UNIVERSIDADE FEDERAL DE ITAJUBÁ

Austin David Roldan Urquiza

Beatriz Helena Moia Damião

Fabio Franco Ferreira

Gabriel Almeida Castro

Maria luisa Barbosa Dos Santos

Projeto Final - ECAE00

Sistema de Monitoramento e Controle Preventivo de Enchentes

Austin David Roldan Urquiza - 2025008610

Beatriz Helena Moia Damião - 2025004999

Fabio Franco Ferreira - 2025017057

Gabriel Almeida Castro - 2025008728

Maria luisa Barbosa Dos Santos - 2025016130

Projeto Final - ECAE00

Sistema de Monitoramento e Controle Preventivo de Enchentes

Projeto de Controle e Automação com ESP32, apresentado como requisito parcial para aprovação na disciplina *Introdução à Engenharia e ao Método Científico – ECAE00*, do curso de Engenharia de Controle e Automação da Universidade Federal de Itajubá (UNIFEI).

Trabalho desenvolvido sob a orientação dos professores Luiz Lenarth G. Vermaas e Jeremias Barbosa Machado.

**SUMÁRIO**

[**1 INTRODUÇÃO 4**](#_heading=h.t4hvuut8rwgn)

[**2 OBJETIVOS 5**](#_heading=h.5zpzdqjd3t2e)

[**2.1 Objetivo Geral 5**](#_heading=h.18n7othhokz)

[**2.2 Objetivos Específicos 5**](#_heading=h.nycdhvp1gw86)

[**3 Pesquisa & Planejamento 6**](#_heading=h.aihjue6ooe58)

[**3.1 Lógica de funcionamento interno 6**](#_heading=h.49ep8c1plcxy)

[**3.2 Lógica da Interface do Utilizador 7**](#_heading=h.t90k0cguj1gr)

[**3.3 Diagrama de Blocos 7**](#_heading=h.e82b0e2fw6w3)

[**4 Desenvolvimento 8**](#_heading=h.tvakbrfnsjae)

[**4.1 Escolha dos componentes 8**](#_heading=h.60zzxivcjotf)

[**4.2 Montagem Física 9**](#_heading=h.cbtyn3awmvkk)

[**4.3 Pinagem do Sistema (Pinout) 10**](#_heading=h.o46pjmn20ulm)

[**4.4 Desenvolvimento do software 11**](#_heading=h.ghqooyinzilv)

[**4.4.1 Bibliotecas Utilizadas 11**](#_heading=h.r2pgk5gpgfws)

[**4.4.2 Funções Implementadas 13**](#_heading=h.rftvonr4sute)

[**4.4.3 Pinos Utilizados 13**](#_heading=h.srvaksb8ujs9)

[**4.4.4 Variáveis 13**](#_heading=h.rgk8gpo3s6nn)

[**4.4.5 Código Final 15**](#_heading=h.nr2vei95ma1)

[**5 Implementação & Testes 16**](#_heading=h.m14o737yvn5g)

[**5.1 Desafios Técnicos e Soluções 16**](#_heading=h.my2kmj3vrufx)

[**5.2 Maquete 16**](#_heading=h.pmgt0cqlm37g)

[**5.2.1 Visão frontal: 18**](#_heading=h.dt19866pk5j8)

[**5.2.2 Visão Superior: 19**](#_heading=h.90jukq89q5l0)

[**6 Conclusão 20**](#_heading=h.dmpo82k8i9zo)

[**7 Referências 21**](#_heading=h.w6xy03n49xv)

[**APÊNDICE A – CÓDIGO FONTE DO FIRMWARE 22**](#_heading=h.unqhs0f9krkf)

# INTRODUÇÃO

Este trabalho tem como objetivo documentar o desenvolvimento de um projeto de automação e controle focado na mitigação de riscos de enchentes, utilizando o microcontrolador ESP32 como unidade central de processamento. A proposta surge diante da necessidade crítica por sistemas de alerta precoce acessíveis e eficientes, capazes de monitorizar o nível de rios e acionar medidas preventivas de forma autónoma.

A solução proposta visa combinar monitorização ambiental, controlo de atuadores e uma interface de utilizador intuitiva para permitir a gestão dos parâmetros do sistema. O sistema foi projetado para operar em dois modos: um modo real, que utiliza um sensor para leituras ambientais, e um modo de simulação, que permite a demonstração controlada de todas as funcionalidades, garantindo a robustez da apresentação.

Para a prototipagem do sistema, foram selecionados componentes eletrónicos que permitem a medição de distância, o controlo proporcional de uma comporta, o acionamento de um alarme e a interação do utilizador via teclado e display.

* ESP32;
* Sensor Ultrassónico HC-SR04;
* Display LCD 16x2 com interface I2C;
* Keypad Matricial 4x4;
* Servo Motor SG90;
* Módulo de Relé;
* LED Vermelho 5mm;
* Resistor 330 Ohms;
* Protoboard;
* Jumpers (cabos de conexão)
* Fonte Ajustável para Protoboard 3.3V e 5V com interruptor / 700mA;
* Fonte de Alimentação Externa (Pack 4xAA)

# OBJETIVOS

# Objetivo Geral

Desenvolver um sistema de automação embarcado que integre a monitorização do nível de um corpo de água, o controle de uma comporta de vazão e o acionamento de um sistema de alarme, visando proporcionar uma solução de baixo custo para a prevenção de enchentes.

# Objetivos Específicos

* Realizar o levantamento dos requisitos e o planejamento das funcionalidades do sistema.
* Projetar e montar fisicamente o circuito eletrônico do protótipo em protoboard.
* Construir uma maquete representativa de um canal de rio com uma comporta móvel.
* Desenvolver o código de controle para o ESP32, integrando o sensor de nível e os atuadores (servo e relé).
* Implementar uma interface de utilizador (LCD e Keypad) para configuração e monitorização dos parâmetros do sistema em tempo real.
* Realizar testes funcionais e implementar ajustes para otimização do desempenho e fiabilidade do sistema.

# Pesquisa & Planejamento

O desenvolvimento do projeto teve início com a definição do problema a ser resolvido: a criação de um sistema de baixo custo para o alerta e controle preventivo de enchentes. Após uma análise crítica das possibilidades e dos componentes disponíveis, a ideia-base foi validada com os professores orientadores, que sugeriram melhorias e incrementos. Com o escopo definido, a equipa iniciou o desenvolvimento da lógica de funcionamento do sistema.

# Lógica de funcionamento interno

A primeira etapa envolveu a implementação da lógica de monitorização e atuação. Nesta fase, foram definidos os três estados operacionais do sistema, baseados na distância medida pelo sensor ultrassónico, que simula o nível do rio numa maquete de 17cm:

* + - * Nível Seguro (distância > 12cm): O sistema permanece em estado de vigilância, com a comporta fechada (ângulo 0°) e o alarme desativado.
      * Nível de Atenção (12cm >= distância > 8cm): O sistema entra em modo preventivo. A comporta é aberta de forma proporcional à diminuição da distância, iniciando o escoamento controlado da água.
      * Nível de Perigo (distância <= 8cm): O nível crítico é atingido. A comporta é totalmente aberta para maximizar a vazão (ângulo 90°), e o alarme é acionado para alertar sobre o risco iminente.

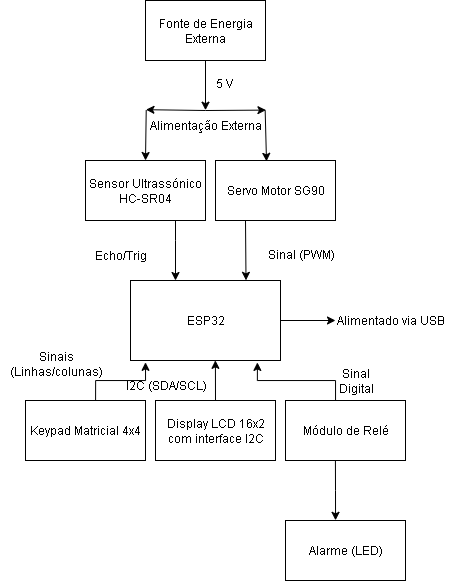
# Lógica da Interface do Utilizador

Com a lógica de controlo consolidada, a etapa seguinte foi o desenvolvimento da interface homem-máquina (IHM), composta pelo display LCD e pelo teclado matricial. O comportamento programado define que o utilizador pode visualizar o estado do sistema em tempo real (Modo, Distância, Ângulo da Comporta, Estado do Alarme). Crucialmente, a IHM permite a reconfiguração do

nivelAlerta através de um menu de configuração (\*), e a alternância entre os modos de operação REAL e SIMULAÇÃO (teclas A e B), dando total controlo e flexibilidade ao operador do sistema.

# Diagrama de Blocos

Para a demonstração simplificada das conexões, um diagrama de blocos baseado na lógica do projeto foi realizado, exibido abaixo:



# Desenvolvimento

O desenvolvimento prático do projeto teve início com a seleção criteriosa dos componentes eletrónicos, considerando as suas características técnicas e a sua compatibilidade com o microcontrolador ESP32. A alocação das portas foi feita de forma estratégica para garantir o funcionamento correto de todos os periféricos.

# Escolha dos componentes

* ESP32 (Microcontrolador):

Foi escolhido por ser o cérebro do projeto, fornecido no kit da disciplina. A sua capacidade de processamento e o número generoso de portas GPIO foram essenciais para controlar todos os sensores e atuadores simultaneamente.

* Sensor Ultrassónico HC-SR04:

Selecionado pela sua eficiência e baixo custo na deteção de distância, sendo ideal para monitorizar o nível da água na maquete.

* Servo Motor SG90:

Empregado como o atuador para a comporta. A sua capacidade de controlo angular preciso foi fundamental para implementar a lógica de abertura proporcional, um dos requisitos do projeto.

* Display LCD 16x2 com Interface I2C:

Utilizado como a principal interface visual para o utilizador. A sua integração via módulo I2C foi uma decisão estratégica para economizar portas no ESP32, necessitando de apenas dois fios para a comunicação.

* Keypad Matricial 4x4:

Implementado como a interface de entrada de dados, permitindo que o utilizador reconfigure parâmetros críticos do sistema, como o nível de alerta, e alterne entre os modos de operação, adicionando uma camada de interatividade e inteligência ao projeto.

* Módulo de Relé 5V:

Utilizado para o chaveamento do alarme. Este componente é crucial para a segurança do circuito, pois permite que um sinal de baixa corrente do ESP32 controle um circuito de maior potência (o LED do alarme), mantendo os dois sistemas eletricamente isolados.

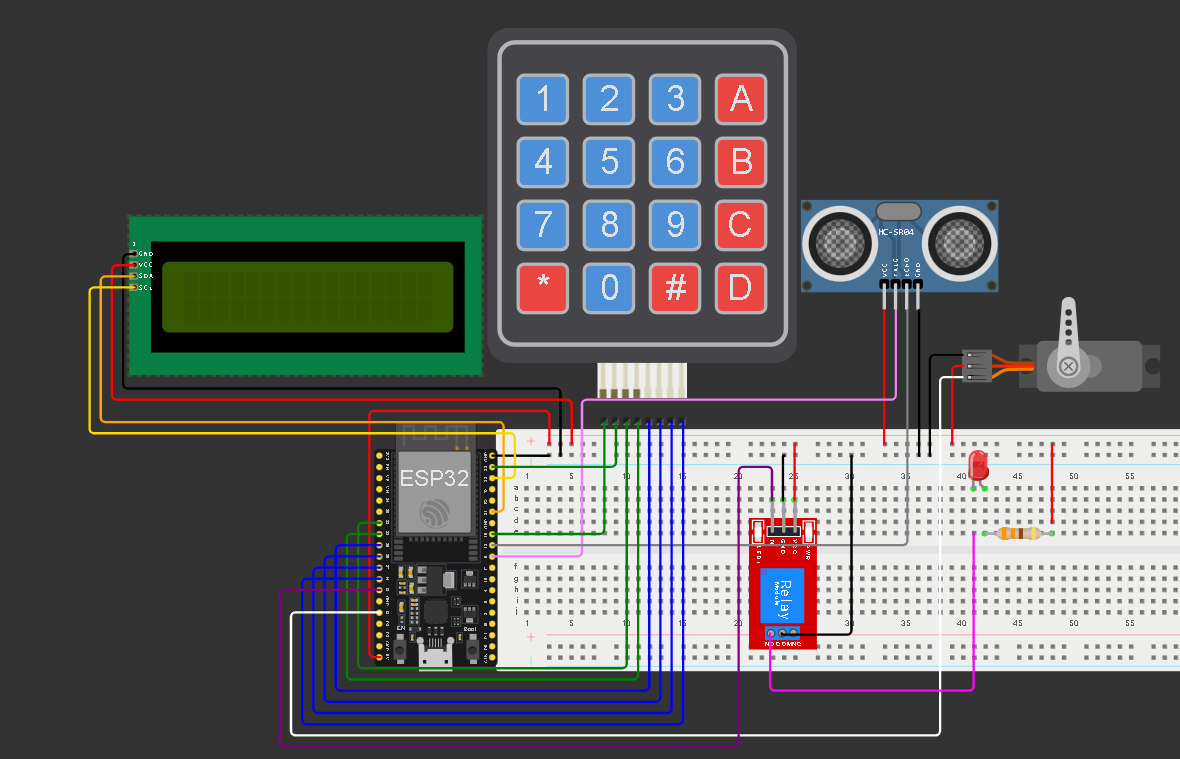
* LED e Resistor:

Utilizados para simular o sistema de alarme visual. O seu acionamento binário (ligado/desligado) serve como uma representação clara e imediata do estado de perigo.

# Montagem Física

A montagem do projeto foi realizada em duas etapas principais. A primeira consistiu na montagem do circuito completo em protoboard, com o objetivo de testar a integração de todos os componentes e validar a lógica do software. Esta fase foi essencial para a depuração de problemas, nomeadamente a instabilidade elétrica que foi solucionada com a adoção de uma fonte de alimentação externa e a correta ligação da "terra comum".

A seguir, uma representação utilizando a prataforma online Wokwi, que a única falta seria a fonte externa.



Cabe ressaltar que, apesar de a montagem apresentada ter um propósito puramente ilustrativo e didático, a validação de seu funcionamento foi feita de forma prática. Para isso, utilizamos um circuito físico real, com todos os componentes devidamente conectados ao microcontrolador ESP32 em uma protoboard. Esse método assegurou que os resultados obtidos fossem confiáveis e refletissem o desempenho do sistema em um cenário de uso real.

# Pinagem do Sistema (Pinout)

Para possibilitar a replicação precisa do projeto físico, segue abaixo a descrição detalhada das conexões entre os componentes e o microcontrolador ESP32.

* Servo Motor (SG90):

GND → GND (Fonte Externa)

VCC → 5V (Fonte Externa)

Sinal → GPIO 13

* Display LCD 16x2 (I2C):

GND → GND

VCC → 3.3V

SDA → GPIO 21

SCL → GPIO 22

* Sensor Ultrassónico (HC-SR04):

GND → GND (Fonte Externa)

VCC → 5V (Fonte Externa)

Trig → GPIO 5

Echo → GPIO 17

* Keypad 4x4:

Pinos de Linha → GPIO 19, 23, 32, 33

Pinos de Coluna → GPIO 25, 26, 27, 14

* Módulo de Relé:

GND → GND

VCC → 3.3V

IN → GPIO 12

* Alarme (LED + Resistor):

Pino 5V do ESP32 → Resistor 220Ω → Anodo (perna longa) do LED

Cátodo (perna curta) do LED → Terminal NO do Relé

Terminal COM do Relé → GND

Nota Crítica de Alimentação: É obrigatório o uso de uma fonte de alimentação externa para os componentes de potência (Servo e Sensor). O pino GND da fonte externa deve ser conectado a um pino GND do ESP32 para estabelecer uma "terra comum" e garantir a estabilidade do sistema.

# Desenvolvimento do software

O software (firmware) do sistema foi desenvolvido utilizando a linguagem C++ na plataforma Arduino IDE, que oferece ampla compatibilidade com o microcontrolador ESP32 e um vasto repositório de bibliotecas. O código foi estruturado de forma modular para facilitar a legibilidade, a manutenção e a depuração. A seguir, são detalhadas as bibliotecas, as principais funções e a estrutura do código final.

# Bibliotecas Utilizadas

O código desenvolvido faz uso de cinco bibliotecas essenciais para o funcionamento dos diferentes módulos do sistema:

* Wire.h: Biblioteca padrão do Arduino para a comunicação via protocolo I2C, utilizada para a comunicação com o display LCD.
* LiquidCrystal\_I2C.h: Biblioteca de alto nível que simplifica o controle do display LCD 16x2 através da interface I2C.
* Keypad.h: Biblioteca padrão para a leitura de teclados matriciais, que abstrai a complexidade de varrer as linhas e colunas para detectar qual tecla foi pressionada.
* ESP32Servo.h: Biblioteca otimizada para o ESP32 que permite o controle preciso de servo motores, gerando os sinais PWM necessários para o posicionamento angular.
* Algorithm: Uma biblioteca padrão da linguagem C++ que foi incluída para utilizar a sua eficiente função std::sort, empregada no filtro de média aparada para ordenar as leituras do sensor.

# Funções Implementadas

Para garantir um código organizado e modular, a lógica foi dividida em 6 funções principais, cada uma com uma responsabilidade clara:

* setup(): Função padrão executada uma única vez na inicialização do ESP32. É responsável por configurar os pinos, inicializar os componentes (LCD, Servo, etc.) e definir o estado inicial do sistema.
* loop(): Função principal que é executada continuamente após o setup(). O seu papel é verificar a entrada do teclado e chamar a rotina de operação normal.
* getFilteredDistance(): Função responsável por realizar múltiplas leituras do sensor ultrassónico e aplicar um filtro de média aparada, retornando um valor de distância estável e fiável, minimizando os erros de leitura.
* processaTecla(): Função que gere toda a interação com o utilizador. Ela interpreta qual tecla foi pressionada (A, B, \*, # ou um número) e altera o estado do sistema (muda de modo, entra em configuração, etc.).
* operacaoNormal(): Contém a rotina principal de funcionamento. É aqui que o sistema lê o sensor (se estiver em Modo Real) e chama a função para atualizar os atuadores e o display.
* atualizarSistema(): Esta é a função central de controlo e feedback. Com base no valor atual de distanciaAtual, ela calcula o ângulo da comporta, decide se o alarme deve ser ativado e redesenha todo o ecrã do LCD com as informações de estado atualizadas.

# Pinos Utilizados

No código-fonte, são declarados explicitamente 13 pinos para o controle dos periféricos. No entanto, o hardware físico utiliza um total de 15 pinos de comunicação (excluindo GND e VCC), uma vez que a comunicação I2C com o display LCD utiliza as portas padrão 21 (SDA) e 22 (SCL) do ESP32, que são geridas internamente pela biblioteca Wire.h.

# Variáveis

O código-fonte desenvolvido para o projeto utiliza um conjunto de 8 variáveis globais para controlar o estado e os parâmetros do sistema. Esta estrutura foi projetada com o objetivo de garantir clareza e eficiência. A seguir, apresenta-se uma análise detalhada sobre cada tipo de variável utilizada e a sua finalidade dentro da lógica do programa:

Tipos Primitivos, Constantes e Strings:

const int (1 variável): ALTURA\_MAX\_CM - Uma constante que define a altura física da maquete (17cm), usada como limite para as leituras do sensor.

int (3 variáveis): distanciaAtual, nivelAlerta, nivelAtencao - Armazenam os valores numéricos cruciais para a lógica de controle: a distância medida pelo sensor e os limiares que definem os estados de "Atenção" e "Perigo".

String (1 variável): valorDigitado - Utilizada para construir dinamicamente o número que o utilizador digita no teclado antes de ser convertido para um valor inteiro.

bool (2 variáveis): modoDeConfiguracao, modoSimulacao - Variáveis booleanas (verdadeiro/falso) que funcionam como "chaves" para controlar o estado atual do sistema, determinando se o programa está no modo de configuração de alerta ou no modo de simulação.

# Código Final

O código-fonte completo do projeto possui aproximadamente 180 linhas e está extensivamente comentado. Por questões de legibilidade e para manter a organização deste documento, o código não será inserido diretamente aqui.

O código completo será apresentado no APÊNDICE A deste relatório.

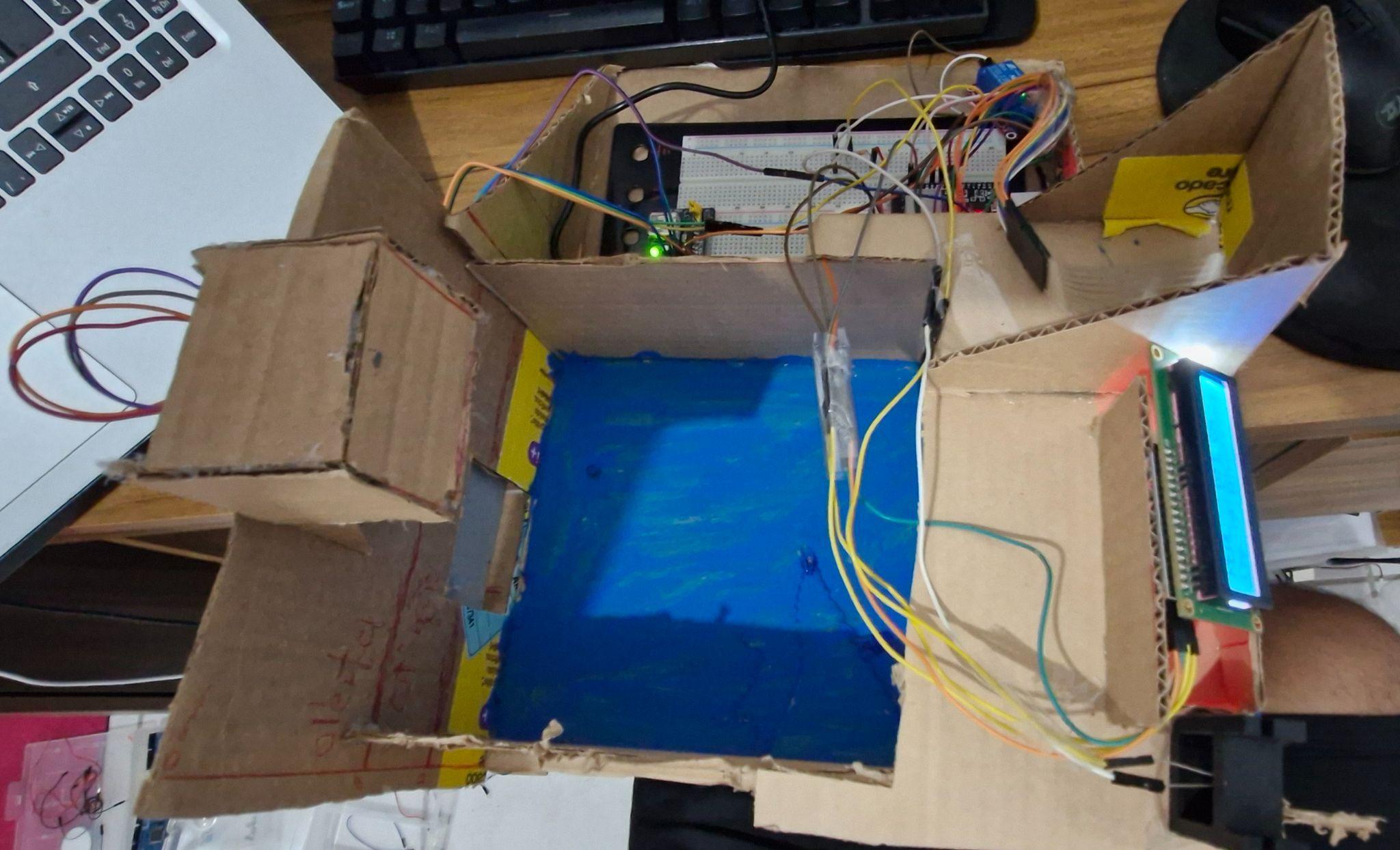
# Implementação & Testes

A fase final do projeto consistiu na implementação prática do protótipo e na realização de testes funcionais para validar a lógica e o hardware.

# Desafios Técnicos e Soluções

O principal desafio do projeto foi a instabilidade do hardware, um obstáculo comum em sistemas embarcados. A falha inicial em obter leituras consistentes do sensor ultrassónico e o não funcionamento dos atuadores (servo e relé) foram diagnosticados como um problema de alimentação elétrica. A corrente fornecida pela porta USB do computador revelou-se insuficiente para alimentar todo o conjunto de forma estável, especialmente durante os picos de consumo do motor.

A solução foi a implementação de uma fonte de alimentação externa dedicada para os componentes de maior consumo e, fundamentalmente, a conexão do pino GND da fonte externa com o pino GND do ESP32 para estabelecer uma "terra comum". Esta alteração, um conceito fundamental da eletrónica, resolveu todos os problemas de instabilidade e garantiu o funcionamento fiável de todo o sistema.



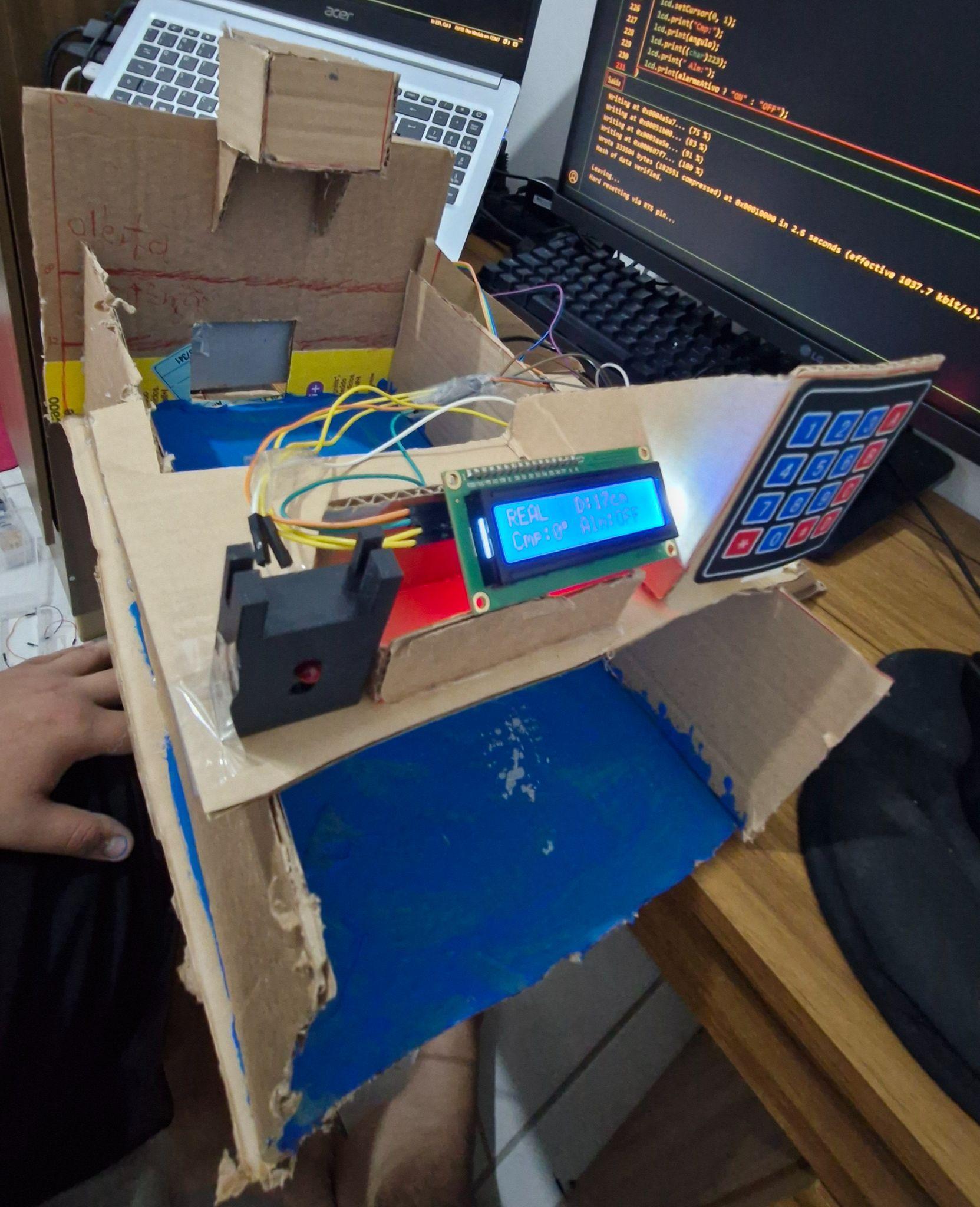
# Maquete

Após a validação completa da lógica de funcionamento na protoboard, foi realizada a montagem final do projeto em uma maquete simplificada, utilizando uma caixa de papelão como estrutura física para representar o cenário de aplicação. Nessa etapa, cada componente foi posicionado de forma estratégica para garantir uma demonstração clara, funcional e didática do sistema.

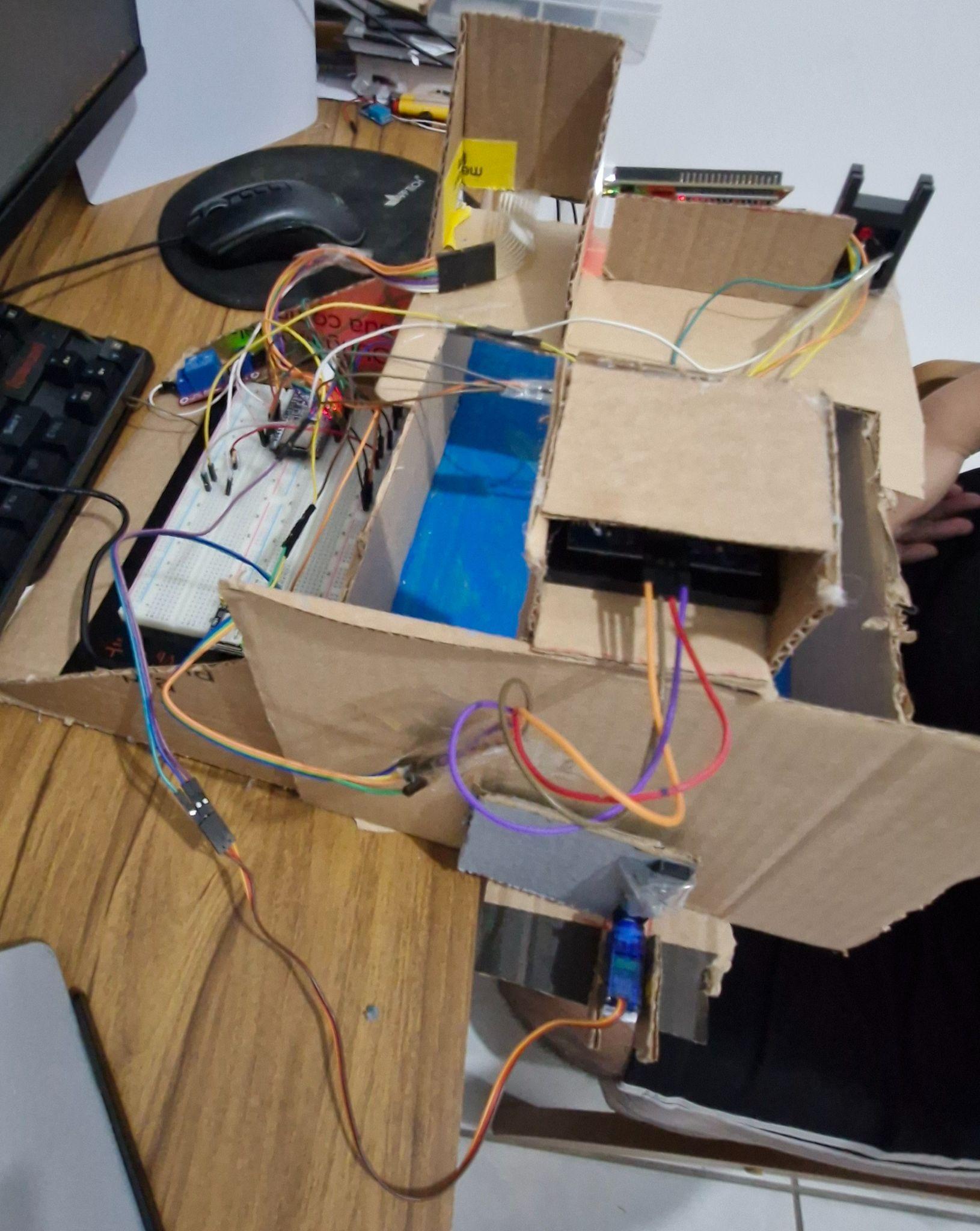
* O display LCD e o teclado matricial foram fixados de modo a ficarem visíveis e acessíveis externamente, compondo a interface homem-máquina (IHM) do sistema, permitindo a leitura das informações e a interação em tempo real.
* O sensor ultrassónico foi posicionado numa estrutura de "ponte" sobre o canal que simula o rio, garantindo uma medição vertical e sem obstruções da distância até o alvo que representa o nível da água.
* Uma estrutura móvel de papelão simulando a comporta do rio foi adicionada na extremidade do canal. Esta comporta é atuada mecanicamente por uma haste conectada ao servo motor, que, por sua vez, foi fixado na parte superior da estrutura da represa.
* O LED representando o alarme de perigo e o módulo de relé que o controla foram instalados de forma visível na parte externa da maquete, permitindo uma visualização clara e imediata do acionamento do estado de alerta.
* Para fins estéticos e de apresentação, a maquete foi finalizada para apresentar um aspecto limpo e organizado.

Esta maquete teve como objetivo principal representar visualmente o funcionamento do sistema de forma compacta e funcional, servindo como um suporte eficaz para a demonstração prática do projeto.

# Visão frontal:

****

# Visão Superior:

****

Esta maquete teve como objetivo principal representar visualmente o funcionamento do sistema de forma compacta e funcional, servindo como um suporte eficaz para a demonstração prática do projeto.

# Conclusão

O presente trabalho resultou na conceção, desenvolvimento e validação de um sistema embarcado funcional voltado para o controlo preventivo de enchentes, utilizando o microcontrolador ESP32 como núcleo da aplicação. A integração entre o sensor de nível, os atuadores (servo e relé) e a interface de utilizador demonstrou a viabilidade de criar soluções práticas e funcionais para problemas do mundo real, mesmo com recursos limitados.

Além da concretização dos objetivos técnicos, o projeto permitiu o confronto com diversos obstáculos reais comuns em sistemas embarcados, desde a instabilidade do sensor ultrassónico até às limitações de alimentação elétrica do protótipo. Estes desafios proporcionaram oportunidades valiosas de aprendizado, exigindo análise crítica, busca por soluções alternativas (como a implementação de filtros de software e a adoção de uma fonte de energia externa com terra comum) e adaptação contínua durante o desenvolvimento.

A experiência consolidou conhecimentos importantes em programação C++ na plataforma Arduino, integração de hardware e software, organização de tarefas em equipa e na elaboração de documentação técnica. O resultado é um sistema funcional, que serve como uma prova de conceito robusta e, ao mesmo tempo, uma base sólida para evoluções futuras. Entre as possibilidades de expansão estão a integração com serviços em nuvem (IoT) para monitorização remota, a criação de uma interface gráfica mais elaborada e a montagem do sistema numa placa de circuito impresso (PCB), aproximando ainda mais o protótipo de um produto final.

# Referências

ESPRESSIF. **ESP32-WROOM-32 Datasheet**. [s.l: s.n.]. Disponível em:<https://www.espressif.com/sites/default/files/documentation/esp32-wroom-32_datasheet_en.pdf>. Acesso em: 8 jul. 2025.

ELECFREAKS. **Ultrasonic Ranging Module HC-SR04**. [s.l: s.n.]. Disponível em:<https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf>. Acesso em: 8 jul. 2025.

**SERVO MOTOR SG90 DATA SHEET**. [s.l: s.n.]. Disponível em:<http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf>. Acesso em: 8 jul. 2025.

MARCO, Schwartz. **LiquidCrystal\_I2C Library**. Arduino Libraries. Disponível em:<https://github.com/marcoschwartz/LiquidCrystal_I2C>. Acesso em: 8 jul. 2025.

STANLEY, Mark; BREVIG, Alexander. **Keypad Library**. Arduino Libraries. Disponível em:<https://github.com/Chris--A/Keypad>. Acesso em: 8 jul. 2025.

# APÊNDICE A – CÓDIGO FONTE DO FIRMWARE

**/\*\***

**\* =================================================================================================**

**\***

**\* PROJETO FINAL: CONTROLE DE ENCHENTES**

**\* VERSÃO DE APRESENTAÇÃO - MODO DUPLO (REAL + SIMULAÇÃO)**

**\***

**\* =================================================================================================**

**\***

**\* Autor: Austin & amigos (AUTOMIND)**

**\* Curso: Engenharia de Controle e Automação - ECAE00**

**\* Data: 08/07/2025**

**\***

**\* -- DESCRIÇÃO --**

**\* Sistema de controle para o nível de um rio em uma maquete de 17cm.**

**\* Opera em dois modos, selecionáveis pelo teclado:**

**\* 1. MODO REAL: Utiliza um sensor ultrassónico com filtro de média aparada para leituras estáveis.**

**\* 2. MODO SIMULAÇÃO: Permite a inserção manual da distância pelo teclado para uma demonstração controlada.**

**\* O sistema controla uma comporta (Servo) e um alarme (Relé) com base na distância (real ou simulada).**

**\* O nível de alerta de perigo é configurável pelo utilizador.**

**\***

**\*/**

**// --- Bibliotecas ---**

**#include <Wire.h>**

**#include <LiquidCrystal\_I2C.h>**

**#include <Keypad.h>**

**#include <ESP32Servo.h>**

**#include <algorithm> // Necessária para a função de ordenação do filtro**

**// --- Definições de Pinos e Constantes ---**

**LiquidCrystal\_I2C lcd(0x27, 16, 2);**

**const int TRIGGER\_PIN = 5;**

**const int ECHO\_PIN = 18;**

**const byte ROWS = 4;**

**const byte COLS = 4;**

**char keys[ROWS][COLS] = {**

**{'1','2','3','A'},**

**{'4','5','6','B'},**

**{'7','8','9','C'},**

**{'\*','0','#','D'}**

**};**

**byte rowPins[ROWS] = {19, 23, 32, 33};**

**byte colPins[COLS] = {25, 26, 27, 14};**

**Keypad customKeypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);**

**Servo servoComporta;**

**const int SERVO\_PIN = 13;**

**const int RELE\_PIN = 12;**

**// --- Variáveis de Controle ---**

**const int ALTURA\_MAX\_CM = 17;**

**int distanciaAtual = ALTURA\_MAX\_CM;**

**int nivelAlerta = 8;**

**int nivelAtencao = 12;**

**String valorDigitado = "";**

**bool modoDeConfiguracao = false;**

**bool modoSimulacao = false;**

**/\*\***

**\* @brief Função de Setup: inicializa todos os componentes.**

**\*/**

**void setup() {**

**Serial.begin(115200);**

**pinMode(TRIGGER\_PIN, OUTPUT);**

**pinMode(ECHO\_PIN, INPUT);**

**servoComporta.attach(SERVO\_PIN);**

**pinMode(RELE\_PIN, OUTPUT);**

**digitalWrite(RELE\_PIN, LOW);**

**lcd.init();**

**lcd.backlight();**

**lcd.print("Iniciando...");**

**lcd.setCursor(0, 1);**

**lcd.print("Sistema vFinal");**

**delay(2000);**

**atualizarSistema();**

**}**

**/\*\***

**\* @brief Loop principal: verifica o teclado e executa a operação normal.**

**\*/**

**void loop() {**

**char key = customKeypad.getKey();**

**if (key) {**

**processaTecla(key);**

**}**

**if (!modoDeConfiguracao) {**

**operacaoNormal();**

**}**

**}**

**/\*\***

**\* @brief Filtro de Média Aparada: faz 7 leituras, descarta a maior e a menor, e calcula a média das 5 restantes.**

**\* @return A distância filtrada e estável em cm.**

**\*/**

**int getFilteredDistance() {**

**const int numReadings = 7;**

**int readings[numReadings];**

**int readingCount = 0;**

**while(readingCount < numReadings){**

**digitalWrite(TRIGGER\_PIN, LOW);**

**delayMicroseconds(2);**

**digitalWrite(TRIGGER\_PIN, HIGH);**

**delayMicroseconds(10);**

**digitalWrite(TRIGGER\_PIN, LOW);**

**long duration = pulseIn(ECHO\_PIN, HIGH, 25000);**

**if(duration > 0){**

**readings[readingCount] = duration \* 0.0343 / 2;**

**readingCount++;**

**}**

**}**

**std::sort(readings, readings + numReadings);**

**long total = 0;**

**for (int i = 1; i < numReadings - 1; i++) {**

**total += readings[i];**

**}**

**int distance = total / (numReadings - 2);**

**if (distance <= 0 || distance > ALTURA\_MAX\_CM) return ALTURA\_MAX\_CM;**

**return distance;**

**}**

**/\*\***

**\* @brief Processa as teclas para os diferentes modos de operação.**

**\*/**

**void processaTecla(char tecla) {**

**// Tecla 'A': Ativa o modo de simulação**

**if (tecla == 'A') {**

**modoSimulacao = true;**

**lcd.clear();**

**lcd.print("MODO SIMULACAO");**

**lcd.setCursor(0,1);**

**lcd.print("Digite dist+ '#'");**

**valorDigitado = "";**

**}**

**// Tecla 'B': Volta para o modo real**

**else if (tecla == 'B') {**

**modoSimulacao = false;**

**modoDeConfiguracao = false; // Garante que sai de qualquer outro modo**

**lcd.clear();**

**lcd.print("MODO REAL");**

**delay(1500);**

**atualizarSistema();**

**}**

**// Tecla '\*': Entra/sai do modo de configuração de alerta**

**else if (tecla == '\*') {**

**modoDeConfiguracao = !modoDeConfiguracao;**

**valorDigitado = "";**

**if (modoDeConfiguracao) {**

**lcd.clear();**

**lcd.print("Novo Alerta(cm):");**

**lcd.setCursor(0, 1);**

**lcd.print("Digite valor+ '#' ");**

**} else {**

**atualizarSistema();**

**}**

**}**

**// Tecla '#': Confirma a entrada de dados**

**else if (tecla == '#') {**

**if (valorDigitado.length() > 0) {**

**if (modoDeConfiguracao) {**

**nivelAlerta = valorDigitado.toInt();**

**nivelAtencao = (ALTURA\_MAX\_CM + nivelAlerta) / 2;**

**lcd.clear();**

**lcd.print("Alerta Salvo!");**

**} else if (modoSimulacao) {**

**distanciaAtual = valorDigitado.toInt();**

**lcd.clear();**

**lcd.print("Dist. Simulada!");**

**}**

**}**

**valorDigitado = "";**

**modoDeConfiguracao = false;**

**delay(1500);**

**atualizarSistema();**

**}**

**// Teclas numéricas**

**else if (isDigit(tecla)) {**

**if (valorDigitado.length() < 3) {**

**valorDigitado += tecla;**

**lcd.setCursor(0, 1);**

**lcd.print(valorDigitado);**

**}**

**}**

**}**

**/\*\***

**\* @brief Rotina de operação normal do sistema.**

**\*/**

**void operacaoNormal() {**

**// Se não estiver em modo de simulação, lê o sensor.**

**if (!modoSimulacao) {**

**distanciaAtual = getFilteredDistance();**

**}**

**atualizarSistema();**

**delay(300);**

**}**

**/\*\***

**\* @brief Atualiza todos os atuadores e o display com base nos dados atuais.**

**\*/**

**void atualizarSistema() {**

**int angulo = map(distanciaAtual, nivelAtencao, nivelAlerta, 0, 90);**

**angulo = constrain(angulo, 0, 90);**

**servoComporta.write(angulo);**

**bool alarmeAtivo = (distanciaAtual <= nivelAlerta);**

**digitalWrite(RELE\_PIN, alarmeAtivo ? HIGH : LOW);**

**String statusNivel;**

**if (distanciaAtual <= nivelAlerta) statusNivel = "PERIGO!";**

**else if (distanciaAtual <= nivelAtencao) statusNivel = "ATENCAO";**

**else statusNivel = "SEGURO";**

**lcd.clear();**

**lcd.setCursor(0, 0);**

**lcd.print(modoSimulacao ? "SIMUL" : "REAL");**

**lcd.setCursor(7, 0);**

**lcd.print("D:");**

**lcd.print(distanciaAtual);**

**lcd.print("cm");**

**lcd.setCursor(0, 1);**

**lcd.print("Cmp:");**

**lcd.print(angulo);**

**lcd.print((char)223);**

**lcd.print(" Alm:");**

**lcd.print(alarmeAtivo ? "ON" : "OFF");**

**}**