UNIVERSIDADE FEDERAL RURAL DO SEMI-ÁRIDO

DEPARTAMENTO DE ENGENHARIAS

CAMPUS CARAÚBAS

CURSO DE BACHARELADO EM ENGENHARIA ELÉTRICA

ADRIEL BATISTA SOARES

MONITORAMENTO E CONTROLE PID DE UM SISTEMA DE TANQUES COM ESP32

CARAÚBAS – RN

2025

ADRIEL BATISTA SOARES

MONITORAMENTO E CONTROLE PID DE UM SISTEMA DE TANQUES COM ESP32.

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia Elétrica da Universidade Federal Rural do Semi-Árido (UFERSA), Campus Caraúbas, como requisito para obtenção do Título de Bacharel em Engenharia Elétrica.

Orientador: Jan Erik Mont Gomery Pinto

CARAÚBAS – RN

2025

S676m Soares, Adriel Batista.
Monitoramento e controle PID de um sistema de tanques com ESP32. / Adriel Batista Soares. - 2025.
65 f. : il.

Orientador: Jan Erik Mont Gomery Pinto.
Monografia (graduação) - Universidade Federal Rural do Semi-árido, Curso de Engenharia Elétrica, 2025.

1. Controle PID. 2. Sistema de tanques. 3. Esp32. I. Pinto, Jan Erik Mont Gomery, orient. II. Título.

Ficha catalográfica elaborada por sistema gerador automático em conformidade com AACR2 e os dados fornecidos pelo) autor(a).
Biblioteca Campus Caraúbas
Bibliotecária: Sarah Freire Bezerra
CRB: 15/1052

ADRIEL BATISTA SOARES

MONITORAMENTO E CONTROLE PID DE UM SISTEMA DE TANQUES COM ESP32

Monografia apresentada a Universidade Federal Rural do Semi-Árido como requisito para obtenção do título de Bacharel em Engenharia Elétrica.

Defendida em: 18 / 12 / 2025.

BANCA EXAMINADORA

Documento assinado digitalmente
gov.br
JAN ERIK MONT GOMERY PINTO
Data: 18/12/2025 17:41:54-0300
Verifique em https://validar.iti.gov.br

Jan Erik Mont Gomery Pinto
Prof. Dr. Presidente

Documento assinado digitalmente
gov.br
RODRIGO DE ALMEIDA COELHO
Data: 18/12/2025 17:50:42-0300
Verifique em https://validar.iti.gov.br

Rodrigo de Almeida Coelho
Prof. Dr. Membro Examinador Interno

Documento assinado digitalmente
gov.br
ERICA MANGUEIRA LIMA
Data: 18/12/2025 17:59:02-0300
Verifique em https://validar.iti.gov.br

Érica Mangueira Lima
Profa. Dra. Membro Examinador Interno

Dedico este trabalho a minha mãe, Maria de Fatima Batista, a memória do meu pai, Francisco Soares Junior e a todos os meus amigos e familiares. Por toda ajuda que me deram, carinho, incentivo e por não deixarem de acreditar.

AGRADECIMENTOS

Primeiramente agradeço a minha mãe Maria de Fátima que sempre esteve ao meu lado, e fez o possível e o impossível para me ajudar. Aos meus familiares que me apoiaram e me incentivaram, tanto moralmente como financeiramente, e mesmo os que já partiram no meio dessa jornada.

Não me esqueceria dos meus amigos e colegas de curso que foram de grande apoio durante os momentos de dificuldade do curso, tanto para me ajudar com os estudos como para me fazer rir. Edney, Fabio, João, Genesis, Luís, Leonardo, Lizandra, Tiago, Marcelo, Renato, Antônio, e Rosimery, um obrigado para vocês e muitos outros que estiveram comigo até aqui.

Agradeço também a todos do corpo docente do curso de engenharia elétrica que me ensinaram, e principalmente, ao meu orientador Jan Erik, por sua ajuda com este trabalho, me fornecendo orientação e recursos. E por fim, mas não menos importante aos meus colegas de trabalhos, e técnicos de laboratório: Allison(mimoso), Charles, Samir, Allisson(alisonzinho), Bandeira, Lincoln e Walber.

O Senhor dos Sonhos aprende que é preciso mudar ou morrer.

Neil Gaiman.

RESUMO

Este trabalho apresenta o desenvolvimento de um sistema de controle proporcional-integral-derivativo (PID) de um sistema de nível com dois tanques, utilizando o ESP32 e com monitoramento das variáveis do processo através de comunicação sem fio. Para obter o nível do sistema foi usado um sensor ultrassônico e uma bomba simples como atuador do sistema. Para validar o uso do controle PID, com algoritmo PI/PID digital, foram realizados diversos testes com diferentes taxas de vazão, sendo observado o comportamento sempre satisfatório desse tipo de controle. A implementação usando equipamentos de baixo custo, mostra que é possível utilizar esse tipo de solução para controlar sistemas de pequeno e médio porte e substituir, eventualmente e como *backup*, por um equipamento mais robusto, além de abrir a possibilidade para outros tipo de controle usando microcontroladores, sensores e atuadores com os controladores do tipo PID usando o algoritmo digital.

Palavras-chave: Esp32, controle PID, nível de água, sensor ultrassônico, sistema de tanques.

ABSTRACT

This paper presents the development of a proportional-integral-derivative (PID) control system for a two-tank level system, using the ESP32 and monitoring process variables via wireless communication. An ultrasonic sensor and a simple pump were used as the system actuator to obtain the system level. To validate the use of PID control with a digital PI/PID algorithm, several tests were performed with different flow rates, and the behavior of this type of control was always satisfactory. The implementation using low-cost equipment shows that it is possible to use this type of solution to control small and medium-sized systems and eventually replace them, as a backup, with more robust equipment, in addition to opening up the possibility for other types of control using microcontrollers, sensors, and actuators with PID-type controllers using the digital algorithm.

Keywords: Esp32, PID control, water level, ultrasonic sensor, tank system.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE GRÁFICOS

LISTA DE ABREVIATURAS E SIGLAS

A	Ampère
cm	Centímetros
CC	Corrente contínua
Hz	Hertz
k	Kilo
L	Litros
s	Segundos
V	Volts
VCC	Tensão de corrente contínua
W	Watts
µ	Micro

SUMÁRIO

1. INTRODUÇÃO

O controle automático de níveis de líquidos em tanques é uma área fundamental da engenharia de controle, com aplicações que se estendem desde processos industriais até sistemas residenciais de automação. Segundo Nise (2012), o controle visa garantir que uma grandeza de interesse, como o nível de um tanque, mantenha um comportamento desejado apesar de distúrbios e variações externas. Para isso, são empregados sistemas que atuam dinamicamente para corrigir desvios, aumentando a eficiência e a segurança dos processos.

A automação desses sistemas, utilizando microcontroladores, tem ganhado destaque nas últimas décadas pela possibilidade de implementar soluções de baixo custo e alta eficiência. O microcontrolador ESP32, por exemplo, apresenta elevado desempenho, baixo consumo e diversas interfaces de comunicação, o que o torna adequado para aplicações em Internet das Coisas (IoT) e controle embarcado (Espressif Systems Company, 2023). No controle de nível, sensores ultrassônicos são amplamente utilizados devido à sua precisão e facilidade de integração digital (Melo e Bernardes, 2020).

O controle Proporcional-Integral-Derivativo (PID) é um dos métodos mais empregados para a regulagem em sistemas dinâmicos, dado seu equilíbrio entre simplicidade e robustez (Bazanella et al., 2024). Em sistemas hidráulicos, pesquisas confirmam a eficácia do controle PID para manter níveis estáveis mesmo frente a variações na vazão, sendo aplicável em tanques industriais e redes de distribuição de água (Monteiro, 2016; Fernandes, 2022).

Apesar da consolidação teórica e prática do controle PID, sua implementação em sistemas de baixo custo utilizando plataformas modernas de prototipagem ainda representa um campo importante para estudos acadêmicos e desenvolvimento tecnológico. Segundo Åström e Hägglund (1995), aproximadamente 95% dos controladores industriais ainda utilizam a estrutura PID, porém a parametrização adequada e a compreensão do comportamento dinâmico do sistema permanecem como desafios práticos. Nesse contexto, o desenvolvimento de protótipos experimentais que integrem sensores, microcontroladores e algoritmos de controle em tempo real contribui tanto para a formação de engenheiros quanto para a validação de conceitos teóricos em aplicações concretas.

2. OBJETIVOS

Este trabalho propõe a implementação e análise de um sistema automático de controle de nível de água em dois tanques interligados, utilizando sensor ultrassônico e microcontrolador ESP32 para execução do algoritmo PID. Os testes realizados incluem variações na vazão através de uma torneira, avaliando o desempenho do sistema e a capacidade de manter o nível do tanque principal dentro de um *setpoint* definido pelo usuário, com margem de erro aceitável.

O objetivo geral é demonstrar a eficácia do controle PID no contexto do sistema implementado, contribuindo para o desenvolvimento de soluções acessíveis em automação hidráulica. E como objetivos específicos, utilizar a sintonia por tentativa e erro, além de produzir uma bancada que seja didática para aulas de controle, além de ser possível de aplicar métodos de sintonias mais sofisticadas no futuro.

Para isso, este trabalho está estruturado em capítulos que abordam a fundamentação teórica, metodologia de implementação, resultados e análises, seguidos das conclusões e sugestões para trabalhos futuros.

3. REFERENCIAL TEÓRICO

3.1 CONTROLE AUTOMÁTICO E SUA IMPORTÂNCIA

O controle automático é uma área fundamental da engenharia de sistemas, responsável por regular o comportamento de processos físicos e tecnológicos de forma autônoma, minimizando a intervenção humana (Nise, 2012). Essa tecnologia possibilita que sistemas industriais, como usinas de energia, manufaturas e redes de distribuição de água, operem com maior eficiência, segurança e confiabilidade, atendendo a requisitos rigorosos de performance (Ogata, 2011).

Segundo Franklin e Powell (2002), o controle automático é a area que estuda a implementação de algoritmos que ajustam variáveis de interesse de forma a manterem-se dentro de limites desejados, apesar da existência de perturbações no sistema. Essa capacidade de automação é crucial para melhorar a produtividade, reduzir custos operacionais e evitar falhas que possam ocasionar danos ambientais ou patrimoniais.

No contexto hidráulico, o controle de nível de líquidos exemplifica a aplicação prática do controle automático, no qual o objetivo é manter o nível de um tanque dentro de uma faixa desejada, mesmo com variações na vazão de entrada ou saída (Monteiro, 2016). Assim, a automação representa uma ferramenta indispensável para otimizar processos e atender às exigências de sustentabilidade e eficiência, cada vez mais presentes na indústria contemporânea.

3.2 CONTROLADOR PID (PROPORCIONAL-INTEGRAL-DERIVATIVO)

3.2.1 Fundamentos do Controle PID

O PID é um dos algoritmos de controle mais utilizados na indústria, sendo empregado em aproximadamente 95% dos processos industriais de controle em malha fechada (Åström; Hägglund, 1995). Sua popularidade deve-se à simplicidade conceitual, facilidade de implementação e eficácia em uma ampla gama de aplicações práticas.

Segundo Ogata (2011), o controlador PID combina três ações de controle distintas para minimizar o erro entre o valor desejado (*setpoint*) e o valor medido da variável de processo. Este erro, denotado por e(t), é definido como:

$$e(t) = r(t) - y(t), \tag{1}$$

Em que $r(t)$ representa o setpoint e $y(t)$ é o valor atual da variável controlada. A lei de controle PID, em sua forma contínua no domínio do tempo, é expressa pela equação (Dorf; Bishop, 2016):

$$U(t) = K_p e(t) + K_i \int e(t)dt + K_d \frac{de(t)}{dt}, \tag{2}$$

em que:

- $u(t)$ é o sinal de controle aplicado ao processo;
- K_p é o ganho proporcional;
- K_i é o ganho integral;
- K_d é o ganho derivativo.

3.2.1.1 Ação Proporcional (P)

A ação proporcional produz um sinal de controle que é diretamente proporcional ao erro instantâneo. De acordo com Nise (2017), esta componente fornece uma resposta imediata às variações do erro, sendo expressa por:

$$u(t) = K_p \cdot e(t), \tag{3}$$

O ganho proporcional K_p determina a magnitude da resposta do controlador ao erro. Valores elevados de K_p resultam em respostas mais rápidas e menor erro em regime permanente, porém podem causar sobressinal excessivo e instabilidade no sistema (Franklin; Powell; Emami-Naeini, 2015).

Uma limitação importante da ação proporcional pura é a presença de erro estacionário em regime permanente, onde existe uma diferença constante entre o *setpoint* e o valor controlado quando o sistema estabiliza (Ogata, 2011).

3.2.1.2 Ação Integral (I)

A ação integral é responsável por eliminar o erro em regime permanente, acumulando o erro ao longo do tempo. Segundo Åström e Hägglund (1995), a componente integral é expressa por:

$$u(t) = K_i \int_0^t e(\tau) d(\tau), \tag{4}$$

Esta ação garante que, em regime permanente, o erro seja nulo, pois o termo integral continua aumentando enquanto houver erro, mesmo que pequeno. Franklin, Powell e Emami-Naeini (2015) destacam que o ganho integral K_i controla a velocidade com que o erro acumulado é corrigido.

Entretanto, a ação integral pode introduzir oscilações no sistema e aumentar o tempo de acomodação se o ganho K_i for muito elevado, fenômeno conhecido como "*wind-up*" integral (Dorf; Bishop, 2016). Este efeito é particularmente problemático em sistemas com saturação do atuador.

3.2.1.3 Ação Derivativa (D)

A ação derivativa atua com base na taxa de variação do erro, proporcionando uma resposta antecipada às mudanças na variável de processo. Conforme Nise (2017), esta componente é dada por:

$$u(t) = K_d \frac{de(t)}{dt}, \tag{5}$$

A principal vantagem da ação derivativa é a melhoria da estabilidade do sistema e a redução do sobressinal, uma vez que ela age para reduzir a velocidade das mudanças no erro (Ogata, 2010). O ganho derivativo K_d determina a sensibilidade do controlador às variações do erro.

Por outro lado, Dorf e Bishop (2016) alertam que a ação derivativa amplifica ruídos de alta frequência presentes no sinal de medição, podendo causar instabilidade. Por esta razão, em aplicações práticas, frequentemente se utiliza um filtro passa-baixa em conjunto com a ação derivativa.

3.2.1.4 Controlador PID no Domínio da Frequência

No domínio de Laplace, a função de transferência do controlador PID é expressa por (Franklin; Powell; Emani-Naeini, 2015):

$$C(s) = K_p + K_i \frac{1}{s} + K_d s, \tag{6}$$

Esta representação é fundamental para a análise de estabilidade e desempenho do sistema em malha fechada, permitindo a utilização de ferramentas como o critério de Routh-Hurwitz e os diagramas de Bode.

3.2.2 Sintonia de Controladores PID

A sintonia adequada dos parâmetros K_p, K_i e K_d é crucial para o desempenho satisfatório do controlador PID. Segundo Åström e Hägglund (1995), existem diversos métodos de sintonia, sendo os mais comuns: Método de Ziegler-Nichels, método da resposta ao degrau, método do ganho último, Método de Cohen-Coon e sintonia experimental.

3.2.2.1 Método de Ziegler-Nichels

O método de Ziegler-Nichols é um dos procedimentos mais conhecidos para sintonia de controladores PID. Existem duas variantes: o método da resposta ao degrau e o método da oscilação contínua (Ogata, 2010).

No método da oscilação contínua (ou método do ganho último), aumenta-se gradualmente o ganho proporcional até que o sistema entre em oscilação sustentada. O ganho crítico (K_u) e o período de oscilação (P_u) são então utilizados para calcular os parâmetros do PID conforme as regras de Ziegler-Nichols (Nise, 2017). Na Tabela 1 há a regra de sintonia de Ziegler Nichols baseada no ganho crítico K_u e no período crítico T_u. T_i é o tempo integral e T_d é o tempo derivativo.

Tabela 1 - Regra de sintonia de Ziegler Nichols.

Controlador	**K_p**	**T_i**	**T_d**
P	$0{,}5K_u$	-	-
PI	$0{,}45K_u$	$T_u/1{,}2$	-
PID	$0{,}6K_u$	$T_u/2$	$T_u/8$

Fonte: Ogata (2010).

3.2.2.2 Método da Resposta ao Degrau

Este método baseia-se na resposta em malha aberta do processo a uma entrada em degrau. A partir da curva de resposta, determinam-se os parâmetros característicos do processo (constante de tempo e tempo morto), que são então utilizados para calcular os ganhos do

controlador (Dorf; Bishop, 2016). A resposta ao degrau permite aproximar o processo por um modelo de primeira ordem com tempo morto (FOPDT - *First Order Plus Dead Time*), cuja função de transferência é dada por (Skogertad, 2003):

$$G(s) = \frac{K \cdot e^{-Ls}}{\tau s + 1}, \tag{7}$$

em que:

- K é o ganho estático do processo;
- L é o tempo morto;
- s é o domínio de Laplace
- τ é a constante de tempo.

Com base nos parâmetros L e T, Ziegler e Nichols (1942) propuseram as regras de sintonia apresentadas na Tabela 2.

Tabela 2 - Regras de sintonia de Ziegler-Nichols pela resposta ao degrau.

Controlador	K_p	T_i	T_d
P	T/(L·K)	∞	0
PI	0,9T/(L·K)	L/0,3	0
PID	1,2T/(L·K)	2L	0,5L

Fonte: Adaptado de Ziegler e Nichols (1942) e Ogata (2011).

3.2.2.3 Método do ganho último

O segundo método proposto por Ziegler e Nichols (1942) baseia-se em experimentos com o sistema em malha fechada, sendo conhecido como método da sensibilidade última ou método do ganho crítico. Conforme descrito por Nise (2017), o procedimento experimental consiste em:

a) Configurar o controlador como puramente proporcional ($K_i = 0$, $K_d = 0$);

b) Aumentar gradualmente o ganho proporcional K_p a partir de valores baixos;

c) Observar a resposta do sistema a uma mudança no *setpoint* ou perturbação;

d) Continuar aumentando K_p até que o sistema entre em oscilação sustentada (limite de estabilidade);

e) Registrar dois parâmetros críticos: K_u (ganho último ou crítico): Valor de K_p que causa oscilação sustentada; P_u (período último): Período da oscilação sustentada.

Do ponto de vista da teoria de controle, o ganho último corresponde ao valor de K_p para o qual o sistema apresenta um par de polos complexos conjugados sobre o eixo imaginário do plano s, caracterizando o limite de estabilidade (Dorf; Bishop, 2016). Com base nos parâmetros K_u e P_u, as regras de sintonia propostas por Ziegler e Nichols são apresentadas na Tabela 3.

Tabela 3 - Regras de sintonia de Ziegler-Nichols pelo método do ganho último.

Controlador	**K_p**	**T_i**	**T_d**
P	$0{,}5K_u$	∞	0
PI	$0{,}45K_u$	$P_u/1{,}2$	0
PID	$0{,}6K_u$	$P_u/2$	$P_u/8$

Fonte: Adaptado de Ziegler e Nichols (1942) e Nise (2017).

3.2.2.4 Método de Cohen-Coon

O método de Cohen-Coon, desenvolvido em 1953, representa uma evolução do método de Ziegler-Nichols da curva de reação, fornecendo melhor desempenho para processos com razão L/T elevada (Cohen; Coon, 1953). Assim como o método de Ziegler-Nichols da curva de reação, o método de Cohen-Coon baseia-se na identificação dos parâmetros K, L e T a partir da resposta ao degrau em malha aberta. No entanto, as regras de sintonia são mais elaboradas, considerando explicitamente a razão ‘L/T ‘(O'Dwyer, 2009). As regras de sintonia de Cohen-Coon são apresentadas na Tabela 4.

Tabela 4 - Regras de sintonia de Cohen-Coon.

Controlador	K_p	T_i	T_d
P	$(T/KL)\cdot(1+L/(3T))$	∞	0
PI	$(T/KL)\cdot(0,9 + L/(12T))$	$L\cdot(30 + 3L/T)/(9 + 20L/T)$	0
PID	$(T/KL)\cdot(4/3 + L/(4T))$	$L\cdot(32 + 6L/T)/(13 + 8L/T)$	$L\cdot4/(11+2L/T)$

Fonte: Adaptado de Cohen e Coon (1953) e Skogestad (2003).

3.2.2.5 Sintonia Experimental

O método de tentativa e erro, também conhecido como sintonia heurística ou experimental, é uma abordagem prática amplamente utilizada na indústria para ajuste de controladores PID. Segundo Dorf e Bishop (2016), embora existam métodos analíticos sofisticados para sintonia de controladores, a abordagem experimental permanece como uma das mais utilizadas devido à sua aplicabilidade direta e não exigir conhecimento detalhado do modelo matemático do processo.

Dorf e Bishop (2016) argumentam que o método de tentativa e erro não é meramente aleatório, mas sim uma abordagem sistemática baseada no conhecimento dos efeitos de cada parâmetro do controlador sobre o comportamento do sistema. Os autores enfatizam que compreender qualitativamente como Kp, Ki e Kd afetam a resposta do sistema é fundamental para o sucesso desta técnica. A Tabela 5 é uma referência essencial para sintonia manual, que resume os efeitos qualitativos do aumento de cada parâmetro PID sobre as características principais da resposta transitória do sistema.

Tabela 5 - Efeitos do aumento dos paramentos PID nas características de resposta.

Parâmetro Aumentado	Tempo de Subida	Sobressinal	Tempo de Acomodação	Erro em Regime Permanente	Estabilidade
K_p	Diminui	Aumenta	Pequena mudança	Diminui	Degrada
K_i	Diminui	Aumenta	Aumenta	Elimina	Degrada
K_d	Pequena mudança	Diminui	Diminui	Nenhuma mudança	Melhora

Fonte: Dorf e Bishop (2016).

3.2.3 Implementação Digital do Controlador PID

Em sistemas embarcados como o ESP32, o controlador PID é implementado em sua forma discreta. Segundo Franklin, Powell e Emami-Naeini (2015), a equação discreta do PID pode ser obtida através da aproximação das operações de integração e derivação:

$$u(k) = K_p \cdot e(k) + K_i \cdot T \cdot \sum_{i=0}^{k} e(i) + \left(\frac{K_d}{T}\right) \cdot [e(k) - e(k-1)], \quad (8)$$

Em que:

- k representa o instante de amostragem atual;
- T é o período de amostragem;
- e(k) é o erro no instante k.

Ogata (2011) destaca que a escolha adequada do período de amostragem T é fundamental para o desempenho do controlador digital, devendo ser suficientemente pequeno para representar adequadamente a dinâmica do processo contínuo.

3.2.3.1 Aplicações do Controle PID em Sistemas de Nível

O controle de nível em tanques é uma aplicação clássica de controladores PID na indústria. Segundo Dorf e Bishop (2016), sistemas de controle de nível são encontrados em processos químicos, petroquímicos, tratamento de água e diversas outras aplicações industriais.

Nise (2017) explica que, em sistemas de tanques, o controlador PID atua ajustando a vazão de entrada (através de válvulas ou bombas) para manter o nível desejado, compensando perturbações como variações na vazão de saída ou mudanças no setpoint.

A dinâmica de um sistema de tanques acoplados apresenta características interessantes para o estudo de controladores PID, incluindo interação entre variáveis, tempo de resposta e comportamento não-linear em certas condições operacionais (Åström; Hägglund, 1995).

3.2.3.2 Vantagens e Limitações do Controle PID

Franklin, Powell e Emami-Naeini (2015) enumeram as principais vantagens do controlador PID:

a) Simplicidade conceitual e facilidade de implementação;

b) Aplicabilidade a uma ampla variedade de processos;

c) Bom desempenho na maioria das aplicações práticas;

d) Possibilidade de sintonia sem necessidade de modelo matemático preciso;

e) Baixo custo computacional, adequado para sistemas embarcados.

Por outro lado, Ogata (2010) e Dorf e Bishop (2016) apontam algumas limitações:

a) Desempenho subótimo em sistemas com grande tempo morto;

b) Dificuldade em lidar com processos fortemente não-lineares;

c) Sensibilidade a ruídos de medição (especialmente a ação derivativa);

d) Necessidade de ressintonia quando há mudanças significativas nas características do processo;

e) Possibilidade de "*wind-up*" integral em sistemas com saturação.

Apesar destas limitações, o controle PID continua sendo a escolha preferencial em aplicações industriais devido à sua robustez e eficácia comprovada (Åström; Hägglund, 1995)

3.3 MICROCONTROLADORES

Os microcontroladores são circuitos integrados programáveis que incorporam em um único chip um processador, memória e periféricos de entrada e saída, sendo amplamente utilizados em sistemas embarcados para automação e controle (Pereira, 2003). Segundo Nicolosi (2002), um microcontrolador difere de um microprocessador convencional por integrar todos os componentes necessários para o funcionamento de um sistema computacional completo, eliminando a necessidade de componentes externos adicionais.

De acordo com Monk (2014), a escolha adequada de um microcontrolador para uma aplicação específica deve considerar fatores como capacidade de processamento, quantidade de memória, número e tipo de periféricos disponíveis, consumo de energia e custo.

3.3.1 O microntrolador ESP32

O ESP32 é um microcontrolador de baixo custo e baixo consumo de energia desenvolvido pela empresa chinesa *Espressif Systems*, lançado comercialmente em 2016 (Espressif Systems, 2023). Este dispositivo representa uma evolução significativa em relação ao seu predecessor ESP8266, oferecendo maior capacidade de processamento e recursos adicionais.

Segundo Kurniawan (2018), o ESP32 foi projetado especificamente para aplicações de Internet das Coisas (IoT), combinando conectividade *Wi-Fi* e *Bluetooth* com capacidades de processamento adequadas para sistemas embarcados complexos. Sua popularidade na em

aplicações industriais deve-se à excelente relação custo-benefício e ao amplo suporte de ferramentas de desenvolvimento. A Figura 1 exibe o ESP32 e seu *layout* de *pinout*, ou 'pinos', estes que são as entradas e saídas físicas, dos sinais que são trabalhados neste microcontrolador.

Figura 1 - Pinout ESP32.

Fonte: upesy.com (2023).

3.3.2 Arquitetura e Características Técnicas do ESP32

O ESP32 utiliza um processador dual-core Xtensa LX6 de 32 bits, operando em frequências de até 240 MHz (Espressif Systems, 2023). Esta arquitetura *dual-core* permite a execução paralela de tarefas, proporcionando melhor desempenho em aplicações que exigem multitarefa em tempo real.

Kurniawan (2018) destaca que a arquitetura Xtensa LX6 oferece um conjunto robusto de instruções, incluindo operações de ponto flutuante e instruções SIMD (*Single Instruction, Multiple Data*), tornando o ESP32 adequado para processamento de sinais digitais e algoritmos de controle complexos.

O ESP32 possui uma arquitetura de memória diversificada que inclui (Espressif Systems 2023):

a) **ROM (*Read-Only Memory*)**: 448 KB para armazenamento do *bootloader* e funções básicas;

b) **SRAM (*Static Random Access Memory*)**: 520 KB de memória RAM interna para execução de programas e armazenamento de dados temporários;

c) ***Flash externa***: Tipicamente 4 MB, podendo variar conforme o módulo, utilizada para armazenamento do programa principal e dados não-voláteis;

d) **RTC SRAM**: 8 KB de memória de baixo consumo que mantém dados durante o modo de hibernação (*deep sleep*).

Segundo Seneviratne (2028), esta configuração de memória é adequada para a maioria das aplicações de controle e automação, permitindo o armazenamento de programas complexos e *buffers* de dados significativos.

Uma das características distintivas do ESP32 é seu conjunto integrado de interfaces de comunicação sem fio:

Wi-Fi: Suporta os padrões 802.11 b/g/n, operando na faixa de 2,4 GHz, com suporte a modos *Station*, *Access Point* e *Station*+AP simultâneo (Espressif Systems, 2023). Esta capacidade permite que o ESP32 se conecte a redes existentes ou crie sua própria rede para comunicação com outros dispositivos.

Bluetooth*:* Inclui *Bluetooth* v4.2 BR/EDR e BLE (*Bluetooth Low Energy*), permitindo comunicação com uma ampla variedade de dispositivos (Kurniawan, 2018). O suporte a BLE é particularmente relevante para aplicações IoT devido ao seu baixo consumo energético.

Monk (2022) observa que a integração nativa de conectividade sem fio elimina a necessidade de módulos externos, reduzindo custos e complexidade do hardware.

3.3.3 Periféricos e Interfaces

O ESP32 oferece um rico conjunto de periféricos integrados, essenciais para aplicações de controle e aquisição de dados, disponibiliza até 34 pinos GPIO programáveis, que podem ser observados na Figura 1, permitindo interface com sensores, atuadores e outros dispositivos digitais (Espressif Systems, 2023). Estes pinos podem ser configurados como entradas ou saídas digitais, com suporte a interrupções e *pull-up/pull-down* interno.

O dispositivo possui dois conversores analógico-digitais SAR (*Successive Approximation Register*) de 12 bits, totalizando 18 canais de entrada analógica (Kurniawan, 2018). Esta característica é fundamental para leitura de sensores analógicos e monitoramento de grandezas físicas. Segundo Seneviratne (2020), a resolução de 12 bits (4096 níveis) é adequada para a maioria das aplicações de instrumentação e controle, embora seja necessário considerar a não-linearidade inerente dos ADCs do ESP32 em aplicações que exigem alta precisão.

Dois canais DAC de 8 bits permitem a geração de sinais analógicos, úteis para controle de atuadores analógicos e geração de formas de onda (Espressif Systems, 2023). O ESP32 oferece 16 canais PWM independentes com resolução configurável de até 16 bits, operando em frequências de até 40 MHz (Monk, 2022). O PWM é amplamente utilizado para controle de motores, LEDs e outros atuadores que requerem sinais de potência modulada.

Segundo Kurniawan (2018), o ESP32 suporta múltiplos protocolos de comunicação serial:

a) **UART**: 3 interfaces UART para comunicação serial assíncrona;

b) **SPI**: 4 interfaces SPI (*Serial Peripheral Interface*) para comunicação síncrona de alta velocidade;

c) **I²C**: 2 interfaces I²C para comunicação com múltiplos dispositivos usando apenas dois fios;

d) **I²S**: Interface para comunicação com dispositivos de áudio digital.

Esta diversidade de interfaces facilita a integração com uma ampla variedade de sensores e módulos periféricos.

O ESP32 possui 4 *timers* de *hardware* de 64 bits, além de *watchdog timers* para garantir a confiabilidade do sistema (Espressif Systems, 2023). Os *timers* são essenciais para implementação de bases de tempo precisas em sistemas de controle.

3.3.5 Recursos de Software e Programação

O ESP32 vem com o FreeRTOS integrado, um sistema operacional em tempo real de código aberto (Kurniawan, 2018). Segundo Barry (2018), o FreeRTOS permite a implementação de sistemas multitarefa com escalonamento preemptivo, essencial para aplicações complexas que exigem gerenciamento eficiente de múltiplas tarefas concorrentes.

O uso de RTOS facilita o desenvolvimento de sistemas de controle que precisam executar múltiplas operações simultaneamente, como leitura de sensores, processamento de algoritmos de controle e comunicação de dados (Seneviratne, 2018).

O ESP32 pode ser programado através de diversas plataformas e linguagens:

ESP-IDF (*Espressif IoT Development Framework*): *Framework* oficial baseado em linguagem C, oferecendo acesso completo aos recursos do hardware (Espressif Systems, 2023). Este ambiente é recomendado para aplicações profissionais que exigem máximo desempenho e controle sobre o *hardware*.

Arduino IDE: Através da instalação do core ESP32 para Arduino, o microcontrolador pode ser programado utilizando a sintaxe simplificada do Arduino (Monk, 2022). Esta abordagem facilita a prototipagem rápida e é amplamente adotada em projetos educacionais e pela comunidade *maker*.

MicroPython: Implementação do Python 3 otimizada para microcontroladores, permitindo programação em linguagem de alto nível (Kurniawan, 2018).

Segundo Monk (2022), a escolha do ambiente de desenvolvimento deve considerar a complexidade do projeto, experiência do desenvolvedor e requisitos de desempenho. A popularidade do ESP32 resultou em um rico ecossistema de bibliotecas de código aberto. Seneviratne (2020) destaca que existem bibliotecas disponíveis para praticamente todos os sensores e atuadores comuns, protocolos de comunicação IoT (MQTT, HTTP, CoAP) e serviços em nuvem, acelerando significativamente o desenvolvimento de aplicações.

O ESP32 foi projetado com foco em eficiência energética, oferecendo diversos modos de operação para otimizar o consumo conforme a aplicação (Espressif Systems, 2023):

a) **Modo ativo**: Consumo de até 240 mA com Wi-Fi e Bluetooth ativos;

b) ***Modem-sleep***: Desliga o rádio Wi-Fi entre transmissões, consumindo cerca de 20-68 mA;

c) ***Light-sleep***: Suspende a CPU mantendo periféricos ativos, consumo de 0.8 mA;

d) ***Deep-sleep***: Modo de hibernação profunda com consumo de apenas 10 µA.

Kurniawan (2018) explica que a capacidade de alternar entre esses modos permite que o ESP32 seja utilizado em aplicações alimentadas por bateria, um requisito comum em dispositivos IoT remotos.

O ESP32 tem sido amplamente adotado em aplicações de automação industrial, residencial e sistemas de controle devido às suas características técnicas e baixo custo (Seneviratne, 2020). Segundo Monk (2022), exemplos de aplicações incluem:

a) Sistemas de controle de processos industriais;

b) Automação residencial (*smart home*);

c) Monitoramento remoto de variáveis ambientais;

d) Controle de atuadores e máquinas;

e) Sistemas de aquisição de dados;

f) Controladores para robótica.

Especificamente para sistemas de controle de nível em tanques, Kurniawan (2018) observa que o ESP32 oferece todos os recursos necessários: entradas para sensores, saídas para acionamento de bombas, capacidade de processamento para algoritmos de controle (como PID) e conectividade para monitoramento remoto.

4. MATERIAIS E METODOLOGIA

4.1 MATERIAIS UTILIZADOS

Este trabalho adotou uma abordagem experimental para desenvolver e avaliar um sistema automático de controle do nível de água em tanques interligados, utilizando o

microcontrolador ESP32 e a lógica de controle PID. O sistema proposto foi montado com dois tanques numerados 1 e 2, no qual o tanque 1 possui uma torneira regulável que permite o escoamento da água para o tanque 2. ambos os tanques podem ser observados na Figura 2 e Figura 3.

Figura 2 – Tanque 1.

Fonte: Autoria própria (2025).

Figura 3 - Tanque 2.

Fonte: Autoria própria (2025).

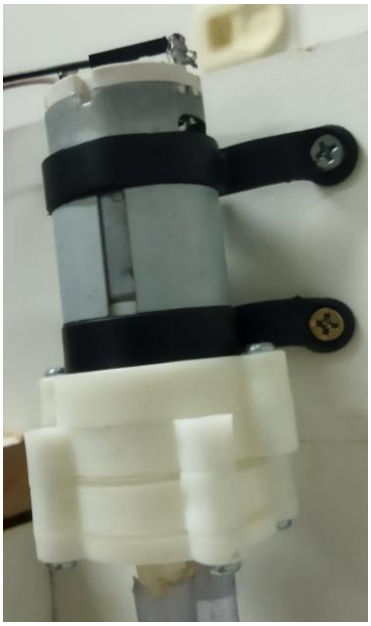
O tanque 1 possui 13 cm de largura, 13 cm de comprimento e 20 cm de altura, e ao todo um volume de 3,39 L. O tanque 2 tem 13 cm de largura, 13 de comprimento e 13 de altura, sendo assim um volume total de 2,19. Por medidas de segurança e questões construtivas, nem todo o volume dos tanques pode ser preenchido, logo, o volume utilizável do tanque 1 é de 2,45 L, e do tanque 2 é de 1,74 L.

Essas limitações nos volumes dos reservatórios se devem às, foi considerado as características físicas dos objetos: ambos os tanques possuem furos de 1 cm de diâmetro em sua parte superior, com o objetivo de encaixe das mangueiras por onde entra e sai água dos tanques. No tanque 1, também foi considerado o volume necessário para começar o escoamento da água pela torneira, já que está a 2 cm de distância do fundo do tanque.

Se o nível da água alcança a alturas desses furos, haverá vazamento de água, em especial no tanque 1, esse furo também possui uma função de proteção, já que impede o nível da água atingir o sensor localizado na parte superior, e que não é a prova de água. A torneira utilizada nesse projeto é uma de vazamento ajustável, comumente utilizada em residências.

Como atuador desse projeto de automação foi utilizado uma bomba d'água. Que pode ser vista na Figura 4 e é controlada pelo ESP32, realiza o bombeamento da água do tanque 2 para o tanque 1, de modo a manter o nível no tanque 1 de acordo com um *setpoint* pré-estabelecido pelo usuário na programação do ESP32. A bomba d'agua utilizada foi uma de 12 V e 10 W, comumente usada em aquários de pequeno porte, que possui uma capacidade de vazão de 1,5 litros por minuto.

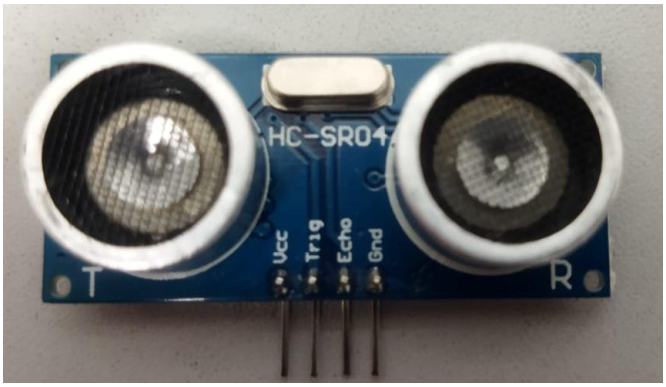
Figura 4 - Bomba d'água.



Fonte: Autoria própria (2025).

Para monitorar o nível de água no tanque 1, foi utilizado sensor ultrassônico HC-SR04 que envia dados de distância ao microcontrolador, permitindo o cálculo preciso da altura do líquido. O sensor pode ser visto na Figura 5, esse tipo específico funciona como um sonar e envia oito pulsos ultrassónicos de 40 KHz, que ao entrarem em contato, com uma superfície, retornam ao receptor do sensor, que envia um sinal, equivalente ao tempo levado entre o envio dos pulsos e o retorno, para o microcontrolador.

Figura 5 - Sensor ultrassônico.

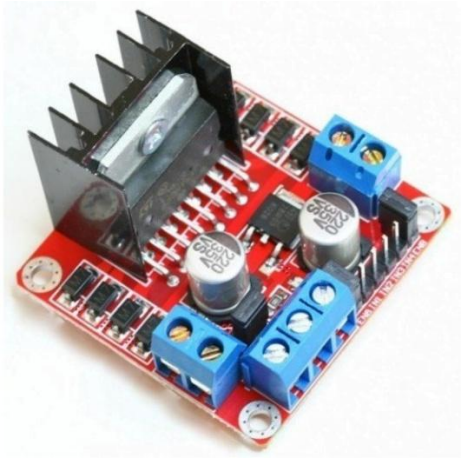


Fonte: Autoria própria (2025).

Também foi usado um modulo de ponte H L298N Figura 6, Driver de potência para acionamento da bomba d'água, permitindo o controle de velocidade através de sinal PWM e

proteção do microcontrolador contra correntes elevadas. Esse *driver* suporta tensões de 5 V a 35 V e corrente contínua de até 2 A por canal. Para alimentação do atuador foi usado uma fonte de 12 V que se conecta a ponte H para fornecer a tensão necessário para seu funcionamento.

Figura 6 - Ponte H.



Fonte: eletrogate.com (2022).

Por fim, outros materiais necessários:

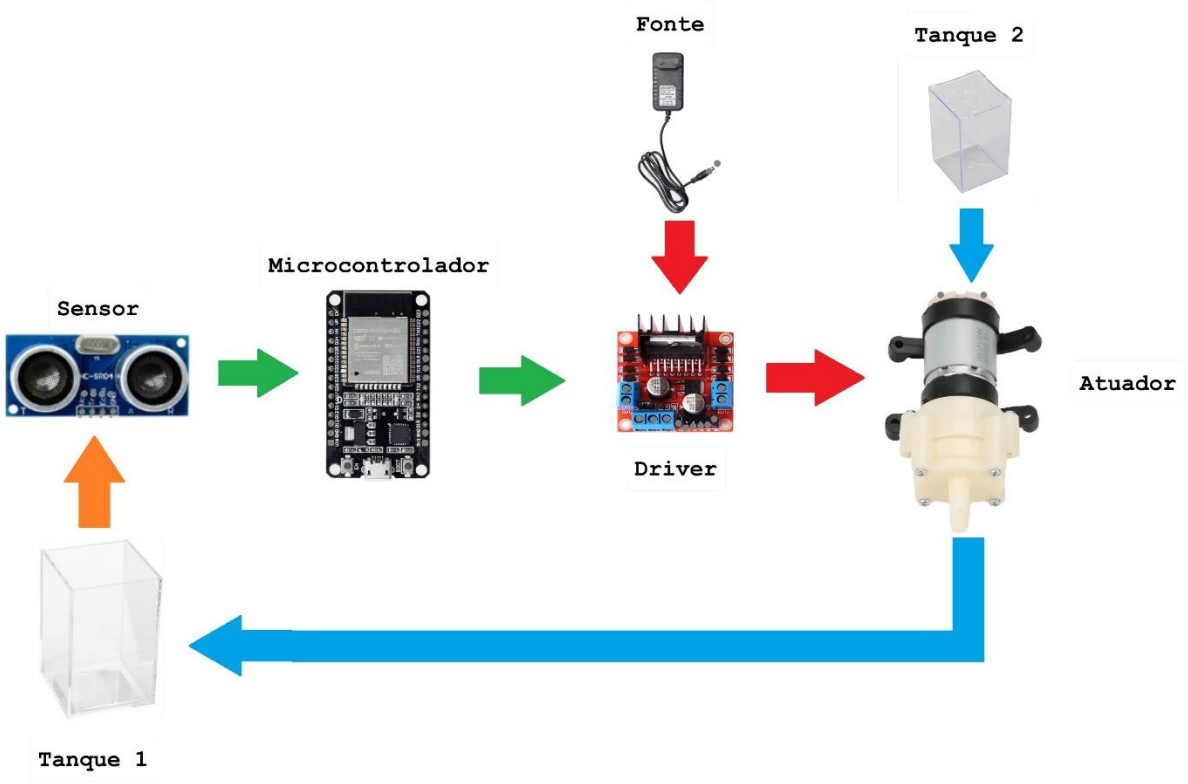
- O microcontrolador ESP32, que será utilizado para controlar todos os processos;
- *Jumpers* que fazem as ligações elétricas entre os componentes eletrônicos;
- Mangueiras de borracha transparente, que se conectam a bomba d'água por onde a água é transportada;
- Uma base de madeira compensada onde todo o projeto vai montado, garantindo uma organização visual.

Na Figura 7, é possível observar um diagrama do fluxo de montagem e de funcionamento do sistema de tanques. O sensor detecta o nível da água no Tanque 1, esse processo é simbolizado pela seta laranja, depois os dados são transmitidos para o microcontrolador, simbolizado seta verde. Após os dados serem analisados, o microcontrolador toma uma decisão baseado no seu algoritmo, e envia um sinal de controle para o *driver*, destacado pela seta verde.

O *driver* atua de acordo com o sinal que recebe, sendo energizado pela fonte (seta vermelha), e enviando o valor de tensão determinado para o atuador (bomba d'agua). Por fim, a seta azul representa o fluxo de água, o líquido em questão é retirado do Tanque 2 e transferido

para o Tanque 1 pelo atuador. A água retorna ao Tanque 2 pela torneira que está presente no Tanque 1.

Figura 7 - Fluxograma do sistema de tanques.



Fonte: Autoria própria (2025).

4.2 MONTAGEM DO *HARDWARE*

Para execução do sistema de tanques, foram feitas as seguintes conexões entre os componentes eletrônicos:

Sensor HC-SR04:

- Pino VCC conectado à alimentação de 5 V do pino VIN do ESP32;
- Pino GND conectado ao terra do pino GND do ESP32;
- Pino TRIG conectado ao pino GPIO D5 do ESP32;
- Pino ECHO conectado ao pino GPIO D18 do ESP32.

Ponte H L298N:

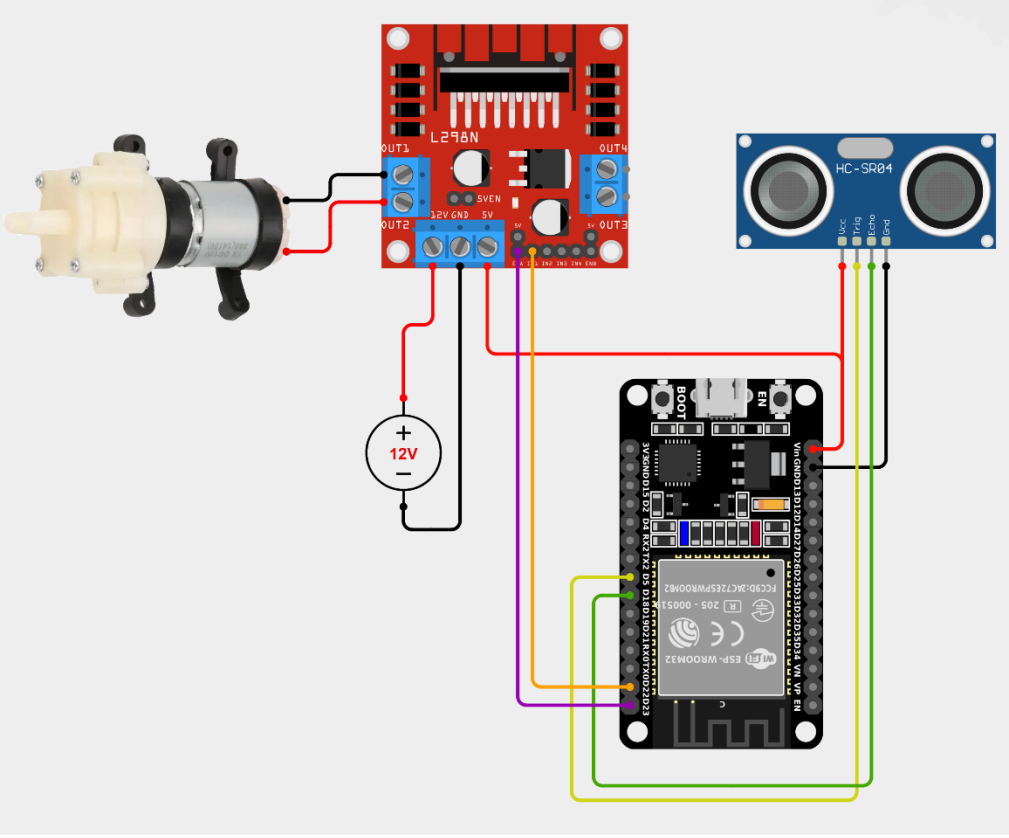
- Pinos de alimentação (VCC e GND) conectados à fonte de 12 V e 2 A;
- Pino ENA (*Enable* A) conectado ao pino GPIO D23 do ESP32 para controle PWM da velocidade da bomba;
- Pino IN1 conectado ao pino D22 do ESP32;
- Pinos OUT1 e OUT2 conectados aos terminais da bomba d'água.

ESP32:

- Alimentado através da porta USB durante a programação e testes;

A Figura 8 é ilustrado o resultado esquemático dessas conexões, sendo possível visualizar o resultado preliminar.

Figura 8 - Esquema de conexões dos componentes.



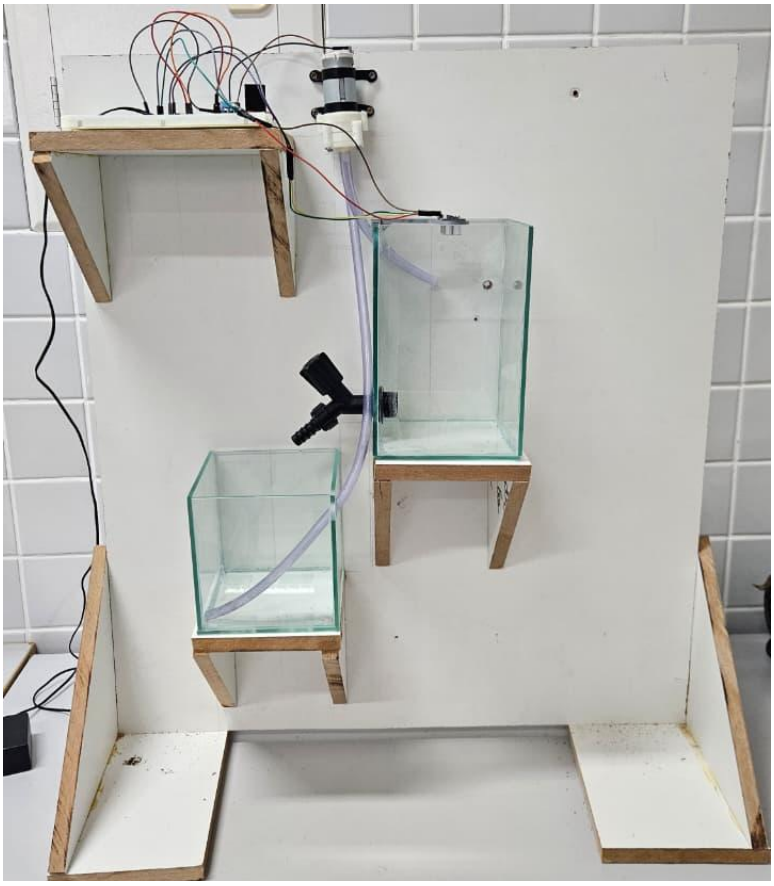
Fonte: Autoria própria (2025).

O sistema foi montado da seguinte forma: os dois tanques foram posicionados verticalmente, há certa distância horizontal um do outro, com o Tanque 2 abaixo do Tanque 1, facilitando o escoamento por gravidade através da torneira. O sensor HC-SR04 foi fixado na parte superior do Tanque 1, centralizado e apontando perpendicularmente para baixo em direção à superfície da água, a uma altura fixa de 19 cm em relação ao fundo do tanque.

A bomba d'água foi posicionada com parafuso na base de madeira, e a mangueira por onde entra a água, com a ponta completamente submersa no Tanque 2 para garantir sucção adequada, e a outra extremidade no encaixe de entrada de água da bomba, uma mangueira conecta a saída da bomba à parte superior do Tanque 1, permitindo o retorno da água;

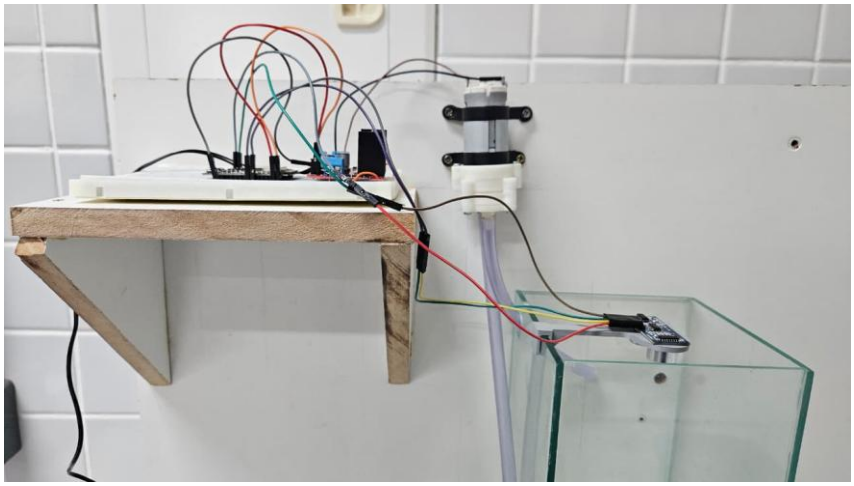
A torneira foi instalada na parte inferior do Tanque 1, com sua vazão direcionando a água de volta ao Tanque 2. Os componentes eletrônicos (ESP32 e ponte H) foram mantidos em área seca, numa plataforma acima, distante dos tanques para evitar contato com água. O produto desse esquema pode ser visto em Figura 9 e Figura 10.

Figura 9 - Vista frontal da automação de tanques.



Fonte: Autoria própria (2025).

Figura 10 - Componentes eletrônicos conectados.



Fonte: Autoria própria (2025).

4.3 IMPLEMENTAÇÃO DO *SOFTWARE*

A implementação do controle PID no ESP32 ajusta dinamicamente a operação da bomba, acionando-a quando o nível está abaixo do *setpoint* e mantém um sinal de controle fixo dentro do ponto de operação desejado ao atingir o valor desejado, garantindo um controle eficiente e com mínima oscilação. A programação utilizada foi feita na IDE arduino, e pode ser encontrada no Anexo A.

O ESP32 executa três funções principais, a primeira é a aquisição de dados. A função ‘lerNivelAgua()’ realiza a leitura do HC-SR04 e retorna a distância medida em centímetros. A Figura 11 exibe o trecho em que a lógica do sensor funciona, ‘digitalWrite(TRIG_PIN, LOW)’ garante que o pino TRIG começa em nível baixo. Evita falsos pulsos se antes estivesse em HIGH.

Já ‘delayMicroseconds(2)’ faz uma pequena estabilização (≥ 2 µs) antes de gerar o pulso de disparo. Normalmente se usa 2 µs para ter certeza de que o TRIG esteja baixo. ‘digitalWrite(TRIG_PIN, HIGH)’ inicia o pulso de disparo. O HC-SR04 requer um pulso de pelo menos 10 µs para emitir o pacote ultrassônico.

O ‘delayMicroseconds(10)’ mantém o pulso HIGH por 10 µs, isso aciona o sensor a emitir um *burst* ultrassônico. Então, ‘digitalWrite(TRIG_PIN, LOW)’ finaliza o pulso de *trigger*. Na linha onde se encontra ‘long duracao = pulseIn(ECHO_PIN, HIGH, 30000)’,

‘pulseIn’ mede quanto tempo (µs) o pino ECHO fica em nível HIGH, este é o tempo que o pulso levou para ir do sensor até a superfície da água e voltar (ida + volta).

O terceiro argumento (30000) é o *timeout* em µs. Se nada for recebido dentro desse tempo, pulseIn retorna 0. Vale ressaltar que 30 ms (30.000 µs) é um *timeout* suficiente, pois considerando 0,0343 cm/µs como velocidade do som, isso corresponde a uma medição de distância máxima teoricamente em torno de 514,5 cm, ou seja, mais que suficiente para um tanque de 20 cm. Em sequência, ‘double distanciaCM = duracao * 0.0343 / 2’ faz a conversão de tempo em distância, como ‘duracao’ é tempo de ida mais tempo de volta, divide-se por 2 para obter a distância de ida.

Para completar as três últimas linhas de código, ‘if (distancia_cm > ALTURA_TANQUE) distancia_cm = ALTURA_TANQUE’, Limita a leitura ao máximo físico do tanque (evita valores absurdos se o sensor ler o ambiente além do tanque), ‘if (distancia_cm < 0) distancia_cm = 0’, é por segurança extra (na prática ‘pulseIn’ não retorna negativo). ‘return distancia_cm’, retorna a distância em centímetros.

Figura 11 - Trecho referente ao uso do sensor ultrassônico.

```
113  // Função para ler o HC-SR04
114  double lerNivelAgua() {
115    digitalWrite(PIN_TRIG, LOW);
116    delayMicroseconds(2);
117    digitalWrite(PIN_TRIG, HIGH);
118    delayMicroseconds(10);
119    digitalWrite(PIN_TRIG, LOW);
120
121    long duration = pulseIn(PIN_ECHO, HIGH, 30000); // Timeout curto para não travar
122
123    if (duration == 0) return erro_anterior; // Falha na leitura, mantém o último valor
124
125    double distancia_cm = duration * 0.034 / 2;
126
127    // Nível é a altura total menos o espaço vazio medido
128    double nivel = ALTURA_TANQUE - distancia_cm;
129
130    // Filtro simples para evitar valores negativos malucos
131    if (nivel < 0) nivel = 0;
132
133    return nivel;
134  }
```

Fonte: Autoria própria (2025).

O valor da distância medida é convertido para nível de água conforme descrito acima. Para o cálculo do erro, o usuário define um nível desejado em centímetros (*setpoint*). A cada iteração do loop, calcula-se:

$$\text{erro} = \text{setpoint} - \text{nível}, \tag{9}$$

Esse erro é a entrada do controlador PID. Ele utiliza três componentes, componente proporcional (K_p)**,** que reage diretamente à magnitude do erro. Quanto maior o erro, maior a ação de controle (maior PWM). Aumentar K_p torna o sistema mais rápido, porém mais suscetível a oscilações. O componente integral (K_i), acumula o erro ao longo do tempo, corrigindo defasagens permanentes e garantindo que o nível atinja exatamente o *setpoint*. Valores altos de K_i podem gerar oscilação lenta e instabilidade. Por fim, o componente derivativo (K_d), avalia a taxa de variação do erro. Atua como um amortecedor, reduzindo *overshoot* e suavizando a aproximação do *setpoint.* Aumentar K_d amplifica o ruído do sistema, resultando em um comportamento errático ou instável. Para este caso os ganhos escolhidos, pelo método experimental, foram:

Configuração 1: $K_p = 10{,}0$, $K_i = 1{,}7$ e $K_d = 0$;

Configuração 2: $K_p = 15{,}0$, $K_i = 3{,}5$ e $K_d = 0$;

Configuração 3: $K_p = 10{,}0$, $K_i = 2{,}2$ e $K_d = 0{,}5$.

tornando assim esse, um sistema flexivelmente PI e PID.

A saída do PID (saidaPID) é limitada entre 60 e 255, representando diretamente o *duty-cycle* do PWM:

$$PWM = \text{saidaPID}, \tag{10}$$

o qual é enviado ao canal PWM do ESP32: ‘ledcWrite(0, (int)saidaPID)’;

Assim:

- 0 = bomba desligada
- 255 = bomba em potência máxima
- valores intermediários = controle proporcional da velocidade da bomba

A ponte H recebe o PWM no pino ENA e utiliza o pino IN1 para definir direção e acionamento. O sistema só aciona a bomba quando o nível está abaixo do setpoint. Quando o nível atinge ou ultrapassa o valor desejado, a bomba é desligada automaticamente, a lógica pode ser vista na Figura 12.

Figura 12 - Trecho da lógica do PID.

```
// Executa o cálculo do PID em intervalos fixos (Sample Time)
if (now - lastTime >= sampleTime) {

  // 1. Ler o sensor e calcular o nível
  double nivel_atual = lerNivelAgua();

  // 2. Calcular o Erro
  erro = SETPOINT - nivel_atual;

  // 3. Calcular Termo Integral (Acumula o erro ao longo do tempo)
  integral += erro;
  if (integral > 100) integral = 100;  // Anti-windup: Limita a integral para não crescer infinitamente
  if (integral < -100) integral = -100;

  // 4. Calcular Termo Derivativo (Variação do erro, prevê o futuro)
  derivada = erro - erro_anterior;

  // 5. Fórmula do PID
  saida_pid = (Kp * erro) + (Ki * integral) + (Kd * derivada);

  // 6. Converter saída do PID para PWM (0 a 255)
  int pwm_output = (int)saida_pid;  // Se o erro for grande, PID será alto. Se negativo (passou do nível), será 0.

```

Fonte: Autoria própria (2025).

Essa segunda parte da programação, é responsável pela saída PID e funciona da seguinte forma:

‘double nivel_atual = lerNivelAgua();’

Que chama a função que mede a distância e converte para nível. Fisicamente falando, ler as informações dadas pelo sensor ultrassónico, que mede o nível da água. Se o tanque tem 20 cm e a água está na metade, ‘nivel_atual’ será **10.0**.

‘erro = SETPOINT - nivel_atual;'

Matematicamente**:** E(t) = SP - PV (Erro = *Setpoint* - *Process Variable*). É a distância entre o *setpoint* e o nível atual da água. Considerando os seguinte cenários: Se o erro for positivo, o *Setpoint* (10) - Nível (8) = 2, significa que falta água. O PID vai gerar um número positivo para ligar a bomba. Se o erro for zero o *Setpoint* (10) - Nível (10) = 0, significa que o *setpoint* foi atingido. O termo Proporcional será zero. Se o erro for negativo então o *Setpoint*

(10) - Nível (12) = -2. Significa que tem água demais. O PID vai tentar diminuir a força (ou desligar).

‘integral += erro;’

Se o erro é pequeno (ex: 0,5 cm). O termo proporcional pode ser fraco demais para girar a bomba. O sistema ficaria parado faltando 0,5 cm para sempre. Funcionando da seguinte forma: Ciclo 1: Erro 0,5 -> Integral = 0,5; Ciclo 2: Erro 0,5 -> Integral = 1,0 (aumentou); Ciclo 10: Erro 0,5 -> Integral = 5,0 (Ficou forte). Mesmo com erro pequeno, a Integral cresce até a bomba ter força para empurrar esse restante de água.

‘if (integral > 100) integral = 100;’

‘if (integral < -100) integral = -100;’

Se caso o sensor seja segurado com a mão do usuário, "simulando" tanque vazio por 1 minuto, a variável integral vai somar exageradamente (ex: chegar a 50.000). Quando posicionada de volta no lugar, o valor será tão alto que a bomba vai ficar ligada no máximo por muito tempo até descer esse valor, transbordando o tanque. Como solução foi feito um limitador, não importa o quanto erre, a memória de erro nunca passa de 100, isso mantem o sistema responsivo.

‘derivada = erro - erro_anterior;’

Como o tempo é fixo (100 ms), basta subtrair os erros. Em um cenário prático, se o tanque está enchendo muito rápido: Tempo 0 ms: Erro = 5,0 (Nível 5); Tempo 100 ms: Erro = 3,0 (Nível 7 - subiu 2 cm rápido); derivada = 3,0 – 5,0 = -2,0. O resultado negativo (-2,0) indica que mesmo que o nível ainda esteja baixo “precisa-se de mais água”, a derivada negativa vai subtrair do valor final, reduzindo o PWM, desacelerando a bomba.

‘saida_pid = (Kp * erro) + (Ki * integral) + (Kd * derivada);’

Analisando essa linha com um exemplo numérico: $K_p = 15$, $K_i = 0,5$ e $K_d = 2$. O erro atual = 2 (Falta água); integral acumulada = 10 (Falta água há um tempo); derivada = -1 (A água está subindo). A conta seria:

1. **Proporcional:** 15 x 2 = 30 (Força base necessária).

2. **Integral:** 0.5 x 10 = 5 (Força extra pela persistência do erro).

3. **Derivada:** 2 x (-1) = -2 (Diminuição da força porque está subindo).

‘Total (saida_pid): 30 + 5 - 2 = 33’. Esse valor "33" será convertido para o PWM da bomba.

Resumo Lógico

- **Proporcional (Kp * erro):** Aumenta a magnitude de força do atuador com base nos dados do sensor.
- **Integral (Ki * integral):** Aumenta ainda mais a força, com base no tempo levado para haver mudança nos dados captados pelo sensor.
- **Derivada (Kd * derivada):** Diminui a força caso a mudança de estado do sistema esteja rápida demais.

4.4 EXPERIMENTAÇÃO E TESTES

Os testes experimentais foram conduzidos de forma sistemática para avaliar o desempenho do sistema de controle PID sob diferentes condições de operação. A metodologia experimental foi estruturada para permitir a análise comparativa entre diferentes configurações de parâmetros do controlador e diferentes níveis de perturbação no sistema.

Os experimentos foram planejados seguindo uma abordagem fatorial, onde foram testadas múltiplas combinações de parâmetros PID (K_p, K_i e K_d), em três condições distintas de vazão de saída. Esta abordagem permite identificar não apenas os efeitos individuais de cada variável, mas também possíveis interações entre elas.

Para simular diferentes níveis de perturbação no sistema, a torneira do Tanque 1 foi ajustada em três posições distintas, caracterizando diferentes vazões de saída:

a) **Sem vazão**: Torneira fechada, sem perturbação no sistema.

b) **Vazão Baixa**: Torneira aberta em aproximadamente 20% de sua capacidade total, representando uma perturbação leve no sistema;

c) **Vazão Média**: Torneira aberta em aproximadamente 50% de sua capacidade total, caracterizando uma perturbação moderada;

d) **Vazão Alta**: Torneira completamente aberta (100%), representando a máxima perturbação possível no sistema.

A abertura da torneira foi controlada manualmente através de sua válvula de ajuste, mantendo-se a posição durante todo o período de cada teste individual. Segundo Ogata (2011), a variação da perturbação permite avaliar a robustez do controlador e sua capacidade de rejeição a distúrbios.

Para cada condição de vazão, foram testadas diversas combinações dos parâmetros K_p, K_i e K_d do controlador PID. Os valores foram ajustados seguindo o método de tentativa e erro, conforme descrito anteriormente, buscando-se identificar a configuração que proporcionasse o melhor desempenho em termos de tempo de resposta, estabilidade e erro em regime permanente.

Os testes foram iniciados com valores conservadores dos parâmetros, aumentando-os gradualmente até observar-se resposta satisfatória ou instabilidade do sistema. De acordo com Dorf e Bishop (2016), esta abordagem incremental é segura e permite compreender o efeito de cada parâmetro no comportamento do sistema.

Cada teste individual seguiu o seguinte protocolo padronizado:

a) **Preparação**: Enchimento inicial do Tanque 2 com volume de água suficiente para operação contínua e ajuste da torneira na posição desejada (0%, 20%, 50% ou 100%);

b) **Programação**: Configuração dos parâmetros K_p, K_i e K_d no código do ESP32 conforme a combinação a ser testada, e definição do setpoint desejado;

c) **Condição inicial**: Início do teste com o Tanque 1 vazio ou com nível significativamente abaixo do *setpoint*, permitindo observar a resposta completa do sistema desde o estado inicial até a estabilização;

d) **Acionamento**: Início da execução do programa no ESP32 e ativação da aquisição de dados através do Monitor Serial da IDE Arduino, e no *smartphone* através do *bluetooth*;

e) **Monitoramento**: Observação contínua do comportamento do sistema durante o período transitório e em regime permanente;

f) **Registro**: Captura dos dados exibidos no *Monitor Serial* e registro das características observadas no *Serial Plotter*;

g) **Duração**: Cada teste foi mantido por tempo suficiente para garantir que o sistema atingisse o regime permanente e demonstrasse estabilidade (ou instabilidade, quando aplicável);

h) **Finalização**: Após conclusão do teste, os dados foram salvos e o sistema foi preparado para o próximo experimento.

i) Para avaliar a robustez e adaptabilidade a combinação dada por $K_p = 10.0$, $K_i = 2.2$ e $K_d = 0.5$ (considerada a melhor), foi conduzido teste adicional com alternância entre as três vazões durante operação contínua. Este teste simula condições práticas onde perturbações variam ao longo do tempo.

Os testes foram realizados em ambiente interno com temperatura controlada, evitando-se variações significativas que pudessem afetar a velocidade de propagação do som no ar e, consequentemente, as medições do sensor ultrassônico.

5. RESULTADOS

5.1 VALIDAÇÃO EXPERIMENTAL DO CONTROLE PID

5.1.1 Síntese dos Testes Realizados

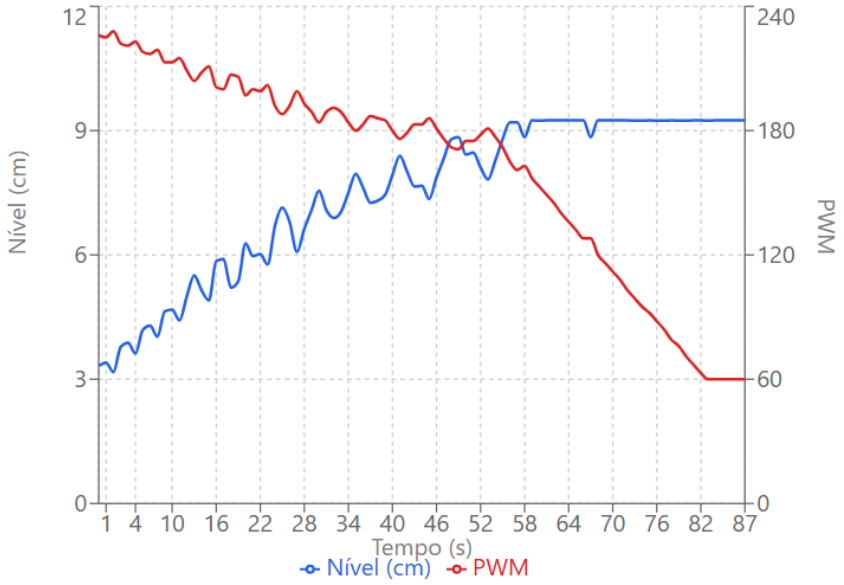
Foram conduzidos 13 experimentos sistemáticos testando três configurações distintas de parâmetros PID (Kp, Ki, Kd) sob quatro condições de perturbação (0%, 20%, 50% e 100% de vazão de saída), além de teste de robustez com vazões alternadas. O *setpoint* estabelecido foi de 9,0 cm para todos os testes, permitindo comparação direta entre configurações.

5.1.2 Resultados Principais

5.1.2.1 Configuração 1 (Kp=10,0 | Ki=1,7 | Kd=0)

Demonstrou resposta lenta, mas estável, adequada apenas para condições de baixa perturbação. Apresentou erro crescente com aumento da vazão, atingindo -34,78% (ou seja, faltando água) sob vazão máxima.

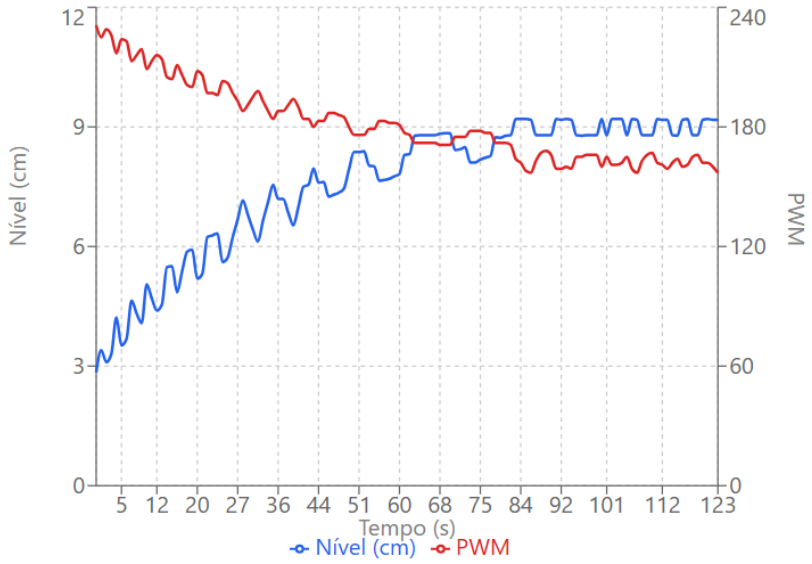
Gráfico 1 - Resposta a config.1 sem perturbação.



Fonte: Autoria própria (2025).

Como pode ser observado no Gráfico 1, sem perturbação, o sistema partiu de um nível inicial de aproximadamente 3,32 cm e alcançou o *setpoint* de 9,0 cm em aproximadamente 33 segundos. Durante a fase inicial, o controlador aplicou PWM máximo (226-228), decrescendo gradualmente conforme o nível se aproximava do setpoint. Observou-se resposta monotônica sem sobressinal significativo, característica típica de sistemas subamortecidos com ganhos conservadores. Segundo Ogata (2010), a ausência de *overshoot* indica fator de amortecimento elevado, embora à custa de maior tempo de resposta.

Gráfico 2- Resposta a config.1 com 20% de perturbação.

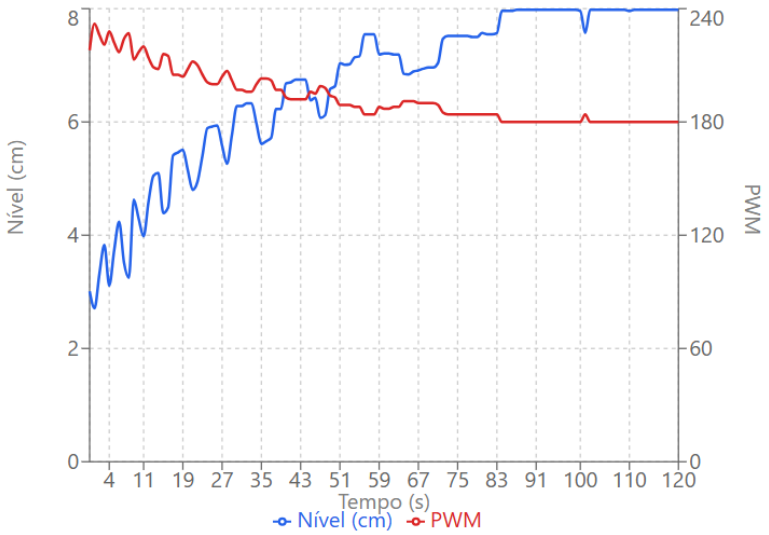


Fonte: Autoria própria (2025).

A introdução das vazões de saída de 20% e 50%, nos Gráfico 2 e Gráfico 3, representa uma perturbação constante no sistema. O tempo de subida aumentou para aproximadamente 44 segundos, refletindo o efeito da perturbação. O sistema em ambos os casos de perturbação alcançou o *setpoint* mas apresentou oscilações de baixa amplitude em torno do valor desejado.

Com 20% de perturbação, o nível estabilizou entre 8,79 cm e 9,20 cm (erro médio de -1,11% a +2,22%), com PWM oscilando entre 157 e 172. Com 50%, O nível final estabilizou em aproximadamente 7,98 cm, representando erro de -11,33% em relação ao *setpoint*. Estas oscilações, segundo Dorf e Bishop (2016), são características de sistemas com ganho integral insuficiente para eliminar completamente o erro frente a perturbações contínuas, resultando em comportamento de "*hunting*" - ajustes contínuos em busca do *setpoint*.

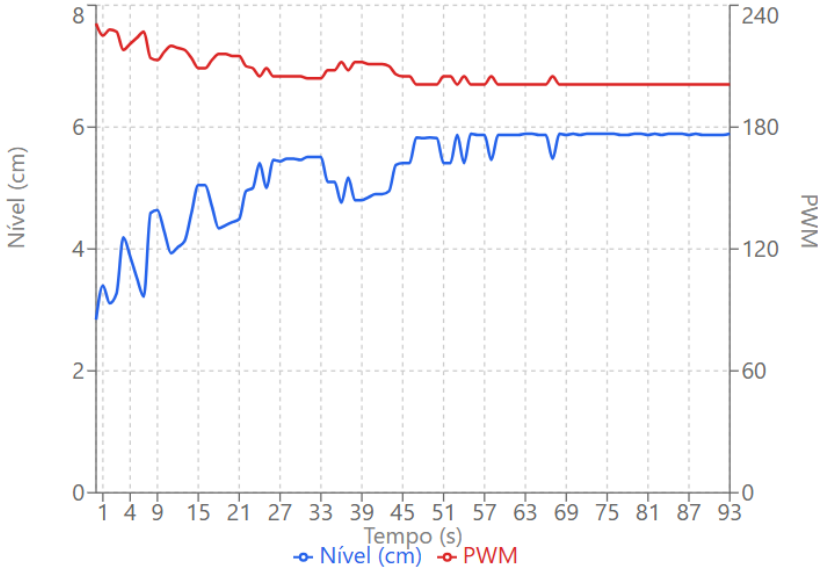
Gráfico 3 - Resposta a config.1 com 50% de perturbação.



Fonte: Autoria própria (2025).

Na condição de máxima perturbação (torneira totalmente aberta), a configuração 1 revelou sua principal limitação: incapacidade de atingir o *setpoint* estabelecido Gráfico 4. O sistema estabilizou em aproximadamente 5,87 cm, correspondendo a erro de -34,78%. Este resultado indica que, com os parâmetros Kp=10,0 e Ki=1,7, o esforço de controle gerado (PWM=201) foi insuficiente para compensar a vazão de saída máxima. Segundo Nise (2017), quando a perturbação excede a capacidade de atuação do sistema, o erro em regime permanente torna-se inevitável, independentemente dos ganhos do controlador.

Gráfico 4- Resposta a config.1 com 100% de perturbação.



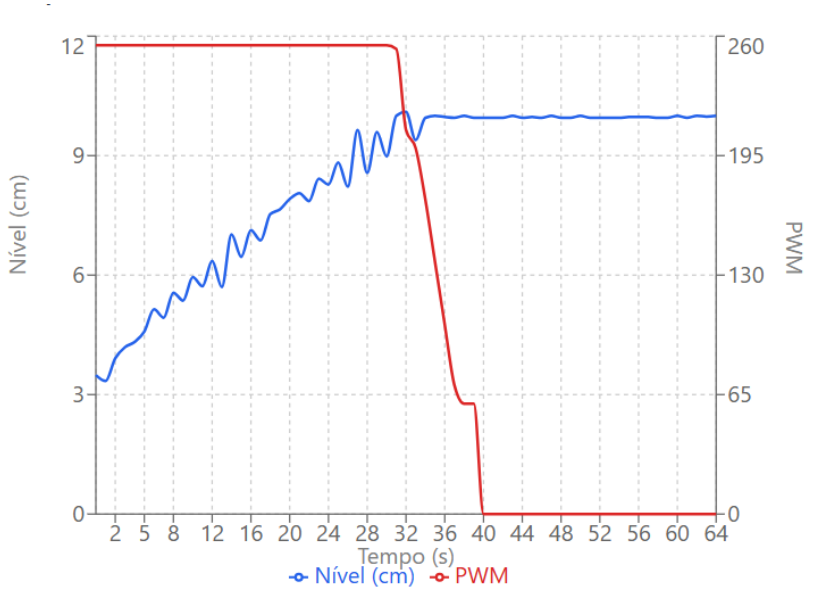
Fonte: Autoria própria (2025).

5.1.2.2 Configuração 2 (Kp=15,0 | Ki=3,5 | Kd=0)

Apresentou resposta rápida e boa rejeição de perturbações, conseguindo manter o *setpoint* mesmo em vazão de 100%. Entretanto, exibiu oscilações sustentadas indesejáveis (±0,8 cm), comprometendo estabilidade.

Com a torneira estando fechada, o sistema demonstrou resposta significativamente mais rápida que a Configuração 1, Gráfico 5, alcançando o *setpoint* em aproximadamente 31 segundos (redução de ~6% no tempo). Durante toda a fase transitória, o controlador manteve PWM=255 (máximo), refletindo o ganho proporcional mais elevado. Observou-se sobressinal de aproximadamente 11,11% (pico de 10,0 cm contra *setpoint* de 9,0 cm aos 37 segundos), seguido de rápida correção. Este comportamento é esperado em sistemas com ganhos mais agressivos, conforme descrito por Dorf e Bishop (2016).

Gráfico 5- Resposta a config.2 sem perturbação.

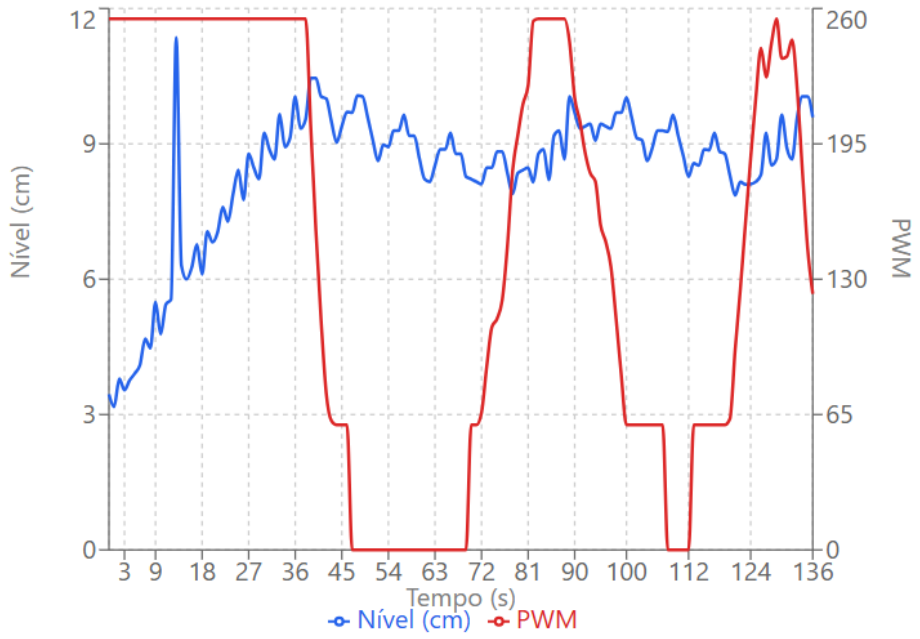


Fonte: Autoria própria (2025).

A resposta sob perturbações de 20% e 50%, revelou características importantes desta configuração. O tempo de subida reduziu em relação à Configuração 1 nas mesma condições. Entretanto, o sistema apresentou comportamento oscilatório mais pronunciado. Após atingir o setpoint, observaram-se oscilações sustentadas com amplitude de aproximadamente ±0,8 cm em torno do valor desejado, como pode ser observado no Gráfico 6 e Gráfico 7. O PWM variou ciclicamente entre valores amplos (0 a 255), indicando ação de controle agressiva.

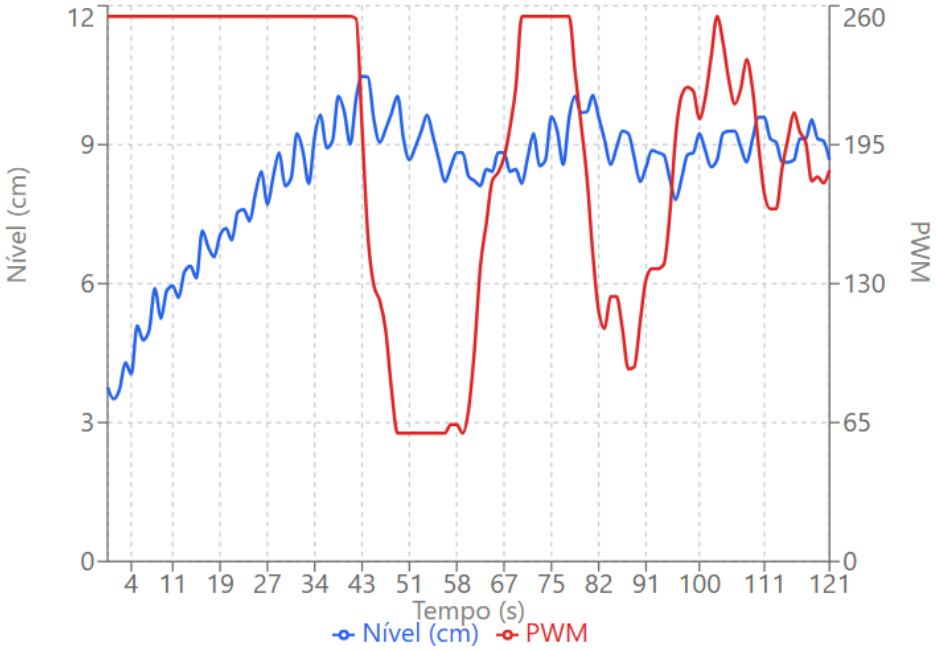
Este comportamento de "*limit cycle*" (ciclo limite), segundo Ogata (2010), é característico de sistemas com ganhos excessivos, onde o controlador sobrecompensa continuamente os desvios. Embora o erro médio seja pequeno, a oscilação contínua é indesejável em aplicações práticas.

Gráfico 6- Resposta a config.2 com 20% de perturbação.



Fonte: autoria própria (2025).

Gráfico 7- Resposta a config.2 com 50% de perturbação.

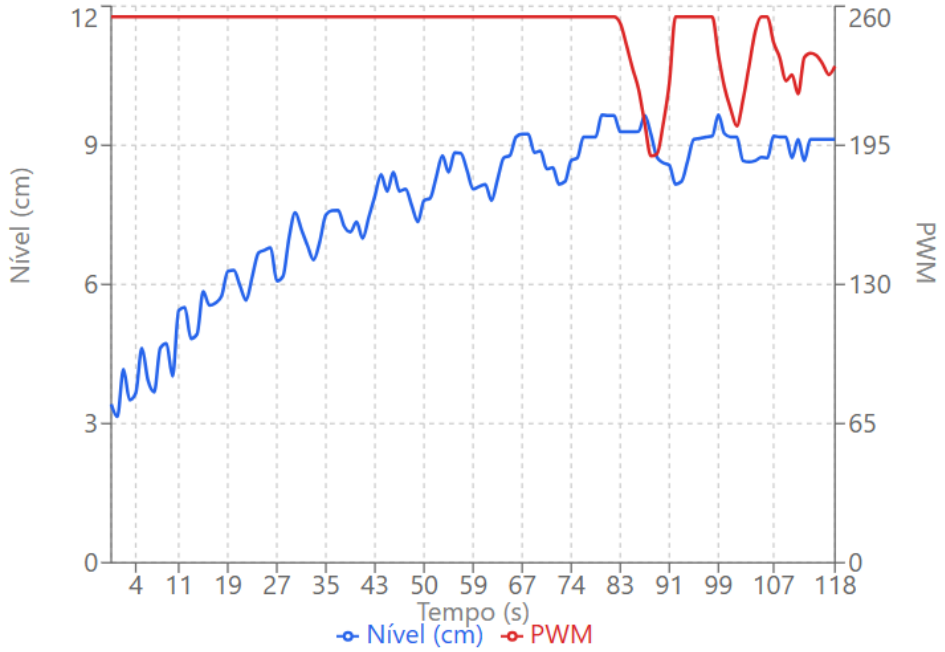


Fonte: autoria própria (2025).

Na condição de perturbação máxima no Gráfico 8, a Configuração 2 conseguiu aproximar-se mais do *setpoint* que a Configuração 1, estabilizando em torno de 8,7-9,2 cm (erro

de -3,33% a +2,22%). Esta melhoria significativa demonstra o efeito benéfico do aumento do ganho integral na rejeição de perturbações constantes. Contudo, o comportamento oscilatório permaneceu presente, com PWM variando entre 204 e 255. As oscilações, embora de menor amplitude que em vazões menores, indicam margem de estabilidade reduzida.

Gráfico 8- Resposta a config.2 com 100% de perturbação.



Fonte: autoria própria (2025).

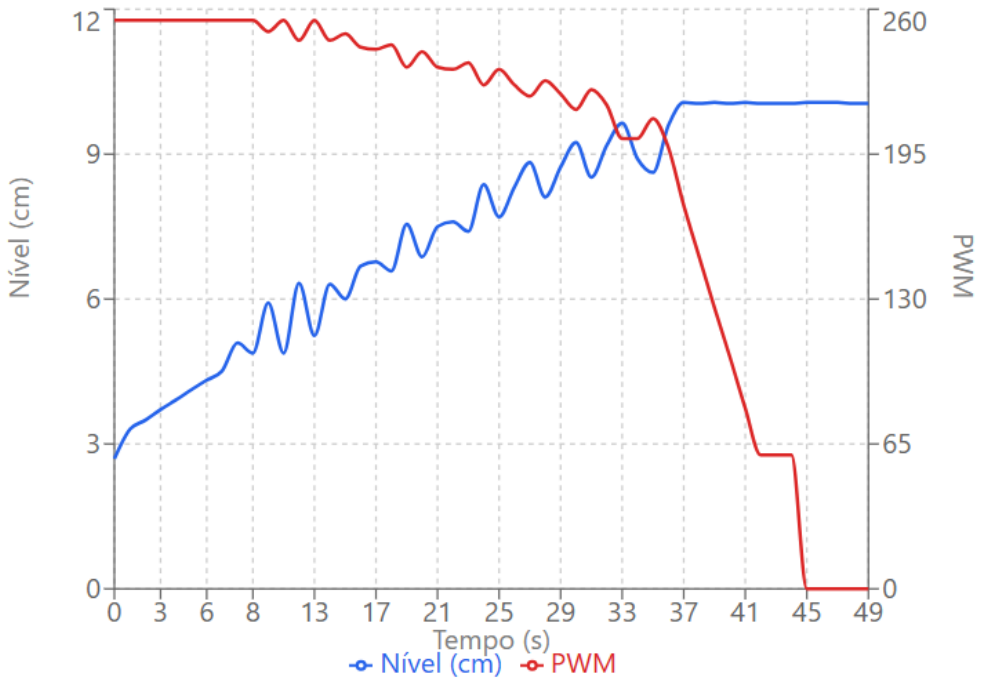
5.1.2.3 Configuração 3 (Kp=10,0 | Ki=2,2 | Kd=0,5)

Identificada como a melhor das três, a configuração 3 alcançou equilíbrio superior entre velocidade de resposta e estabilidade. A introdução da ação derivativa (Kd=0,5) eliminou oscilações sustentadas, resultando em comportamento suave e previsível. Manteve erro inferior a ±5% para vazões até 50%.

O sistema apresentou excelente desempenho, combinando rapidez e estabilidade, Gráfico 9. O *setpoint* foi alcançado em aproximadamente 43 segundos, com sobressinal muito pequeno (~11,89% ou 10,07 cm) rapidamente amortecido. A ação derivativa manifestou-se claramente na fase de aproximação ao setpoint: o PWM foi reduzido gradualmente de 255 para

valores entre 60-172 conforme o nível se aproximava de 9,0 cm, demonstrando ação antecipatória característica do termo derivativo. Esta modulação suave, conforme explicam Åström e Hägglund (1995), evita sobressinal excessivo ao "frear" preventivamente o sistema.

Gráfico 9 - Resposta a config.3 sem perturbação.

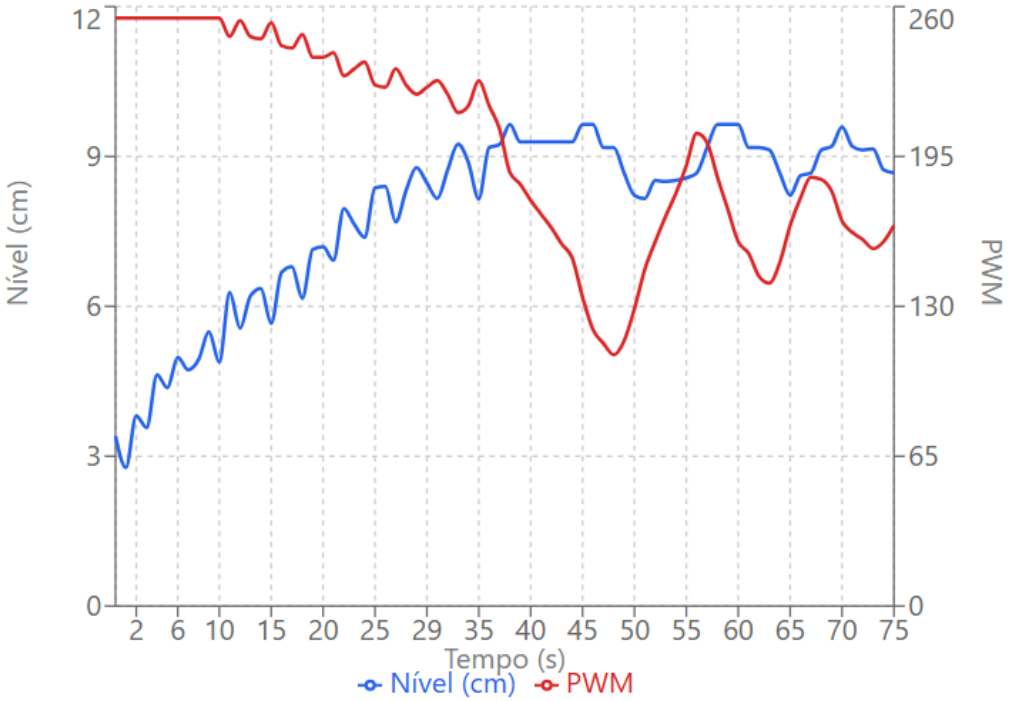


Fonte: Autoria própria (2025).

Com perturbações de 20% e 50%, a Configuração 3 demonstrou comportamento superior que pode ser visto nos Gráfico 10 e Gráfico 11. O tempo de subida foi de aproximadamente 42 segundos, comparável à Configuração 2 mas com resposta muito mais estável. O sistema alcançou o *setpoint* com sobressinal mínimo (~7,11% ou 9,64 cm) e estabilizou rapidamente. Em regime permanente, observou-se comportamento excepcional: o nível oscilou suavemente entre 9,13 e 9,64 cm (erro de +1,44% a +7,11%), com PWM variando moderadamente entre 143 e 205.

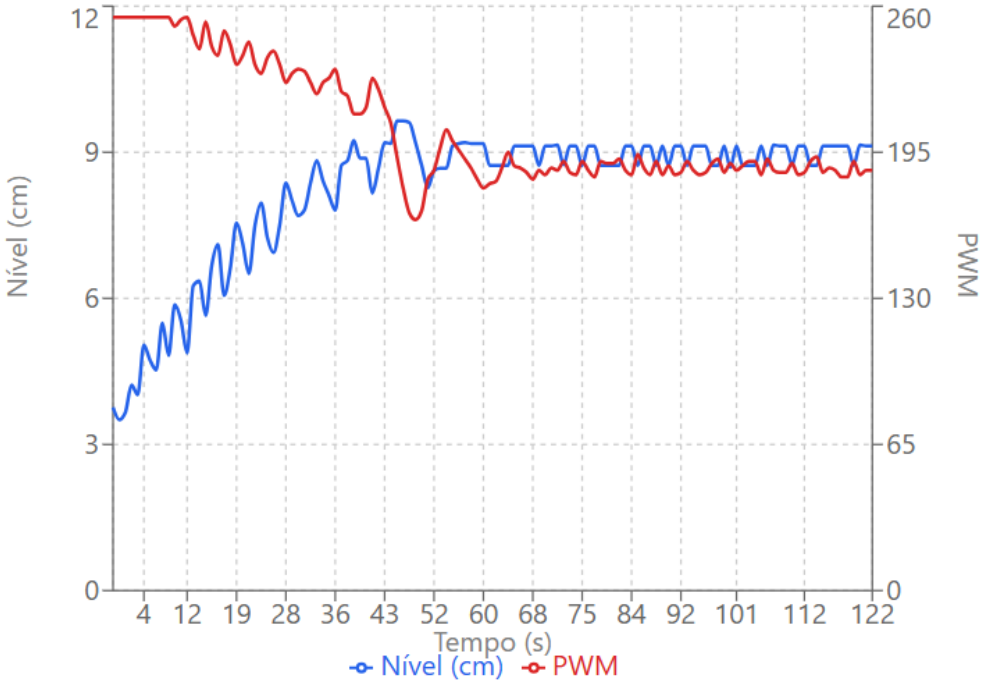
A diferença fundamental em relação à Configuração 2 foi a ausência de oscilações bruscas. O termo derivativo, segundo Nise (2017), proporcionou amortecimento adicional, resultando em ajustes suaves e graduais. Este comportamento é ideal para aplicações práticas, minimizando desgaste mecânico e consumo energético.

Gráfico 10 - Resposta a config.3 com 20% de perturbação.



Fonte: Autoria própria (2025).

Gráfico 11 - Resposta a config.3 com 50% de perturbação.

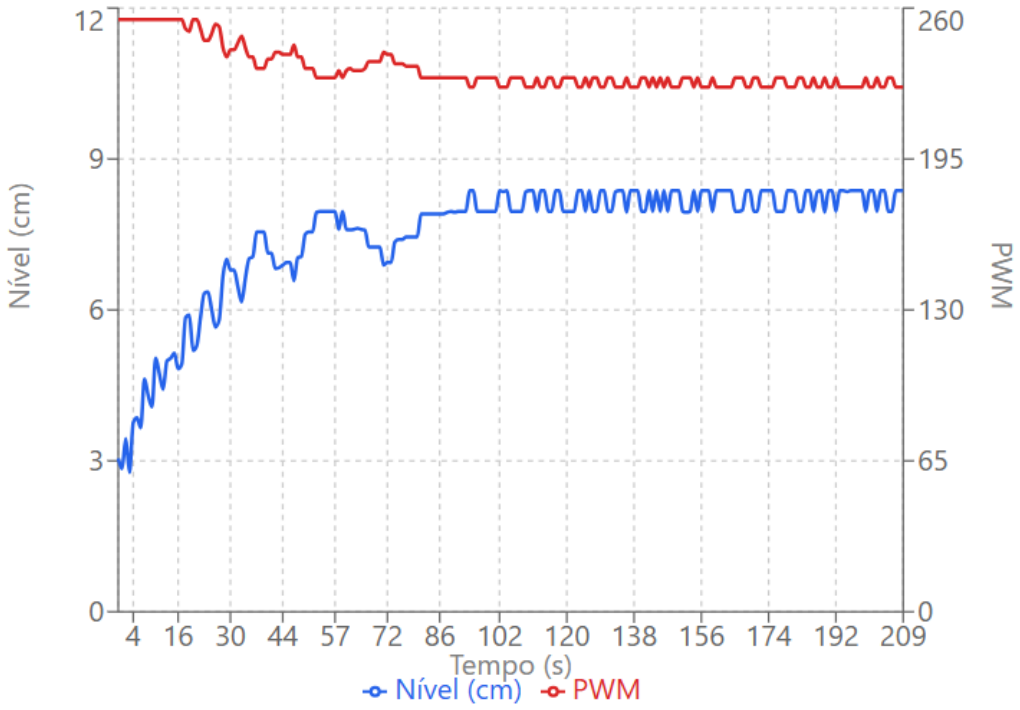


Fonte: autoria própria (2025).

Na condição de perturbação máxima, a Configuração 3 apresentou resultado notável e inesperado: o sistema estabilizou não no *setpoint* de 9,0 cm, Gráfico 12, mas em aproximadamente 7,96-8,37 cm (erro de -11,56% a -7,00%). Este comportamento difere das expectativas teóricas. A análise dos dados revela que o PWM se manteve em 226-230, valor inferior ao máximo disponível (255). Isto sugere que, diferentemente da Configuração 2 que aplicava PWM=255 intermitentemente, a Configuração 3 adotou estratégia mais conservadora devido à ação derivativa.

Em regime permanente com erro constante, o termo derivativo é zero, e o controle depende principalmente das ações proporcional e integral. O ganho integral Ki=2,2, embora superior ao da Configuração 1, mostrou-se insuficiente para gerar PWM=255, necessários para vencer a perturbação máxima.

Gráfico 12 - Resposta a config.3 com 100% de perturbação.



Fonte: Autoria própria (2025).

Para o último teste, de perturbação variável, o sistema demonstrou notável capacidade de adaptação às mudanças de perturbação, Gráfico 13. Cada transição resultou em breve período transitório (~10-15 segundos) seguido de nova estabilização.

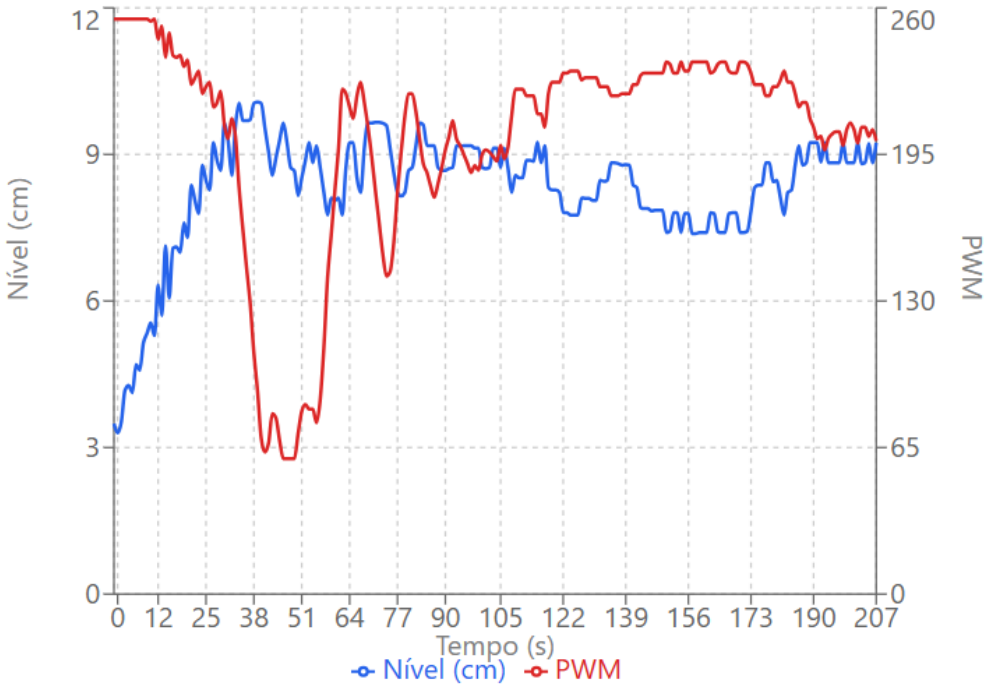
Fase 1 (0-45s): Torneira fechada → 20% O nível alcançou ~9,0 cm. Ao introduzir vazão de 20% (t≈45s), observou-se queda momentânea seguida de recuperação para ~8,8-9,2 cm.

Fase 2 (45-90s): 20% → 50% Aumento da perturbação resultou em nova queda para ~8,7-9,1 cm. O controlador respondeu aumentando PWM gradualmente, estabilizando em novo ponto de equilíbrio.

Fase 3 (90-150s): 50% → 100% Transição para perturbação máxima causou maior impacto, com nível caindo para ~7,4-8,8 cm, confirmando limitação já observada no teste dedicado de 100% de vazão.

Fase 4 (150-207s): Estabilização em condição variável Sistema manteve operação estável apesar das variações, com PWM ajustando-se continuamente entre 180 e 236.

Gráfico 13 - Resposta a config.3 com perturbação variável.



Fonte: Autoria própria (2025).

Este teste demonstra que a Configuração 3 é robusta frente a perturbações variáveis, característica essencial em aplicações reais. Segundo Åström e Hägglund (1995), controladores que mantêm estabilidade sob perturbações variáveis apresentam elevada margem de

estabilidade e são adequados para implementação prática. A capacidade de adaptação sem necessidade de ressintonia é vantagem significativa, validando a escolha desta configuração como mais apropriada para o sistema desenvolvido.

Tabela 6 - Comparativo entre as configurações.

Configuração	Resposta	Estabilidade	Erro (0-50%)	Erro (100%)	Avaliação
1	Lenta	Excelente	±2-11%	-35%	Adequada
2	Rápida	Ruim	Oscilante	-1%	Instável
3	Adequada	Excelente	±2-5%	-9%	Melhor

Fonte: Autoria própria (2025).

A investigação experimental de três configurações PID aplicadas ao sistema de controle de nível em tanques acoplados revelou que:

1. **A Configuração 3 (K_p=10,0 | K_i=2,2 | K_d=0,5) foi identificada como a melhor das três**, apresentando o melhor equilíbrio entre velocidade de resposta, estabilidade e rejeição de perturbações, Tabela 6;
2. **A ação derivativa foi fundamental** para alcançar desempenho superior, reduzindo sobressinal e eliminando oscilações sustentadas observadas nas configurações PI puras, Tabela 6;
3. **O sistema demonstrou robustez adequada** para operação em condições variáveis (teste de vazões alternadas), validando a escolha dos parâmetros;
4. **As limitações identificadas** (erro de ~9% em perturbação máxima) são atribuíveis principalmente à saturação do atuador, não a deficiências do algoritmo de controle, Tabela 6;
5. **A margem de erro de 7%** observada em condições normais de operação é aceitável considerando as características do *hardware* utilizado (sensor HC-SR04, bomba de baixo custo);
6. **O método de sintonia por tentativa e erro**, embora demandante em tempo, demonstrou-se eficaz e resultou em parâmetros superiores aos que seriam obtidos por métodos analíticos aplicados a modelo simplificado.

5.2 Resultado do Experimento do Sistema de Controle

O presente trabalho resultou no desenvolvimento completo de um sistema de controle automático de nível em tanques acoplados, integrando *hardware*, *software* e teoria de controle. O sistema construído representa uma plataforma experimental funcional que demonstra a aplicabilidade prática de conceitos fundamentais de engenharia de controle e automação.

5.2.1 Integração *Hardware-Software*

A implementação bem-sucedida do sistema evidenciou a viabilidade de utilizar microcontroladores de baixo custo, especificamente o ESP32, para aplicações de controle em tempo real. Segundo Kurniawan (2018), o ESP32 oferece capacidade de processamento adequada para algoritmos de controle digital, o que foi confirmado pela execução estável do controlador PID com frequência de amostragem de aproximadamente 1 Hz.

A integração entre os componentes físicos (sensor HC-SR04, bomba d'água, ponte H L298N) e o *software* embarcado demonstrou que sistemas de automação eficazes podem ser desenvolvidos com tecnologias acessíveis, reduzindo significativamente o custo em comparação a soluções industriais convencionais. Esta característica é particularmente relevante para aplicações educacionais e de pesquisa.

5.2.2 Plataforma Experimental para Ensino

Um dos resultados mais significativos deste trabalho é a criação de uma plataforma experimental completa para estudo de sistemas de controle. O sistema desenvolvido oferece características ideais para propósitos didáticos:

a) Visualização direta: O uso de tanques transparentes permite observação visual imediata da resposta do sistema, facilitando a compreensão intuitiva de conceitos como tempo de resposta, sobressinal e estabilidade;

b) Ajuste de perturbações: A torneira com abertura variável possibilita simular diferentes condições operacionais, permitindo estudar rejeição de perturbações e robustez do controlador;

c) Plataforma de programação acessível: O uso da IDE Arduino e da linguagem C/C++ torna o sistema acessível a estudantes com conhecimentos básicos de programação;

d) Baixo custo: O custo total do sistema (aproximadamente R$ 300-400) é compatível com orçamentos educacionais, permitindo replicação em laboratórios de ensino;

e) Segurança: Operação com tensões baixas (12V) e água à temperatura ambiente garante segurança durante experimentação.

Segundo Monk (2014), plataformas práticas de experimentação são fundamentais para consolidação do aprendizado em engenharia, permitindo que conceitos teóricos sejam vivenciados e compreendidos de forma concreta.

6. CONCLUSÃO

O desenvolvimento deste trabalho proporcionou experiência prática abrangente integrando conhecimentos de diversas áreas: teoria de controle, programação embarcada, eletrônica, instrumentação e metodologia experimental. A transição do conhecimento teórico adquirido em sala de aula para aplicação prática em sistema físico real foi desafiadora, mas extremamente enriquecedora.

Os objetivos estabelecidos foram plenamente atingidos: desenvolveu-se um sistema funcional de controle automático, validou-se experimentalmente a eficácia do controle PID, identificou-se configuração ótima de parâmetros e criou-se plataforma educacional permanente para uso futuro.

A margem de erro de 7% observada em condições normais de operação é aceitável considerando o custo e complexidade do sistema, demonstrando que soluções eficazes não necessariamente requerem investimentos elevados. Este resultado é particularmente relevante para contextos educacionais e de pesquisa com recursos limitados.

O sistema construído permanecerá disponível para utilização por estudantes futuros, cumprindo função educacional duradoura. A documentação completa elaborada facilita a compreensão, replicação e modificação do projeto, maximizando seu impacto educacional.

Por fim, este trabalho demonstra que a engenharia de controle e automação, pode ser concretizada em aplicações práticas tangíveis utilizando ferramentas acessíveis. O conhecimento adquirido, as dificuldades superadas e os resultados alcançados constituem

contribuição valiosa tanto para o desenvolvimento pessoal do autor quanto para a comunidade acadêmica, consolidando o aprendizado em engenharia através do ciclo completo de concepção, desenvolvimento, implementação e validação experimental de sistema de controle automatizado.

REFERÊNCIAS

ÅSTRÖM, K. J.; HÄGGLUND, T. **PID Controllers: Theory, Design, and Tuning**. 2. ed. Research Triangle Park: Instrument Society of America, 1995.

BARRY, R. **Mastering the FreeRTOS Real Time Kernel: A Hands-On Tutorial Guide**. London: Real Time Engineers Ltd., 2018.

BAZANELLA, A. et al. Controlador PID paralelo em um sistema RTOS. 2024. Disponível em:https://repositorio.animaeducacao.com.br/bitstreams/769e7649-5fc0-46e1-b1b9-f4cf90a106d1/download. Acesso em: 05 nov. 2025.

COHEN, G. H.; COON, G. A. **Theoretical consideration of retarded control**. Transactions of the ASME, v. 75, p. 827-834, 1953.

DORF, R. C.; BISHOP, R. H. **Sistemas de Controle Modernos**. 13. ed. Rio de Janeiro: LTC, 2016.

ELETROGATE. Guia Definitivo de uso da Ponte H L298N. Disponível em: https://blog.eletrogate.com/guia-definitivo-de-uso-da-ponte-h-l298n/. Acesso em: 01 dez. 2025.

ESPRESSIF SYSTEMS COMPANY. ESP32 technical reference manual. 2023. Disponível em:https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf. Acesso em: 05 nov. 2025.

FRANKLIN, G. F.; POWELL, J. D.; EMAMI-NAEINI, A. **Sistemas de Controle para Engenharia**. 6. ed. Porto Alegre: Bookman, 2015.

FERNANDES, H. B. **Controle de velocidade em uma linha de ar. 2022**. Trabalho de Conclusão de Curso, Instituto Federal. Disponível em: https://repositorio.ifg.edu.br/bitstream/prefix/1252/1/tcc_Henrique%20Borges%20Fernandes.pdf. Acesso em: 05 nov. 2025.

KURNIAWAN, A. **Internet of Things Projects with ESP32**. Birmingham: Packt Publishing, 2018.

MELO, G. A. F.; BERNARDES, M. C. **Instrumentação e controle de uma maquete de nível de líquidos. 2020**. Disponível em: https://www.ene.unb.br/adolfo/Monographs/Graduation/TG06%20Gustavo%20A.F.%20Melo%20e%20Mariana%20C.%20Bernardes.pdf. Acesso em: 05 nov. 2025.

MIKE MONTEIRO, A. C. L. **Aplicação de um controlador PID em redes automatizadas de distribuição de água**. 2016. Disponível em: https://www.ibeas.org.br/congresso/Trabalhos2016/X-007.pdf. Acesso em: 05 nov. 2025.

MONK, S. **Programming the ESP32: Learn MicroPython Coding and Electronics**. New York: McGraw-Hill Education, 2022.

MONTEIRO, A. C. L. **Modelagem e controle de sistemas hidráulicos. Rio de Janeiro: Editora UFRJ**, 2016.

NICOLOSI, D. E. C. **Microcontrolador 8051: Detalhado**. São Paulo: Érica, 2002.

NISE, N. S**. Control systems engineering**. 2012. 6. ed. Wiley.

O'DWYER, A. **Handbook of PI and PID Controller Tuning Rules**. 3. ed. London: Imperial College Press, 2009.

OGATA, K. **Engenharia de Controle Moderno**. 5. ed. São Paulo: Pearson, 2011.

PEREIRA, F. **Microcontroladores PIC: Programação em C**. São Paulo: Érica, 2003.

SENEVIRATNE, **Hands-On Internet of Things with Blynk**. Birmingham: Packt Publishing, 2018.

SENEVIRATNE, P. **Practical Internet of Things with JavaScript: Build standalone exciting IoT projects with Raspberry Pi 3 and JavaScript (ES5/ES6)**. Birmingham: Packt Publishing, 2020.

SKOGESTAD, S. **Simple analytic rules for model reduction and PID controller tuning**. Journal of Process Control, v. 13, n. 4, p. 291-309, 2003.

UPESY. ESP32 Pinout: How use GPIO pins? 2023. Disponível em: https://www.upesy.com/blogs/tutorials/esp32-pinout-reference-gpio-pins-ultimate-guide. Acesso em: 02 dez. 2025.

ZIEGLER, J. G.; NICHOLS, N. B. **Optimum settings for automatic controllers**. Transactions of the ASME, v. 64, n. 11, p. 759-765, 1942.

ANEXOS

ANEXO A

CÓDIGO DE PROGRAMAÇÃO DO ESP32 – PROTÓTIPO

```
#include "BluetoothSerial.h"

// Verifica se o Bluetooth foi ativado
#if !defined(CONFIG_BT_ENABLED) || !defined(CONFIG_BLUEDROID_ENABLED)
#error Bluetooth is not enabled! Please run `make menuconfig` to enable it.
#endif

BluetoothSerial SerialBT;
// --- Definição dos Pinos ---
#define PIN_TRIG 5
#define PIN_ECHO 18
#define PIN_ENA  23  // Pino PWM da Ponte H
#define PIN_IN1  22  // Controle de direção 1
#define PIN_IN2  21  // Controle de direção 2

// --- Parâmetros do Tanque ---
const double ALTURA_TANQUE = 18.5; // cm (Altura total do sensor até o fundo)
const double SETPOINT = 9.0;       // cm (Nível de água desejado)

// --- Variáveis PID ---
double Kp = 10.0;  // Ganho Proporcional (Ajuste isso!)
double Ki = 2.2;   // Ganho Integral (Ajuste isso!)
double Kd = 0.5;   // Ganho Derivativo (Ajuste isso!)
double erro = 0;
double erro_anterior = 0;
double integral = 0;
double derivada = 0;
double saida_pid = 0;

// --- VARIÁVEIS GLOBAIS DE MEDIÇÃO E SAÍDA ---
// Declaradas aqui para que o bloco de impressão possa acessá-las
double nivel_atual = 0;
int pwm_output = 0;

// --- TEMPOS (DOIS TEMPORIZADORES INDEPENDENTES) ---
// 1. Tempo para o PID (Rápido)
unsigned long lastTime = 0;
unsigned long sampleTime = 100; // PID roda a cada 100ms (10 vezes por segundo)

// 2. Tempo para a Impressão (Lento)
unsigned long lastPrintTime = 0;
unsigned long printInterval = 1000; // Impressão roda a cada 1000ms (1 vez por segundo)

// Configuração do PWM
const int freq = 5000;
const int channel = 0;
const int resolution = 8;

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

  // INICIALIZAÇÃO DO BLUETOOTH
  SerialBT.begin("ESP32_PID_Nivel");
```

```
  Serial.println("Bluetooth Serial iniciado.");

  // Configuração dos Pinos
  pinMode(PIN_TRIG, OUTPUT);
  // ... (restante dos pinos) ...
  pinMode(PIN_ECHO, INPUT);
  pinMode(PIN_IN1, OUTPUT);
  pinMode(PIN_IN2, OUTPUT);

  // Configuração do PWM (LEDC)
  ledcSetup(channel, freq, resolution);
  ledcAttachPin(PIN_ENA, channel);

  // Define a direção da bomba
  digitalWrite(PIN_IN1, HIGH);
  digitalWrite(PIN_IN2, LOW);

  Serial.println("Iniciando Controle PID de Nível...");
}

void loop() {
  unsigned long now = millis();

  // -----------------------------------------------------
  // 1. LOOP DE CONTROLE PID (Roda a cada 100ms)
  // -----------------------------------------------------
  if (now - lastTime >= sampleTime) {

    // O CALCULO ATUALIZA AS VARIÁVEIS GLOBAIS 'nivel_atual' e 'pwm_output'
    nivel_atual = lerNivelAgua();

    // Cálculo do PID
    erro = SETPOINT - nivel_atual;
    integral += erro;
    if (integral > 100) integral = 100;
    if (integral < -100) integral = -100;
    derivada = erro - erro_anterior;
    saida_pid = (Kp * erro) + (Ki * integral) + (Kd * derivada);

    // Conversão e Limites do PWM (Atualiza a variável global pwm_output)
    pwm_output = (int)saida_pid;
    if (pwm_output > 255) pwm_output = 255;
    if (pwm_output < 0) pwm_output = 0;

    // Zona Morta (Deadzone)
    if (pwm_output > 0 && pwm_output < 60) {
       pwm_output = 60;
    }

    // Aplicação na Ponte H
    ledcWrite(channel, pwm_output);

    // Atualizar variáveis
    erro_anterior = erro;
    lastTime = now;
  }

  // -----------------------------------------------------
  // 2. LOOP DE IMPRESSÃO (Roda a cada 1000ms / 1s)
  // -----------------------------------------------------
```

```
  if (now - lastPrintTime >= printInterval) {

    // Formato de saída de dados
    String dataOutput = "Nivel: " + String(nivel_atual, 2) + " cm | " +
                        "PWM: " + String(pwm_output) + " | " +
                        "Erro: " + String(erro, 2);

    // Saída para o Monitor Serial USB
    Serial.println(dataOutput);

    // Saída para o Celular via Bluetooth
    if (SerialBT.hasClient()) {
      SerialBT.println(dataOutput);
    }

    // Atualiza o tempo da última impressão
    lastPrintTime = now;
  }
}

// Funções Auxiliares (Não modificadas)
double lerNivelAgua() {
  digitalWrite(PIN_TRIG, LOW);
  delayMicroseconds(2);
  digitalWrite(PIN_TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(PIN_TRIG, LOW);

  long duration = pulseIn(PIN_ECHO, HIGH, 30000);

  if (duration == 0) return nivel_atual; // Se falhar, usa o último nível conhecido

  double distancia_cm = duration * 0.034 / 2;

  double nivel = ALTURA_TANQUE - distancia_cm;

  if (nivel < 0) nivel = 0;

  return nivel;
}
```