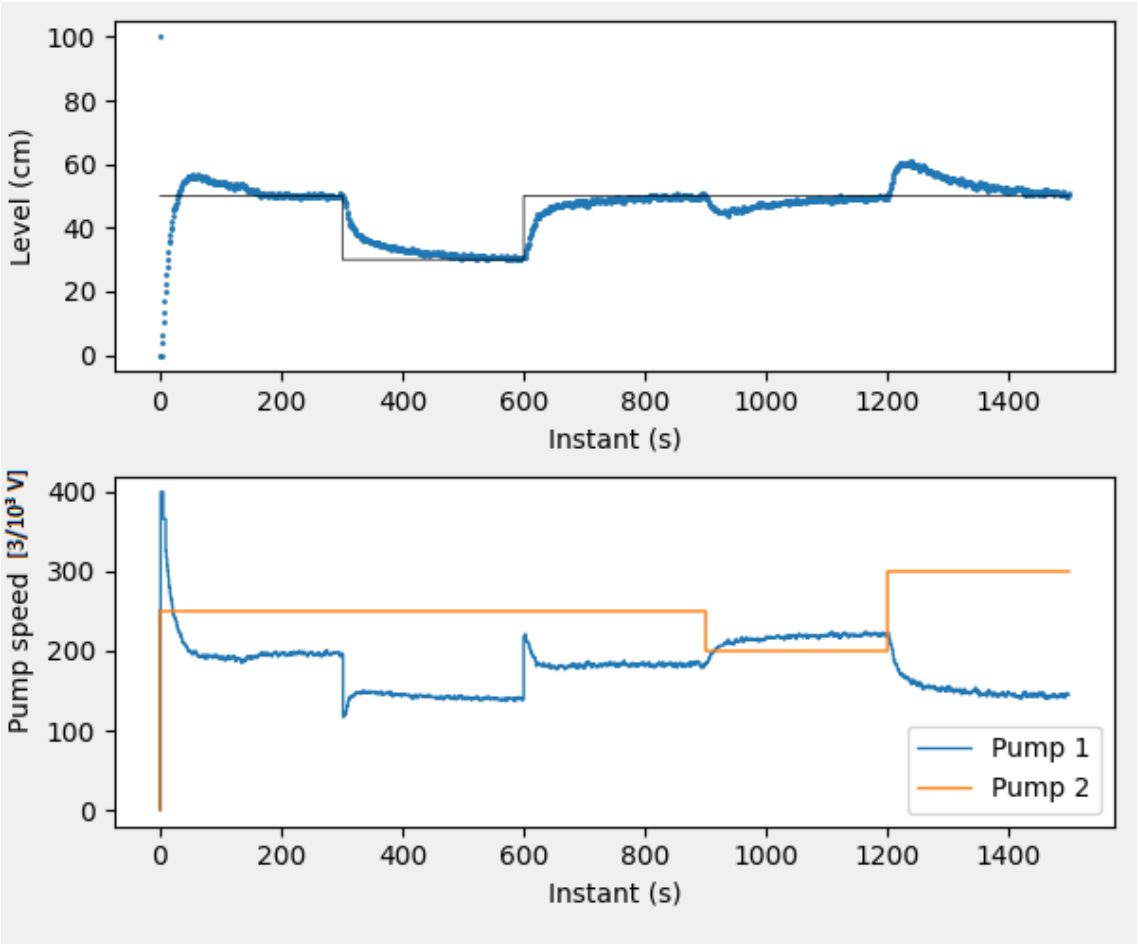
O tempo de assentamento (t) foi calculado para cada uma das cinco etapas descritas acima e representa o tempo necessário para que a altura do nível de água se estabilizasse no valor do *setpoint* ± 1,5 cm até o fim do *step*.

O desvio padrão (DP) calculado foi a medida estatística desvio padrão amostral dos valores do nível de água logo após o controlador atingir o tempo de assentamento até o fim do *step*.

4.5 Resultados do método Cohen-Coon

Para o ensaio do método Cohen-Coon, foram usados os valores 3,94 e 66,31 para os ganhos proporcional (K_c) e integral (τ_i), respectivamente. Os resultados e gráfico são mostrados abaixo.

Figura 39. Resultados do método Cohen-Coon



Fonte: Autor

Pode-se observar que no primeiro *step*, o nível ultrapassou o *setpoint* devido à alta velocidade aplicada pelo controlador na bomba 1 e em seguida foi corrigido automaticamente pelo processo PID.

Já nas etapas seguintes, percebe-se que o nível de água foi corrigido de maneira menos abrupta e, portanto, não oscilava sobre o *setpoint*. Esse grande gradiente de velocidade na primeira etapa acontece, pois, ao iniciar o processo PID o nível se encontrava em 0, o que gerou um valor de erro muito grande e, consequentemente, produziu um valor de variação de velocidade muito brusco.

Tabela 4. Resultados do método Cohen-Coon

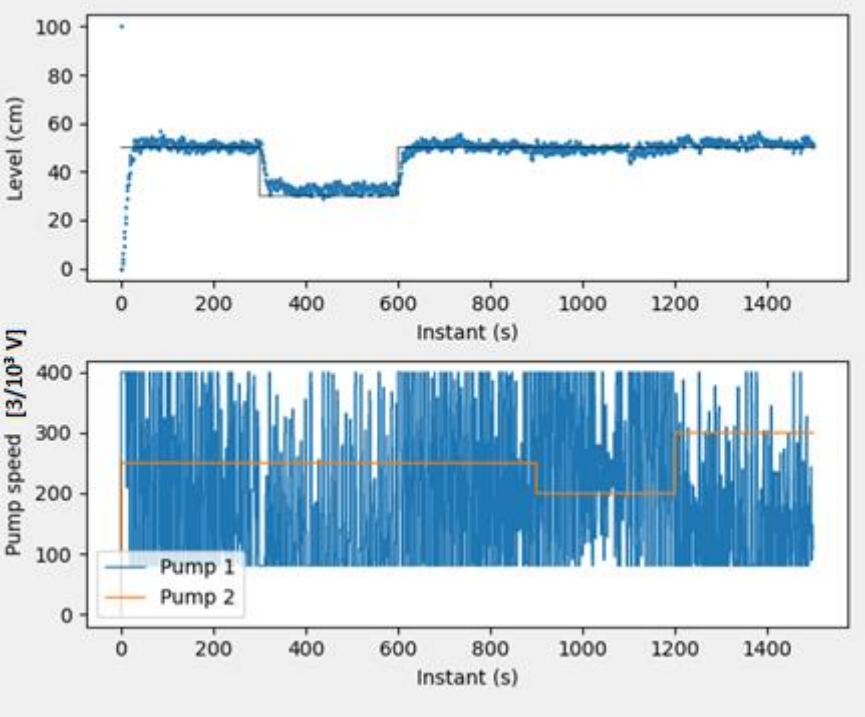
	Step 1	Step 2	Step 3	Step 4	Step 5
t (s)	161,20	230,30	166,80	229,80	269,70
Desvio Padrão (DE)	0,4332	0,3050	0,4867	0,3553	0,4467

Podemos perceber, também, que, muito provavelmente devido à abrupta mudança de velocidade no primeiro *step*, houve um menor tempo de assentamento. Porém, em contrapartida, o desvio padrão apresentado foi relativamente alto se comparado aos 2 menores desvios.

4.6 Resultados do método Ziegler-Nichols

Para o ensaio do método Ziegler-Nichols, foram usados os valores 121,20 e 6,67 para os ganhos proporcional (K_c) e integral (τ_i), respectivamente. Os resultados e gráfico são mostrados abaixo.

Figura 40. Resultados do método Ziegler-Nichols



Fonte: Autor

Já nesse método, podemos perceber um ganho proporcional muito maior, gerando variações muitos bruscas na velocidade da bomba 1, o que é bem explicitado no gráfico de baixo, em que a vazão da bomba está sendo alterada para o máximo e mínimo a todo instante. Apesar de ter atingido o *setpoint* de forma bem rápida, somente pela leitura visual do gráfico podemos perceber que o nível não se manteve muito constante e próximo ao *setpoint* desejado.

Tabela 5. Resultados do método Cohen-Coon

	Step 1	Step 2	Step 3	Step 4	Step 5
t (s)	Não atingiu	Não atingiu	298,10	296,60	298,30
Desvio Padrão (DE)	-	-	1,5537	0,8684	0,3952

Pelas análises quantitativas percebe-se que o método de Ziegler-Nichols não atingiu o assentamento ideal estipulado para este projeto (*setpoint* ± 1,5 cm) nas etapas 1 e 2 e apresentou grande tempo de assentamento nas etapas seguintes.

4.7 Resultados do método ITAE

Para o ensaio do método ITAE, foram usados os valores 58,80 e 64,69 para os ganhos proporcional (K_c) e integral (τ_i), respectivamente. Os resultados e gráfico são mostrados abaixo.

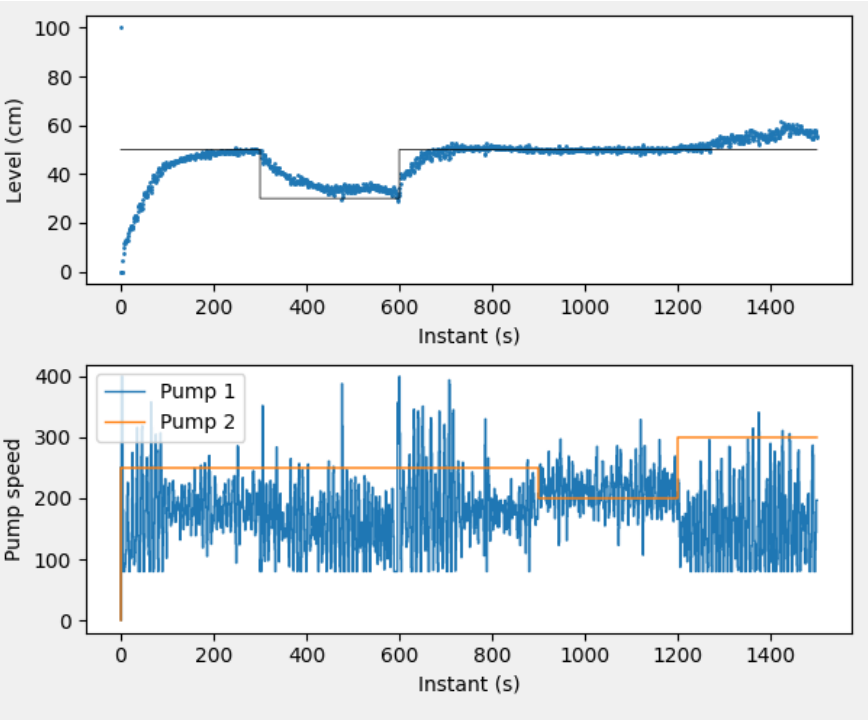


Figura 41. Resultados do método ITAE

Fonte: Autor

Já neste método, podemos perceber um ganho proporcional ainda relativamente alto, mas mais baixo que o método Ziegler-Nichols, e um ganho integral alto. A ação integral corrige o valor da variável manipulada em intervalos regulares, chamado tempo integral. E[3/10³ V]e tempo integral é definido como o inverso do ganho integral. Se o ganho integral é l xo, o sistema pode levar muito tempo para atingir o valor de referência. No entanto, se o ganho integral for muito alto, o sistema pode tornar-se instável, que foi exatamente o que aconteceu neste método. Graficamente fica fácil notar que a combinação de um alto ganho proporcional e integral gerou instabilidade no sistema.

Tabela 6. Resultados do método ITAE

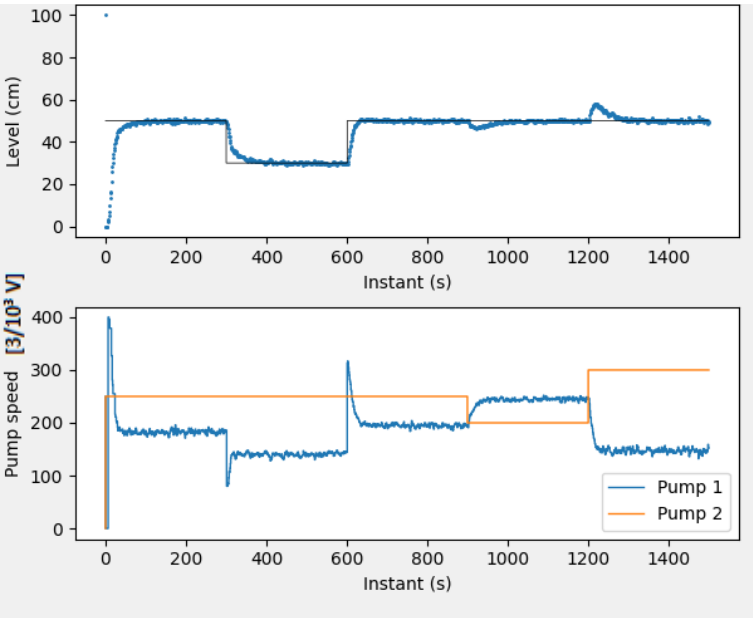
	Step 1	Step 2	Step 3	Step 4	Step 5
t (s)	253,50	294,30	280,90	226,40	Não atingiu
Desvio Padrão (DE)	0,4969	0,8598	0,4970	0,5497	-

Analisando quantitativamente os resultados do método ITAE, percebemos que o tempo de assentamento foi bastante alto, tendo em vista que cada etapa leva 300 segundos. Além disso, não foi possível atingir o assentamento no *step* 5. Esses resultados muito provavelmente foram frutos da combinação de ganhos proporcional e integral relativamente altos.

4.8 Resultados do método empírico

Para o ensaio do método empírico, foram usados os valores 8,00 e 30,00 para os ganhos proporcional (K_c) e integral (τ_i), respectivamente. Os resultados e gráfico são mostrados abaixo.

Figura 42. Resultados do método empírico



Fonte: Autor

Após observar os resultados dos métodos teóricos para o sistema em questão, percebeu-se que valores de ganho proporcional e integral muito altos não produziram resultados com boa exatidão para esse procedimento específico, então foi feita uma tentativa com valores mais baixos.

Pela análise do gráfico, é possível perceber que com ganhos proporcional e integral menores, houve uma variação menos brusca na velocidade da bomba 1 pelo controlador, gerando resultados com mais exatidão e estáveis.

Tabela 7. Resultados do método empírico

	Step 1	Step 2	Step 3	Step 4	Step 5
t (s)	174,40	111,70	24,40	67,90	299,50
Desvio Padrão (DE)	0,4372	0,4201	0,4398	0,4134	0,1933

Por fim, é possível perceber que o método empírico produziu os menores tempos de assentamento entre os *steps*, muito provavelmente devido à combinação de baixo ganho proporcional e integral.

CAPÍTULO 5

5 CONCLUSÕES

Os resultados demonstraram que a plataforma Arduino juntamente com os *softwares* utilizados se mostram eficientes para o uso no *kit* experimental de controle de processos. A instrumentação se mostrou de fácil configuração, com baixo nível de ruído e custo reduzido, o que torna viável a construção de *kits* desta natureza para utilização em escalas industriais, com as devidas alterações de componentes.

Do ponto de vista da plataforma implementada e de seu uso, os resultados obtidos permitem concluir que:

- Os resultados experimentais mostraram que a plataforma proposta (*hardware* e *software*) é eficiente para uso em configurações de estudo e pode ser até mesmo aplicada processos reais. A instrumentação possui fácil configuração e baixo custo. O sistema pode ser eficaz para o aprendizado de engenharia, programação e processos de controle.
- A placa Arduino é simples, prática, eficiente, possui baixo custo e de relativamente fácil aprendizagem, com apenas algum conhecimento prévio em programação e componentes eletrônicos é possível construir sistemas de controle complexos e de grande utilidade.
- A integração da plataforma Arduino com Python através da biblioteca PySerial permite facilitar ainda mais a parte computacional do projeto, com sintaxe mais simplificada e amigável para se programar a placa quando comparada com a linguagem C/C++. Além disso, permite o aproveitamento de toda robustez e bibliotecas próprias do Python, sendo possível a plotagem de gráficos em tempo real, criação de interfaces de usuário (GUI), modelagem de dados, cálculos estatísticos e até mesmo o uso de inteligência artificial.
- A utilização de filtros digitais implementados em *software* se mostrou suficiente para reduzir o nível de ruído

Do ponto de vista de teoria de controle de processos, os resultados obtidos permitem concluir que:

- Tanto o método teórico Cohen-Coon quanto o método empírico apresentaram resultados satisfatórios para um sistema de controle eficiente e com boa exatidão. Portanto, para o controlador em questão, deve ser feito o uso de constantes de ganho proporcional menores, gerando menos variações abruptas na velocidade da bomba e, consequentemente, uma estabilização melhor e tempo de assentamento menor.
- O controlador PID deste projeto teve êxito em sua função de controle automático do nível de água e, com os ajustes certos nos parâmetros proporcional e integral, conseguiu obter um bom desempenho em manter a altura estável após mudança de setpoint e/ou perturbações de vazão.

5.1 Sugestões para trabalhos futuros

Após a conclusão do projeto, algumas sugestões para trabalhos futuros são:

- Utilizar um tanque com altura menor ou bombas mais potentes, de modo que a voltagem aplicada nas mesmas seja a menor possível para superar a inércia do rotor e iniciar o bombeamento de líquido até o topo do tanque. Isso possivelmente pode gerar mais exatidão no controle de nível.
- Aproveitar a afinidade de Python com métodos de *machine learning* para realizar o controle utilizando inteligência artificial e fazer um comparativo com o método PID.
- Utilizar o Arduino e o sistema PID para aplicação em outros sistemas, tais como controle de temperatura, pressão, posição etc.

CAPÍTULO 6

6 REFERÊNCIAS BIBLIOGRÁFICAS

COHEN, G.H., COON, G.A (1953); ‘‘Theoretical considerations of retarded control’’. Transactions of the ASME, 827–834.

ZIEGLER, J.G & NICHOLS, N. B. (1942). "Optimum settings for automatic controllers" (PDF). Transactions of the ASME. 64: 759–768. Archived from the original (PDF) on 2017-09-18.

ASTROM, K. J. (1995). "PID Controllers: Theory, Design and Tuning”

MARINTS, F.G. (2005). **“**Tuning PID Controllers using the ITAE Criterion**”**

KONDAVEETI, Hari Kishan (2021). “A systematic literature review on prototyping with Arduino: Applications, challenges, advantages and limitations”.

MONK, Simon (2011). “Programmin Arduino, Getting Started with Sketches”.

TKINTER: INTERFACES GRÁFICAS EM PYTHON. Devmedia, 2016. Disponível em: *https://www.devmedia.com.br/tkinter-interfaces-graficas-em-python/33956*. Acesso em: 23 de fevereiro de 2022.

PYSERIAL. Pythonhosted, 2015. Disponível em: *https://pythonhosted.org/pyserial/*. Acesso em: 21 de fevereiro de 2022.

MATPLOTLIB: VISUALIZATION WITH PYTHON. Matplotlib, 2012. Disponível em: *https://matplotlib.org/.* Acesso em: 21 de fevereiro de 2022.

NUMPY USER GUIDE. Numpy, 2022. Disponível em: *https://numpy.org/doc/stable*. Acesso em: 21 de fevereiro de 2022.

ARDUINO DOCUMENTATION. Docs Arduino, 2006. Disponível em: *https://docs.arduino.cc/*. Acesso em: 17 de fevereiro de 2022.