INSTITUTO FEDERAL DO ESPÍRITO SANTO
CAMPUS VILA VELHA
CURSO DE QUÍMICA INDUSTRIAL

NÍCOLAS DEZAN DOS SANTOS

AUTOMAÇÃO DE UM MALTEADOR LABORATORIAL: DESENVOLVIMENTO DE FIRMWARE E APLICATIVO PARA CONTROLE DE PROCESSOS

VILA VELHA-ES
2025

NÍCOLAS DEZAN DOS SANTOS

AUTOMAÇÃO DE UM MALTEADOR LABORATORIAL: DESENVOLVIMENTO DE FIRMWARE E APLICATIVO PARA CONTROLE DE PROCESSOS

Trabalho de Conclusão de Curso apresentado à Coordenadoria do Curso de Química Industrial do Instituto Federal de Educação, Ciência e Tecnologia do Espírito Santo, como requisito parcial para a obtenção do título de Químico Industrial.

Orientador: Prof. Dr. Ernesto Corrêa Ferreira

VILA VELHA-ES

2025

RESUMO

A viabilização de pesquisas em malteação demanda equipamentos capazes de controlar variáveis como temperatura, umidade e tempo. Contudo, soluções comerciais nem sempre são acessíveis para uso acadêmico, especialmente em laboratórios que operam com recursos limitados. Este trabalho apresenta o desenvolvimento de um sistema de automação baseado em um microcontrolador *ESP32-C3*, escolhido principalmente pela sua relação custo-benefício, com *firmware* programado em *MicroPython* e um aplicativo *Android* desenvolvido em *Kotlin*, a linguagem oficial da plataforma *Android*, capaz de monitorar e controlar o processo de malteação em laboratório. O sistema permite a configuração remota dos parâmetros de processo e o monitoramento em tempo real por meio da comunicação via *Bluetooth Low Energy* (BLE). A arquitetura do *firmware* foi montada com estrutura assíncrona, permitindo que múltiplos processos, como leitura de sensores e comunicação *Bluetooth*, ocorram simultaneamente sem bloqueios. O aplicativo *Android* adota uma arquitetura moderna em camadas, que separa as responsabilidades entre acesso a dados, lógica de negócio e interface do usuário com interface construída em *Jetpack Compose*, a tecnologia oficial do Android para desenvolvimento de interfaces. Os códigos desenvolvidos estão disponibilizados em repositórios *GitHub*, com o objetivo de promover reprodutibilidade e facilitar a adoção por outros pesquisadores. A solução proposta visa atender às demandas do LACEMP-IFES por uma plataforma de baixo custo e alta eficiência, contribuindo para a pesquisa aplicada em tecnologia de alimentos e automação de processos.

Palavras-chave: Automação. Malteação. Firmware. Android. Controle de Processos.

ABSTRACT

The development of research in malting requires equipment capable of controlling variables such as temperature, humidity, and time. However, commercial solutions are not always accessible for academic use, especially in laboratories operating with limited resources. This work presents the development of an automation system based on the ESP32-C3 microcontroller, with firmware programmed in MicroPython and an Android application developed in Kotlin — the official language of the Android platform — capable of monitoring and controlling the malting process in a laboratory environment. The system allows remote configuration of process parameters and real-time monitoring via Bluetooth Low Energy (BLE) communication. The firmware architecture was designed with an asynchronous structure, enabling multiple processes, such as sensor reading and Bluetooth communication, to run simultaneously without blocking. The Android application adopts a modern layered architecture that separates responsibilities between data access, business logic, and user interface, with the interface developed using Jetpack Compose — the official Android toolkit for building declarative and responsive interfaces. The developed code is available in public repositories on GitHub, aiming to promote reproducibility and facilitate adoption by other researchers. The proposed solution meets the demands of LACEMP-IFES for a low-cost and highly efficient platform, contributing to applied research in food technology and process automation.

Keywords: Automation. Malting. Firmware. Android. Process Control.

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 INTRODUÇÃO

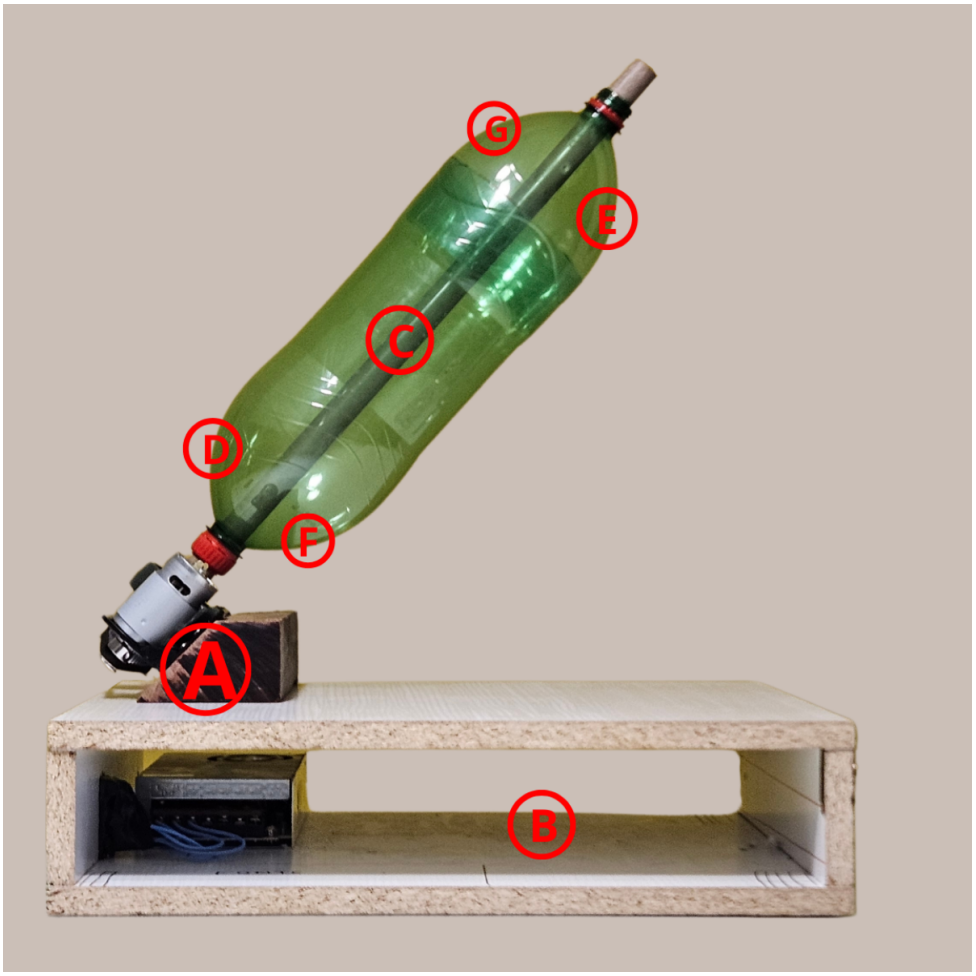
O malte é um produto de extrema importância para diversos setores industriais, com ele sendo o principal insumo na produção de cerveja, onde fornece açúcares fermentáveis, cor e aroma à bebida. No setor cervejeiro, o malte de cevada é o mais utilizado; no entanto, há um crescente interesse em pesquisas que buscam desenvolver malte a partir de outros grãos, como arroz, milho e trigo, visando atender demandas específicas, como a produção de cervejas livres de glúten (Ceccaroni *et al.*, 2019). Além disso, o malte tem ganhado destaque como ingrediente funcional na indústria alimentícia, sendo utilizado, por exemplo, na fabricação de pães, devido às suas propriedades nutricionais (Koistinen *et al.*, 2020).

A produção de malte, no entanto, enfrenta desafios significativos, especialmente em climas quentes, onde o controle preciso de temperatura, umidade e tempo, durante as etapas de maceração, germinação e secagem, é crucial para garantir a qualidade do produto (Kovalova *et al.*, 2024). O controle desses parâmetros torna a malteação um processo complexo, exigindo equipamentos especializados e sistemas automatizados para otimizar a produção. No contexto acadêmico e de pesquisa, a falta de equipamentos acessíveis e confiáveis para malteação em laboratório limita o desenvolvimento de estudos e inovações nesta área, evidenciando a importância de soluções de baixo custo.

Nesse contexto, um dos principais desafios enfrentados é a falta de equipamentos que permitam a malteação controlada em laboratório, comprometendo a precisão e a reprodutibilidade dos experimentos. Durante uma iniciação científica (IC), foi desenvolvido um protótipo inicial (Figura 1), não finalizado, restando completar a montagem física e desenvolver o sistema de controle e automação. O presente trabalho dá continuidade à IC ao propor uma solução de *software* responsável por garantir que o processo de malteação ocorra adequadamente.

A proposta visa o desenvolvimento de um sistema (Figura 2) baseado no microcontrolador ESP32-C3, da fabricante chinesa Espressif Systems, integrado com um aplicativo Android. Esse microcontrolador foi escolhido por sua relação custo-benefício, capacidade de processamento e suporte a tecnologias de comunicação como Bluetooth e

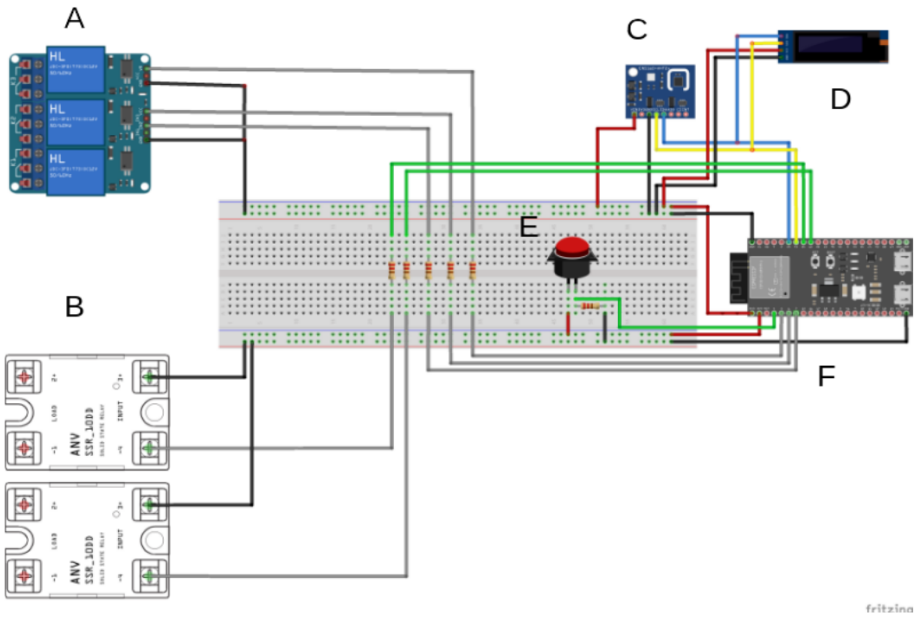
Figura 1 – Protótipo desenvolvido durante a IC. (A) Interface, placa de desenvolvimento e relés das válvulas; (B) Relé de estado sólido; (C) Parafuso de revolvimento; (D) Entrada de ar; (E) Entrada de água; (F) Saída de água; (G) Sensor de dióxido de carbono, umidade e temperatura.



Fonte: Autoria própria.

Wi-Fi (Rodrigues *et al.*, 2021; Santos; Junior, 2019). Para seu firmware, optou-se pela linguagem MicroPython, que oferece uma curva de aprendizado suave e é bastante utilizada em projetos de prototipagem rápida e Internet das Coisas (IoT) (Tollervey, 2017; Brito, 2020). Já o aplicativo Android foi desenvolvido em Kotlin, linguagem oficial para o desenvolvimento Android (Santana, 2020). A comunicação entre o firmware e o aplicativo é realizada via Bluetooth Low Energy (BLE), uma tecnologia de baixo consumo energético e amplamente disponível em dispositivos móveis (Heydon; Hunn, 2012), permitindo a configuração remota e o monitoramento em tempo real dos parâmetros operacionais. Além disso, a documentação dos softwares desenvolvidos pode servir como referência para outros estudos envolvendo IoT dentro do IFES, especialmente no que se refere à integração entre ESP32 e Android.

Figura 2 – Esquema eletrônico do malteador desenvolvido durante a IC: (A) relés para atuação das válvulas; (B) relés para atuação do motor e da resistência elétrica; (C) sensor de umidade, temperatura e dióxido de carbono; (D) painel LCD; (E) botão de emergência; (F) microcontrolador ESP32



Fonte: Autoria própria.

Este trabalho surge a partir da demanda do Laboratório de Análises de Cerveja e Matérias-Primas (LACEMP), que busca iniciar estudos experimentais sobre o processo de malteação. A ausência de um equipamento adequado e acessível para pesquisas acadêmicas motivou o desenvolvimento de um sistema que permita monitorar e controlar as variáveis do processo de forma precisa e reprodutível.

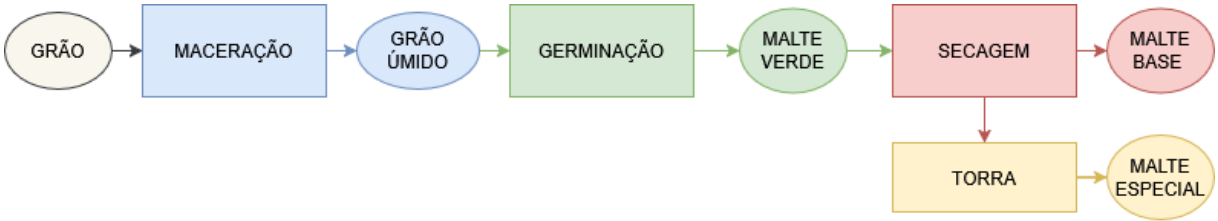
Por fim, a automação proposta busca simplificar a operação do equipamento e fornecer dados estruturados sobre o processo, facilitando futuras análises e ajustes nos experimentos conduzidos no LACEMP. Com isso, espera-se que esta solução de baixo custo promova avanços tanto na pesquisa acadêmica quanto no desenvolvimento de tecnologias acessíveis para a indústria cervejeira e áreas afins.

2 REFERENCIAL TEÓRICO

2.1 O PROCESSO DE MALTEAÇÃO

A malteação é um dos processos fundamentais na indústria cervejeira, responsável pela conversão do grão cru em um ingrediente essencial para a produção de cerveja. Esse processo ocorre em três etapas principais: maceração, germinação e secagem. Essas etapas são fundamentais para a produção de malte base: um malte que possui uma boa quantidade de enzimas e estoques de amido (Briggs *et al.*, 2004; Cenci *et al.*, 2021). Além disso, pode-se considerar uma quarta etapa no processo: a torrefação, que ocorre após a secagem e produz um malte especial no lugar de um malte base. Maltes especiais são conhecidos por serem formados com bastante influência das reações de Maillard (Coghe *et al.*, 2004), conjunto complexo de reações não enzimáticas que envolvem a condensação de açúcares redutores com aminoácidos, gerando compostos intermediários (como os produtos de Amadori) e, enfim, melanoidinas que são responsáveis pela cor e aroma característicos (Nursten, 2005). A complexidade desse processo se dá no fato da manipulação de um organismo vivo, o grão, que exige controle rigoroso de suas condições de crescimento para se tornar um produto viável para comercialização (Mallett, 2022). Além disso, em condições inadequadas de malteação, podem ocorrer ataques microbiológicos de diversas espécies, incluindo *Fusarium sp.*, *Penicillium* e *Aspergillus genera* (Luarasi; Troja; Pinguli, 2016).

Figura 3 – Fluxograma incluindo as etapas de malteação.



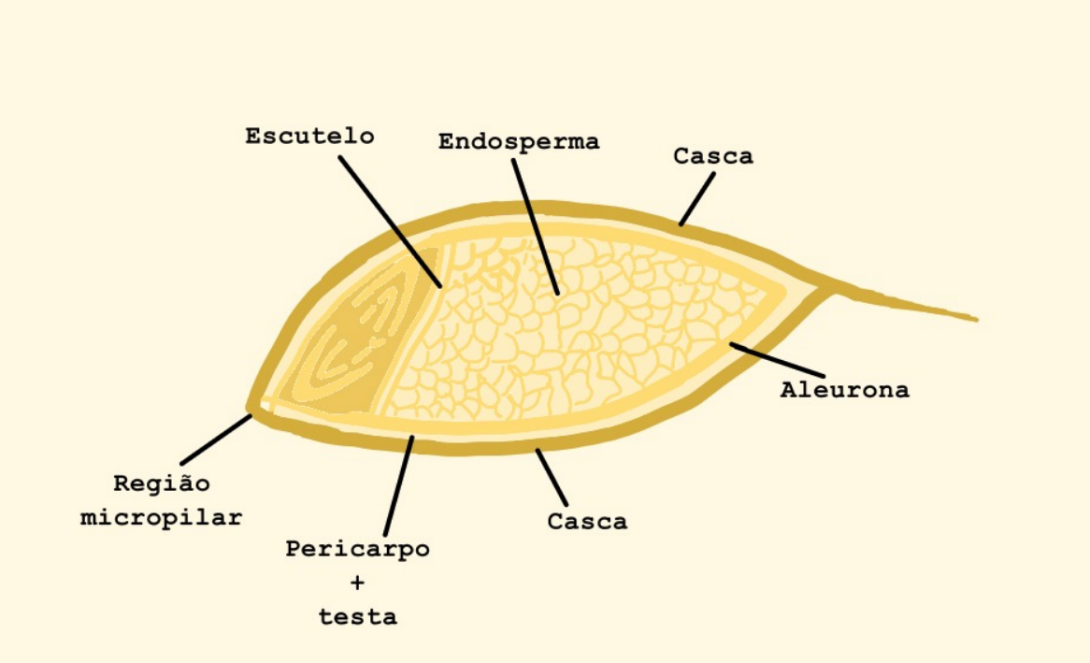
Fonte: Autoria própria.

2.1.1 Maceração

Durante a maceração, com a submersão do grão em água, ocorre a liberação de hormônios e enzimas que darão início ao crescimento e desenvolvimento do grão (Lewis; Young, 2012). O teor de água no grão cresce rapidamente após o início dessa primeira etapa, que ocorre em duas fases distintas: uma inicial, em que o embrião

e o escutelo absorvem água rapidamente, e uma segunda fase, mais lenta, em que o endosperma é gradualmente hidratado. Esse processo é crucial para a ativação de enzimas como a amilase, a ribonuclease e a fosfatase, que desempenham papeis essenciais na modificação do grão (Reynolds; MacWilliam, 1966). Também é crucial que o oxigênio seja fornecido ao mesmo tempo em que o dióxido de carbono é eliminado dos grãos. O aumento da umidade intensifica o metabolismo do grão, elevando sua taxa respiratória, ou seja, mais oxigênio é requerido (Kunze, 1996). Por isso, a presença de tanques de maceração que bombeiam ar através dos grãos submersos é frequente nas produções de larga escala (Cenci *et al.*, 2021). Mas, muitos estudos ainda investigam os impactos de se trabalhar com ciclos de períodos secos e submersos, especialmente, quando se busca viabilizar novas variedades de grãos para a malteação (Mayer *et al.*, 2014; Turner *et al.*, 2019).

Figura 4 – Estrutura do grão de cevada.



Fonte: Autoria própria. Adaptado de Lewis & Young (2012).

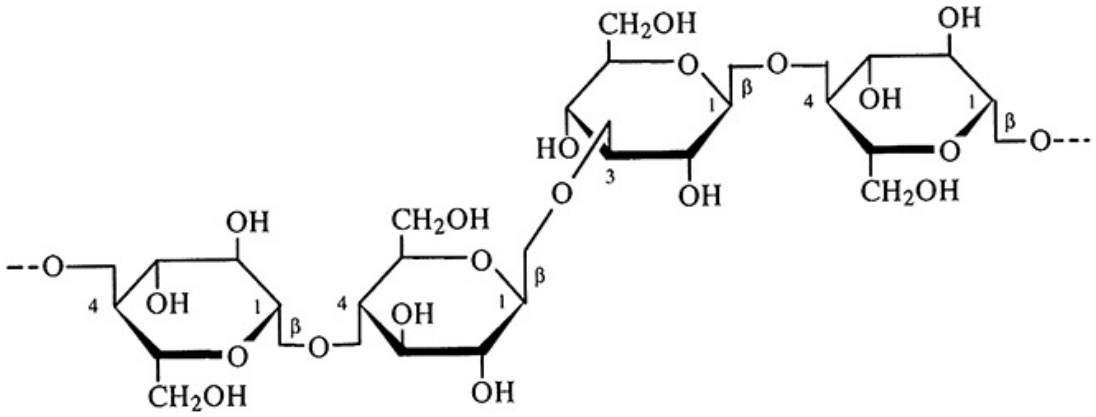
2.1.2 Germinação

Ao fim da maceração, que pode durar até 72 horas, o teor de umidade dos grãos chega a aproximadamente 45%, e as radículas tornam-se visíveis, indicando o momento adequado para o início da germinação. Essa segunda etapa é caracterizada por um elevado nível metabólico nos grãos, com a ocorrência de transformações bioquímicas

essenciais para o processo de malteação (Mallett, 2022). Do ponto de vista do malteador, a duração da germinação deve ser cuidadosamente controlada, pois um período insuficiente ou excessivo pode comprometer a qualidade do malte. Se a germinação for muito curta, as transformações necessárias nos grãos não ocorrerão de forma adequada. Por exemplo, as enzimas não conseguirão degradar completamente as paredes celulares proteicas que envolvem o amido, tornando-o menos disponível para as etapas subsequentes do processo cervejeiro (Fox, 2009). Por outro lado, uma germinação prolongada pode levar ao esgotamento excessivo dos nutrientes do grão visando o crescimento da planta, afetando negativamente o produto final (Lewis; Young, 2012).

Durante a germinação, a ativação de enzimas hidrolíticas desempenha um papel essencial na modificação do grão. As β-glucanases promovem a degradação dos β-glucanos (Figura 5) presentes na parede celular do endosperma, substâncias estas que são indesejadas no processo cervejeiro em altas concentrações (Lewis; Young, 2012). Essas enzimas hidrolisam as ligações β-1,3 e β-1,4 dos glucanos de cadeia mista, comprometendo a integridade da parede celular e facilitando a liberação de proteínas presentes nas células da camada de aleurona (Figura 4) (Bobade; Gupta; Sharma, 2022). Paralelamente, proteases hidrolisam a matriz proteica que envolve os grânulos de amido, liberando pequenos peptídeos e aminoácidos (Fox, 2009; Gupta; Abu-Ghannam; Gallaghar, 2010). Já a conversão do amido (Figura 6) é mediada por enzimas amilolíticas, sendo a α-amilase responsável pela quebra aleatória das ligações α-1,4-glicosídicas e a β-amilase pela liberação de maltose, um dos principais açúcares fermentáveis do processo cervejeiro (Gupta; Abu-Ghannam; Gallaghar, 2010; Mallett, 2022). Em essência, o grão cru entra no processo com compostos de alto peso molecular e, ao fim da malteação, gera como produto um malte com compostos de baixo peso molecular e boa concentração de enzimas (Kunze, 1996). Essa transformação, que ocorre principalmente na germinação, é o que torna o malte indispensável para o processo cervejeiro (Cenci *et al.*, 2021).

Figura 5 – Esquema de uma molécula de β-glucano.



Fonte: Rop, Mlcek & Jurikova (2009).

Tabela 1 – Comparativo entre as enzimas α-amilase e β-amilase.

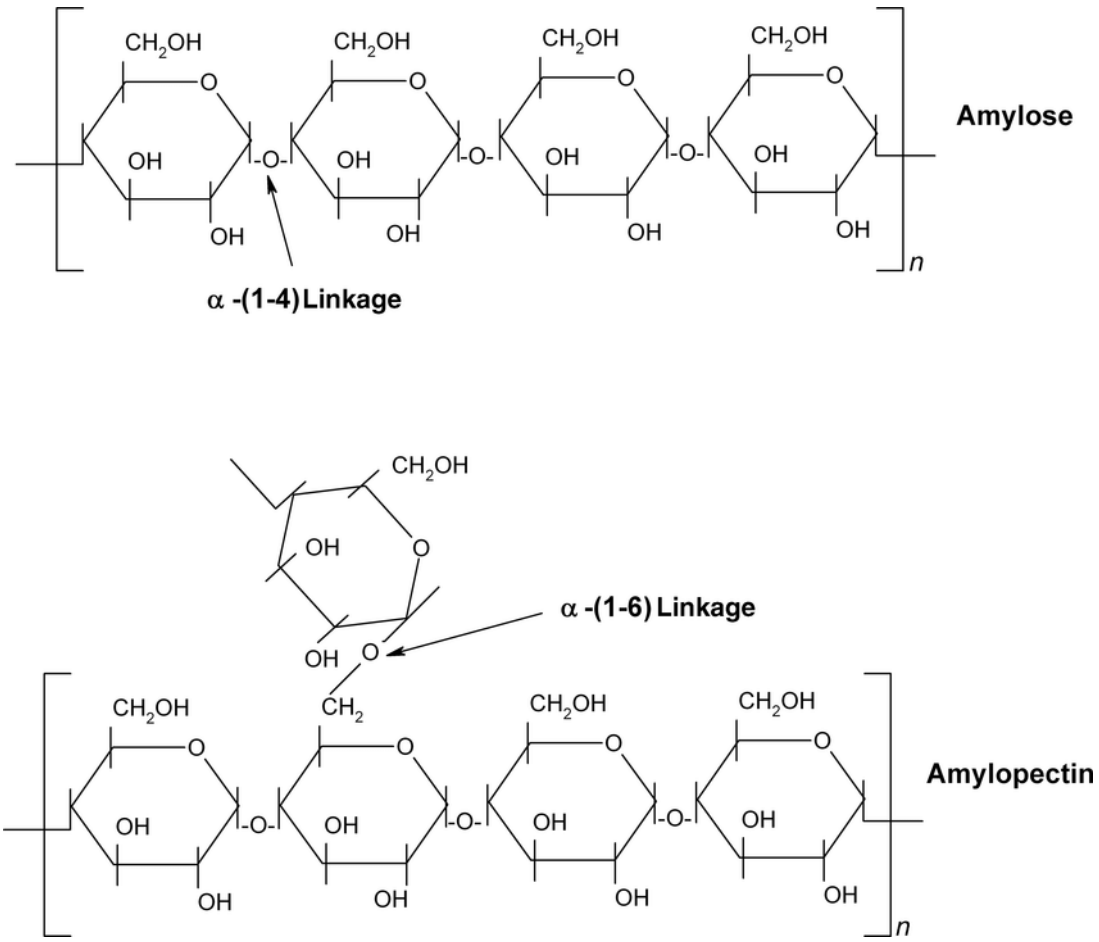
Característica	**α-amilase**	**β-amilase**
Posição de ataque	Interna na cadeia	Extremos não redutores
Produto principal	Maltose, glicose e dextrinas	Maltose
Ligação hidrolisada	α-1,4	α-1,4
Temperatura ideal	70 – 75 °C	60 – 65 °C
Termoestabilidade	Alta	Moderada
Papel no processo	Liquefação do amido	Produção de açúcares fermentáveis

Fonte: Adaptado de Gupta, Abu-Ghannam & Gallaghar (2010), Lewis & Young (2012), Mallett (2022).

2.1.3 Secagem

Quando a germinação atinge o tempo otimizado para promover as melhores transformações, inicia-se a etapa de secagem. Nessa fase, de acordo com Griffiths (1992 apud Woffenden *et al.*, 2002), os grãos são desidratados por até 30 horas, o que resulta em um malte base de fácil manuseio e adequado para armazenamento. Além da redução da umidade para aumento da estabilidade do produto final, a secagem promove o desenvolvimento de aromas desejados e de coloração no malte (Bamforth, 2003). Apesar disso, se a intenção é produzir um malte base, a temperatura não pode ser excessiva, com o fim de preservar as enzimas, como as amilases, no produto final (Lewis; Young, 2012).Outra questão benéfica é que a secagem elimina boa parte dos microorganismos que cresceram de forma indesejada durante os processos anteriores (Douglas; Flannigan, 1988; Petters; Flannigan; Austin, 1988).

Figura 6 – Estrutura química do amido representando as unidades de amilose (*amylose*) e amilopectina (*amylopectin*).



Fonte: Visakh *et al.* (2012).

2.2 VARIÁVEIS NA MALTEAÇÃO

2.2.1 Temperatura

O controle da temperatura durante a germinação é essencial para minimizar perdas causadas pelo crescimento excessivo das radículas e do embrião da planta, evitando o consumo desnecessário dos estoques de amido do grão (Pitz, 1990; Mallett, 2022). Além disso, a temperatura influencia diretamente a atividade enzimática e a degradação de componentes estruturais do grão. Um estudo conduzido por Baxter, Reeves & Bamforth (1980) avaliou os efeitos da maceração em uma temperatura superior à faixa usual de 12-16 °C, chegando a 30 °C. Os resultados indicaram que temperaturas elevadas comprometem a atividade enzimática e reduzem a eficiência da degradação de β-glucanos e proteínas, afetando negativamente a qualidade do malte.

Outro fator crítico relacionado à temperatura é o crescimento microbiológico. Segundo Tangni & Larondelle (2002), temperaturas mais altas na malteação favorecem a proliferação de *Aspergillus clavatus*, um fungo produtor de micotoxinas. A contaminação microbiológica ocorre predominantemente durante a germinação, mas também pode ser observada ao final da maceração (Petters; Flannigan; Austin, 1988). Dessa forma, a manutenção de temperaturas controladas entre 12 e 22 °C durante as etapas de maceração e germinação é fundamental não apenas para garantir a qualidade do malte, mas também para assegurar a segurança sanitária do processo (Tangni; Larondelle, 2002).

2.2.1.1 Temperatura na secagem

Na etapa de secagem, o controle da temperatura desempenha um papel crucial em dois aspectos principais: garantir que o malte atinja a umidade adequada para armazenamento e determinar o tipo de malte produzido (Kunze, 1996). A secagem ocorre, geralmente, em múltiplas fases, seguindo uma rampa de temperatura. Os valores típicos variam na faixa de 50 a 110 °C, dependendo do perfil desejado para o malte final (Lewis; Young, 2012).

De acordo com Skendi & Papageorgiou (2018), a temperatura de secagem influencia diretamente a composição de açúcares fermentáveis no malte e a coloração do mosto produzido. No estudo, a secagem a 80 °C resultou em um mosto com maior teor de açúcares fermentáveis do que a secagem a temperaturas superiores, como 90 °C. Além disso, foi observado um escurecimento do mosto com o aumento da temperatura de secagem, um fator determinante na definição das características finais do malte. Essa relação entre temperatura e cor ocorre devido à intensificação das reações de Maillard e à degradação térmica de compostos presentes no malte (Kunze, 1996).

2.2.2 Aeração

A aeração dos grãos é um aspecto essencial da malteação, uma vez que o processo envolve um organismo vivo que depende da respiração para seu desenvolvimento (Mallett, 2022). De acordo com Wilhelmson *et al.* (2006), no início da maceração ocorre uma deficiência de oxigênio devido à submersão dos grãos, mas esse fator

não compromete a qualidade do malte. No entanto, à medida que a germinação avança, a disponibilidade de O_2 torna-se mais crítica, pois a respiração dos grãos gera acúmulo de dióxido de carbono, o que pode inibir a germinação. Esse efeito pode ser intensificado pelo emaranhamento das radículas, que dificulta a circulação de ar e reduz ainda mais a disponibilidade de oxigênio. Para mitigar esse problema, sistemas revolvedores são frequentemente empregados para movimentar os grãos e garantir uma aeração adequada ao longo do processo (Cenci *et al.*, 2021).

2.2.3 Tempo

A duração de cada etapa da malteação é um fator crítico para a qualidade do malte, influenciada pela variedade do grão e pelas condições do processo (umidade, temperatura e aeração). A otimização desse parâmetro é essencial para adaptar novas variedades às demandas industriais. Como demonstrado por Farzaneh *et al.* (2017), o tempo de germinação (3 a 7 dias) afeta diretamente as propriedades do malte: períodos mais longos (7 dias) elevam a atividade de β-glucanase e α-amilase, reduzindo o teor de amido e β-glucano devido ao consumo enzimático, além de aumentar as perdas por crescimento. Assim, a definição do tempo ideal deve equilibrar modificação enzimática e eficiência do processo, considerando o perfil desejado no malte. Por exemplo, germinação prolongada é indicada para cervejas de alta fermentabilidade, enquanto períodos mais curtos (3-5 dias) preservam polissacarídeos, sendo ideais para estilos encorpados.

Na etapa de maceração, o tempo necessário para a hidratação dos grãos é um fator determinante para a ativação enzimática e o desenvolvimento adequado do malte. Montanuci *et al.* (2017) demonstraram que períodos mais longos de maceração favorecem a absorção de água, impactando diretamente a degradação de β-glucanos e o desenvolvimento das enzimas α- e β-amilase. No entanto, o tempo ideal depende da temperatura empregada: enquanto macerações a 10 °C por 24 horas resultam em maior teor de açúcares no malte, temperaturas mais elevadas (20 °C por 12 horas) aceleram a hidratação, porém podem comprometer a viabilidade dos grãos e aumentar a degradação de componentes essenciais. Além disso, tempos excessivos de maceração podem favorecer o crescimento microbiológico indesejado, exigindo um

controle rigoroso para evitar contaminações e perdas na qualidade do malte (Luarasi; Troja; Pinguli, 2016). Dessa forma, a definição do tempo de maceração deve equilibrar a eficiência da hidratação com a preservação da integridade dos grãos, garantindo um substrato adequado para as etapas subsequentes da malteação.

O tempo de secagem deve ser ajustado em conjunto com a temperatura para garantir uma remoção eficiente da umidade, reduzindo-a para aproximadamente 4%, sem comprometer a qualidade enzimática e sensorial do malte (Lewis; Young, 2012).

2.3 AUTOMAÇÃO E CONTROLE DE PROCESSOS

A automação desempenha um papel essencial na indústria no geral, proporcionando maior qualidade no produto final, otimização da produção e aumento da segurança operacional (Seborg *et al.*, 2016). Para garantir um controle eficaz dos processos industriais, são utilizados sistemas instrumentados, compostos por três elementos fundamentais: sensores, processadores de sinais e interfaces de visualização (Bolton, 2021). Os sensores são responsáveis pela coleta de informações sobre variáveis do processo, como temperatura, pressão e vazão, permitindo o monitoramento contínuo das condições operacionais. Os processadores de sinais, por sua vez, realizam o tratamento e a interpretação dos dados captados, aplicando algoritmos de controle para ajustar automaticamente os parâmetros do sistema conforme necessário. Já as interfaces de visualização possibilitam que operadores e engenheiros acompanhem o comportamento do processo em tempo real, viabilizando a tomada de decisões informadas e a rápida identificação de falhas (Bolton, 2021). Além disso, sistemas automatizados contribuem significativamente para a reprodutibilidade dos processos industriais, reduzindo variações indesejadas e garantindo conformidade com padrões regulatórios (Seborg *et al.*, 2016).

2.3.1 Controlador ON-OFF

O controlador ON-OFF é um dos métodos mais simples de controle de processos, operando com apenas dois estados: ligado (ON) e desligado (OFF). Esse tipo de controle é amplamente utilizado quando precisão extrema não é necessária e o sistema pode tolerar pequenas oscilações na variável controlada (Bolton, 2021). Seu funcionamento

é baseado em um ponto de ajuste (*setpoint*): quando a variável de processo ultrapassa esse valor, o atuador é ativado ou desativado, sem intermediários. Embora seja uma solução de fácil implementação e baixo custo, pode gerar oscilações constantes em torno do *setpoint*, tornando-se inadequado para processos que exigem estabilidade mais refinada (Seborg *et al.*, 2016).

2.3.2 Interfaces Gráficas

A Interface Gráfica do Usuário, ou HMI (*Human-Machine Interface* - Interface Homem Máquina), é um elemento essencial na automação, pois permite a interação intuitiva entre operadores e sistemas de controle. Segundo Bolton (2021), esse tipo de interface apresenta informações de processo de forma visual e interativa, utilizando janelas, ícones, menus e dispositivos apontadores. Em ambientes industriais, as telas de supervisão geralmente empregam displays miméticos, que representam esquematicamente as principais partes de uma planta, exibindo valores atualizados de variáveis controladas, gráficos de tendências e alarmes em tempo real. Além disso, as interfaces gráficas possibilitam que os operadores definam valores de *setpoint* e ajustem parâmetros do processo diretamente pela tela, eliminando a necessidade de comandos textuais complexos.

2.4 MICROCONTROLADORES DA ESPRESSIF (CHINA)

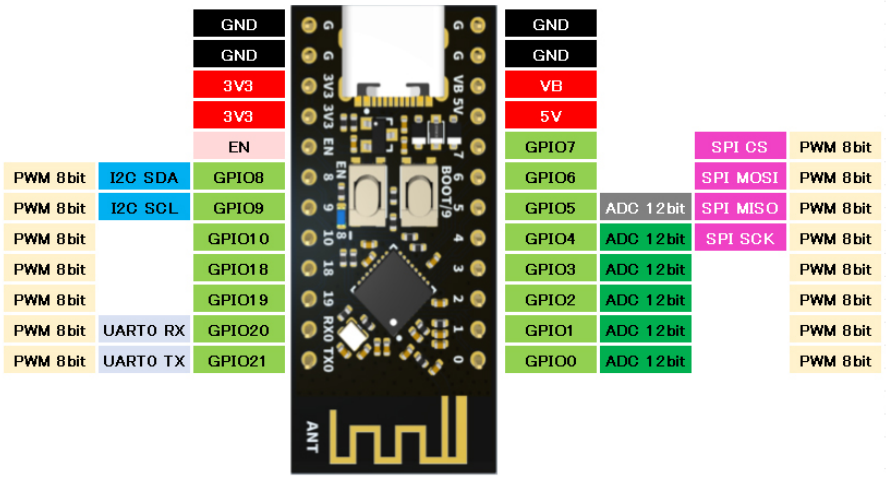
De acordo com Kolban (2017), os circuitos integrados baseados no microcontrolador ESP32, da Espressif (China), são amplamente utilizados devido ao seu baixo custo e à capacidade de executar aplicações autônomas, tornando-se uma escolha popular para projetos de automação e controle de processos. Além disso, sua versatilidade permite a integração com diversos sensores e dispositivos periféricos, tornando-o adequado para uma ampla gama de aplicações em automação e sistemas embarcados.

Graças à sua conectividade com a internet, as placas ESP32 estão fortemente associadas ao conceito de IoT. O IoT tem papel fundamental na otimização de processos industriais, permitindo a coleta de dados em larga escala (*big data*) e a detecção de falhas, o que contribui para a redução de custos operacionais (Ferencz; Domokos, 2020). Essa tecnologia é composta por uma variedade de dispositivos, como sensores,

atuadores e controladores, que podem se comunicar por meio de diferentes protocolos, incluindo Bluetooth, Ethernet e Wi-Fi (Shinde; Bhagat, 2017).

Um exemplo representativo dessa linha é o **ESP32-C3**, também desenvolvido pela Espressif Systems, que combina conectividade sem fio com processamento eficiente em um encapsulamento compacto. Esse modelo integra um núcleo RISC-V de 32 bits operando a até 160 MHz, além de periféricos como UART, SPI, I²C, temporizadores, controladores PWM e ADCs de 12 bits com até seis canais. A memória interna é composta por 400 kB de SRAM, 384 kB de ROM e 8 kB adicionais dedicados à operação em modo de baixo consumo. A operação típica do chip ocorre com alimentação de 3.3 V, com suporte a múltiplos modos de economia de energia, incluindo o *deep-sleep*, no qual o consumo é reduzido à ordem de microamperes (Espressif Systems, 2025).

Figura 7 – Esquema de uma placa com chip ESP32-C3FH4 (WeActStudio, China)



Fonte: Devicemart.

A flexibilidade dos pinos de entrada e saída (Figura 7) programáveis, somada à capacidade de comunicação via Wi-Fi e Bluetooth Low Energy, tornam esses microcontroladores uma alternativa robusta e econômica para aplicações embarcadas em automação, monitoramento remoto e sistemas de controle (Espressif Systems, 2025).

2.4.1 MicroPython

O MicroPython é um interpretador da linguagem Python 3 desenvolvido especificamente para microcontroladores (Plauska; Liutkevičius; Janavičiūtė, 2022). A linguagem Python

destaca-se por sua sintaxe simplificada e ampla adoção na comunidade tecnológica, sendo especialmente útil para programadores iniciantes (Tollervey, 2017).

Estudos conduzidos por Plauska, Liutkevičius & Janavičiūtė (2022) avaliaram o desempenho das principais linguagens utilizadas na programação do ESP32, incluindo o MicroPython. Os resultados indicaram que, embora sua eficiência em termos de desempenho seja inferior à de linguagens compiladas, como C e C++, o MicroPython continua sendo uma alternativa viável para aplicações onde o controle direto e preciso do hardware não é essencial. Sua principal vantagem reside na facilidade de prototipação e desenvolvimento rápido, tornando-o ideal para projetos com foco em lógica de alto nível ou ensino (Tanganelli; Vallati; Mingozzi, 2019).

2.5 ANDROID

O Android é um dos principais sistemas operacionais para dispositivos móveis, sendo um projeto de código aberto baseado no kernel Linux e liderado pelo Google (Ableson; King; Ortiz, 2011). Inicialmente, os aplicativos desenvolvidos para essa plataforma eram escritos predominantemente em Java (Ableson; King; Ortiz, 2011). No entanto, em 2017, o Google anunciou o suporte oficial ao Kotlin como linguagem de programação integrada ao *Android SDK*, permitindo que os desenvolvedores utilizem a linguagem nativamente no ambiente de desenvolvimento padrão da plataforma (Sills *et al.*, 2023).

A integração entre sistemas IoT e dispositivos móveis baseados em Android representa um avanço significativo para aplicações de monitoramento remoto. Diferentes estudos demonstram a viabilidade do uso de smartphones como interface de visualização e controle em arquiteturas distribuídas. Mohanasundaram *et al.* (2024) apresentaram um sistema para monitoramento da qualidade da água em tempo real, com sensores de pH, turbidez, temperatura e nível, cujos dados eram transmitidos para uma aplicação Android, permitindo o acompanhamento contínuo e remoto dos parâmetros analisados. De forma semelhante, Nascimento & Cichaczewski (2021) desenvolveram um sistema para monitoramento do nível de água em reservatórios, empregando sensores ultrassônicos e um microcontrolador ESP8266, com os dados sendo exibidos em gráficos em um aplicativo Android. O uso do dispositivo móvel como interface contribuiu para facilitar

a visualização remota e a interação com o sistema, demonstrando sua aplicabilidade em soluções IoT voltadas ao monitoramento contínuo.

Outro caso representativo foi descrito por Dhingra *et al.* (2019), que implementaram um sistema de monitoramento da qualidade do ar com sensores integrados a uma plataforma em nuvem, permitindo a consulta do índice de qualidade do ar em tempo real por meio de um aplicativo Android. O sistema utiliza geolocalização para gerar rotas personalizadas com base na poluição atmosférica, incluindo mapas interativos com alertas visuais. Tais abordagens reforçam o papel do Android como plataforma acessível e eficiente para aplicações embarcadas conectadas ao IoT.

2.5.1 Kotlin

O Kotlin se destaca como uma linguagem moderna, concisa, segura e pragmática, oferecendo total interoperabilidade com código Java, o que facilita a migração e a integração de projetos legados (Jemerov; Isakova, 2017). Atualmente, é amplamente adotado como a principal linguagem para o desenvolvimento de aplicativos móveis na plataforma Android, proporcionando maior produtividade e reduzindo a incidência de erros comuns durante a programação. Essa adoção é fortalecida pela integração nativa do Kotlin no *Android Studio*, ambiente de desenvolvimento oficial mantido pelo Google, que fornece ferramentas específicas para facilitar a escrita, o teste e a depuração de aplicativos Android (Android Developers, 2023).

2.5.2 Bluetooth Low Energy

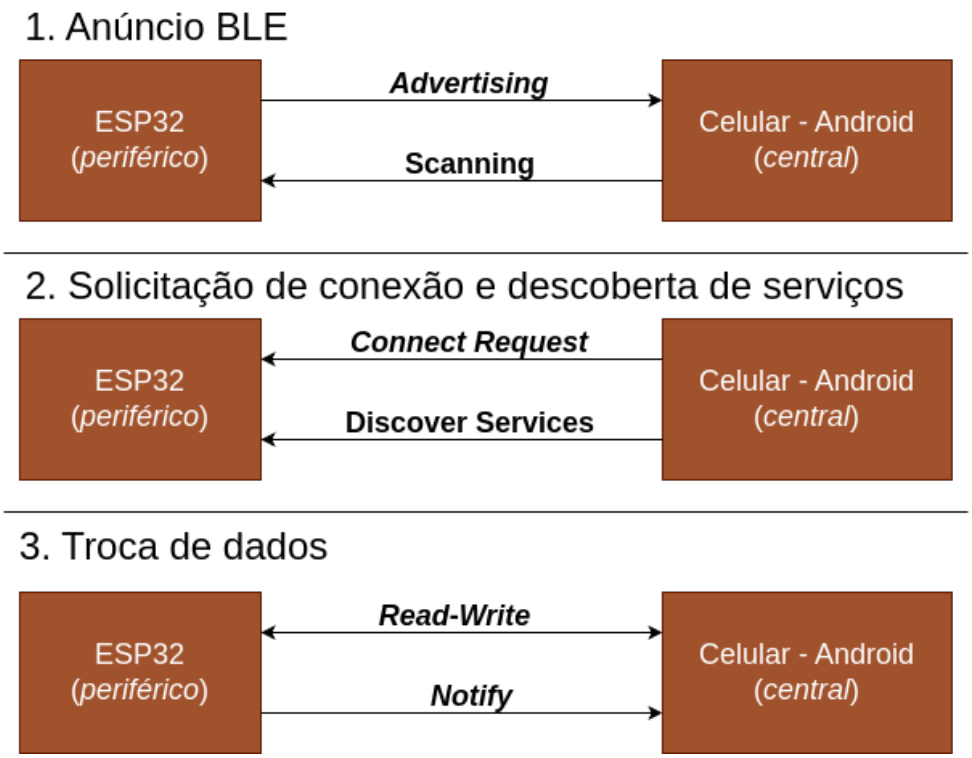
A plataforma Android oferece suporte a diferentes protocolos de comunicação via Bluetooth, incluindo o Bluetooth Low Energy (BLE), uma tecnologia especialmente adequada para conexões eficientes com dispositivos de baixo consumo energético, como o ESP32 (Android Developers, 2024).

Desenvolvido como uma evolução do Bluetooth clássico, o BLE foi projetado para reduzir significativamente o consumo de energia sem comprometer a conectividade (Heydon; Hunn, 2012). Essa característica o torna ideal para aplicações que demandam comunicação contínua com eficiência energética, como sistemas de automação e

monitoramento remoto. Nesse contexto, a integração entre dispositivos Android (como smartphones) e microcontroladores (como o ESP32) via BLE viabiliza a troca de dados em tempo real, dispensando a necessidade de conexões Wi-Fi ou cabos.

A comunicação BLE baseia-se em uma arquitetura assimétrica, envolvendo dois tipos de dispositivos: o *central* (por exemplo, um smartphone Android) e o *periférico* (como o ESP32). Como ilustrado na Figura 8, o processo inicia-se com o envio de pacotes de anúncio (*advertising packets*) pelo periférico, que sinaliza sua disponibilidade. O dispositivo central, ao realizar uma varredura (*scanning*), detecta esses pacotes e pode solicitar uma conexão. Uma vez estabelecido o pareamento, o central identifica os serviços disponíveis no periférico e inicia a troca estruturada de dados. Além disso, esse modelo permite que o periférico otimize o consumo de energia, mantendo seu rádio desligado durante períodos de inatividade, enquanto o central assume as operações mais complexas (Bluetooth, 2024).

Figura 8 – Ilustração da conexão BLE de um dispositivo periférico (ESP32) com um celular (Android)

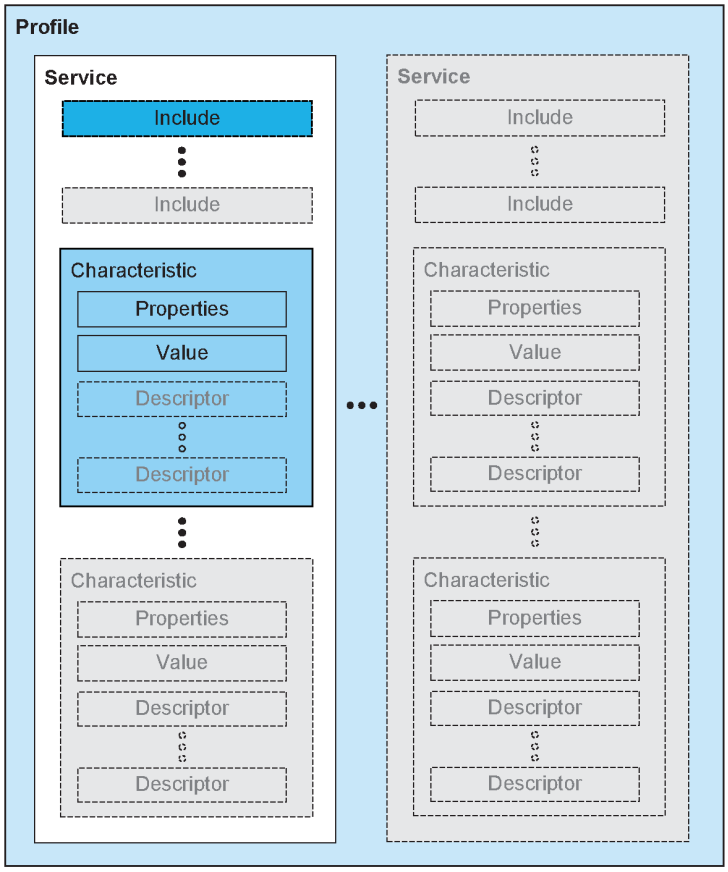


Fonte: Autoria própria.

A estrutura do BLE é organizada em camadas, com destaque para o perfil GATT (*Generic Attribute Profile*), responsável pela organização e transmissão dos dados

após a conexão. No GATT, as informações são agrupadas em serviços, que contêm características individuais identificadas por UUIDs (*Universally Unique Identifiers*). Por exemplo, um serviço dedicado a medições de temperatura pode incluir uma característica específica para o valor da leitura. A Figura 9 detalha essa hierarquia, mostrando como os perfis são compostos por serviços e características, cada uma com propriedades definidas (como leitura, escrita ou notificação). Essa padronização permite que dispositivos centrais, como smartphones Android, acessem dados de forma consistente, independentemente do periférico utilizado (Bluetooth SIG, 2025).

Figura 9 – Hierarquia GATT



Fonte: Bluetooth.

3 METODOLOGIA

3.1 DESENVOLVIMENTO DO FIRMWARE

3.1.1 Ambiente de Desenvolvimento

O firmware foi desenvolvido para a placa WeAct ESP32-C3FH4, baseada no microcontrolador ESP32-C3 da Espressif (China), que opera com arquitetura RISC-V de 32 bits, frequência de até 160 MHz, 400 kB de RAM, 384 kB de ROM, 4 MB de FLASH e até 22 pinos GPIO programáveis (WeAct Studio, 2022). Foi utilizada a linguagem MicroPython na versão 1.24.1, compatível com a arquitetura da placa, cuja imagem de firmware foi obtida no site oficial do projeto (MicroPython Team, 2024). A gravação do firmware e a transferência dos scripts foram realizadas por meio da Thonny IDE, comumente adotado para aplicações com MicroPython (Annamaa, 2024).

3.1.2 Estruturação do Projeto e Modularização

A arquitetura do firmware foi organizada em três diretórios principais (`lib`, `data`, `tasks`) complementados pelo arquivo `main.py`, responsável por coordenar a inicialização do sistema. No diretório `lib`, foram alocadas bibliotecas específicas, incluindo a `aioble`, uma biblioteca recomendada pela própria documentação do MicroPython para a maioria das aplicações com Bluetooth Low Energy (BLE) (MicroPython Contributors, 2025). Essa biblioteca abstrai a complexidade das operações de conexão, emparelhamento e troca de dados via GATT, facilitando a implementação de servidores BLE no ESP32-C3 (MicroPython Team, 2025). O subdiretório `utils` concentrou módulos auxiliares, como `bluetooth_config.py`, para definição de parâmetros de conexão, e `data_converter.py`, para serialização dos dados trocados com o aplicativo Android.

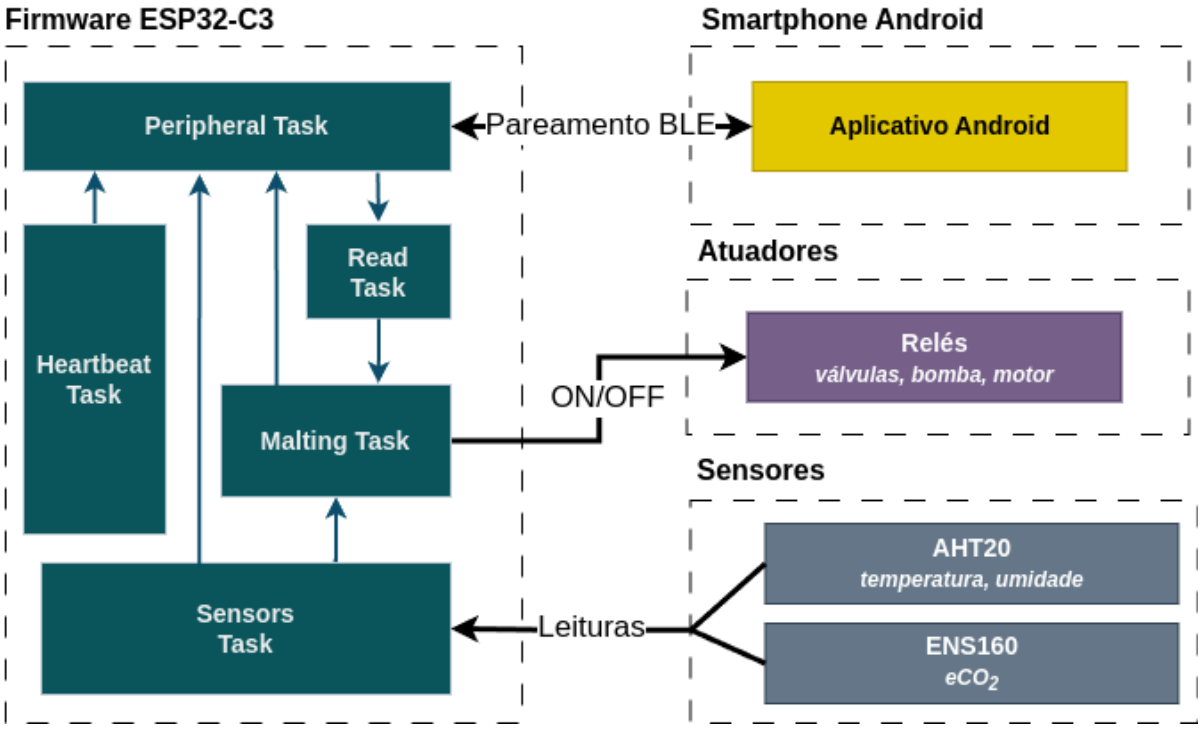
A persistência de configurações operacionais foi implementada no diretório `data` através do arquivo `config.json`, complementado por um conjunto de valores padrão em `default_data.py` como mecanismo de segurança para cenários de corrupção de dados salvos.

3.1.3 Gestão de Tarefas Assíncronas

A lógica central do firmware foi desenvolvida no arquivo `main.py`, onde foram instanciadas cinco corrotinas principais utilizando o módulo `asyncio` do MicroPython. A função `main()` coordena:

- `peripheral_task()`, responsável pelo ciclo de vida das conexões BLE;
- `send_heartbeat_task()`, que emite pacotes de sincronização a cada 2 segundos contendo métricas de integridade do sistema utilizando o módulo utilitário `memory_usage.py`;
- `read_task()`, dedicada ao processamento de dados recebidos via Bluetooth com despacho para handlers especializados;
- `malting_task()`, núcleo do controle do processo de malteação.
- `sensors_task()`, que gerencia a leitura periódica dos sensores instalados.

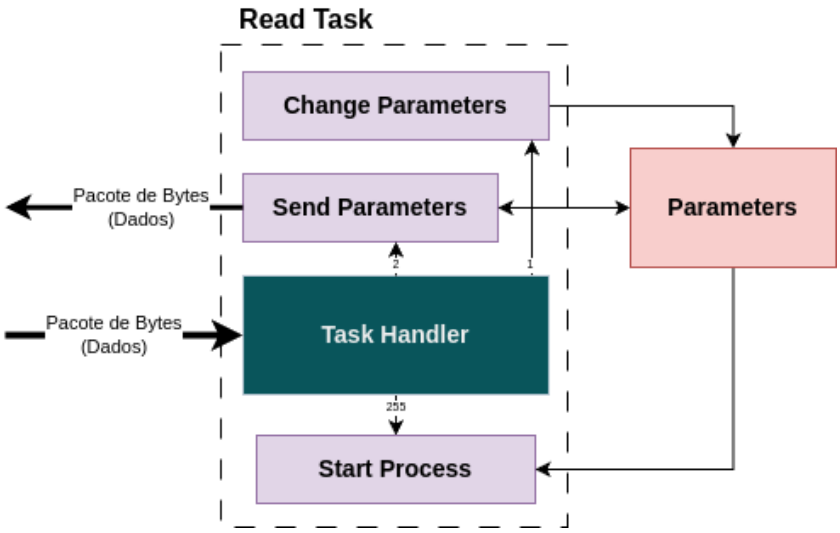
Figura 10 – Esquema geral do projeto com as tarefas assíncronas em evidência



Fonte: Autoria própria.

A tarefa `read_task` fica aguardando o recebimento de *bytes*. Quando um pacote é recebido, essa tarefa delega o tratamento dos dados para `task_handler`, que, por sua vez, direciona os dados para outros módulos de acordo com o primeiro *byte* do *array* (lista). Por exemplo, ilustrado pela Figura 11, se o primeiro *byte* for igual a 1, vai acionar o módulo de alteração dos parâmetros, que são utilizados para a realização da malteação — que é iniciada quando o *array* inicia com 255.

Figura 11 – Lógica do recebimento de comandos do dispositivo

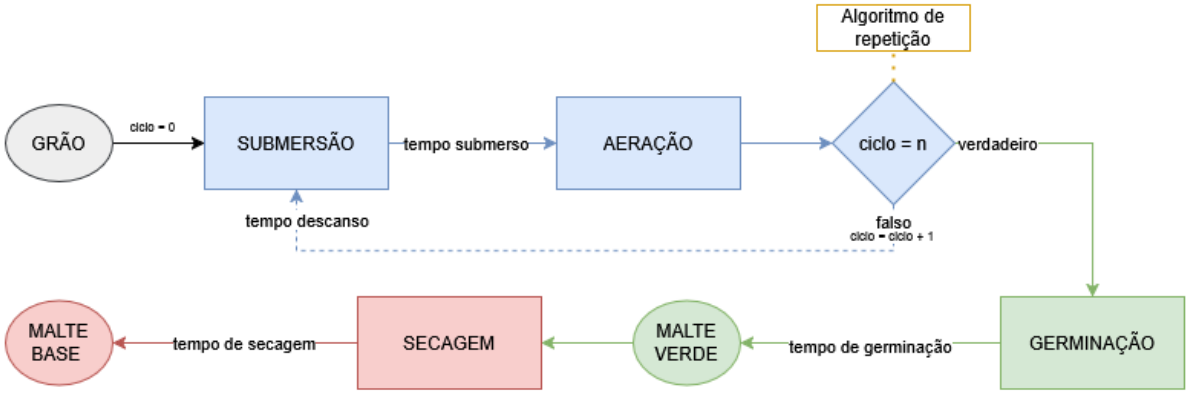


Fonte: Autoria própria.

3.1.4 Algoritmos de Controle do Processo de Malteação

A sequência de maceração, germinação e secagem foi modelada no subdiretório `malting_stages` em módulos especializados (`steeping.py`, `germination.py`, `kilning.py`). Cada etapa opera como uma corrotina independente, monitorando variáveis críticas com base nas condicionais dos parâmetros recebidos via Bluetooth. A estrutura assíncrona permitiu a execução não bloqueante desses processos, essencial para a continuidade da comunicação Bluetooth mesmo durante o processo. As lógicas de marcação de tempo foram desenvolvidas com o uso do módulo utilitário `uptime.py`, que se baseia no tempo total de execução do dispositivo.

Figura 12 – Fluxograma do controle planejado para o algoritmo de malteação



Fonte: Autoria própria.

3.2 DESENVOLVIMENTO DO APLICATIVO

3.2.1 Ambiente e Ferramentas de Desenvolvimento

O ambiente de desenvolvimento foi estabelecido utilizando a IDE Android Studio (Koala Feature Drop - 2024.1.2) com linguagem Kotlin na versão 1.9.10. Os testes foram conduzidos em um dispositivo físico com Android 14, garantindo compatibilidade com as APIs mais recentes do sistema operacional. Escolheu-se o ecossistema Android Jetpack como base tecnológica devido a sua estabilidade e suporte oficial da plataforma Google para desenvolvimento de aplicativos.

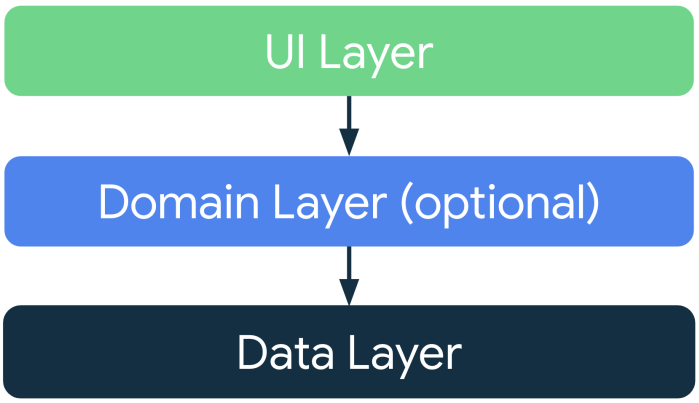
3.2.2 Arquitetura e Padrões de Projeto

Adotou-se uma arquitetura em camadas conforme as diretrizes de boas práticas do Android Moderno, estruturada em três módulos principais: *data*, *domain* e *presentation*. O módulo *data* concentra a gestão de fluxos de informação, incluindo a comunicação *Bluetooth* e o armazenamento local de dados, implementado com o uso das bibliotecas *Room* e *DataStore*. A *Room* é uma biblioteca de que facilita o acesso a bancos SQLite no sistema Android (Developers Android, 2025b). Já a *DataStore* é projetada para armazenar preferências e pequenas quantidades de dados de forma mais eficiente (Developers Android, 2025a).

Na camada *domain*, implementou-se a lógica de negócios e o tratamento de dados. Por fim, a camada de *presentation* foi construída com *Jetpack Compose*, utilizando os

princípios do *Design Material 3* que define diretrizes de usabilidade, hierarquia visual e acessibilidade para o desenvolvimento de interfaces (Google Material Design Team, 2024).

Figura 13 – Diagrama da arquitetura em camadas



Fonte: Developers Android.

3.3 DOCUMENTAÇÃO

A gestão do código foi implementada mediante a utilização do sistema de controle de versão Git, com hospedagem pública em dois repositórios GitHub dedicados: um para o firmware <https://github.com/NicolasDezan/ESP32C3-MaltingControl.git> e outro para o aplicativo Android <https://github.com/NicolasDezan/MALT-ESP.git>. A separação em repositórios distintos buscou garantir a modularidade entre os componentes de software, além de facilitar a manutenção independente de cada projeto.

4 RESULTADOS E DISCUSSÃO

4.1 ANÁLISE DO CÓDIGO

4.1.1 Comunicação Bluetooth

A comunicação entre o firmware e o aplicativo é realizada por meio do BLE, a qual impõe uma limitação significativa: por padrão, cada pacote transmitido ou recebido possui um tamanho útil máximo de 20 bytes. Isso ocorre porque a MTU (*Maximum Transmission Unit*) inicial, definida pela especificação Bluetooth, é de 23 bytes, sendo 3 reservados ao cabeçalho do protocolo ATT (*Attribute Protocol*), restando 20 bytes para os dados (Coleman, 2019). Com base nisso, o código foi estruturado de forma a otimizar o uso dos pacotes disponíveis, minimizando a quantidade de transmissões necessárias e, consequentemente, aumentando a eficiência da comunicação.

Para isso, optou-se por transmitir dados diretamente no formato `byte`, o que, por consequência, permite o envio de até 20 valores distintos em um único pacote. Essa abordagem, no entanto, limita os valores possíveis ao intervalo de 0 a 255, conforme a representação padrão de um `byte`. Em situações onde foi necessário ultrapassar esse limite, foi implementada uma lógica de fragmentação dos dados, permitindo a representação de valores mais elevados por meio da combinação de dois ou mais bytes. Com esse método, tornou-se possível transmitir qualquer número, com precisão suficiente para valores decimais simples.

Essa lógica foi aplicada, por exemplo, na transmissão da informação referente ao percentual de uso da memória RAM, que exige a representação de números com casas decimais. Nessa aplicação específica, o valor percentual é multiplicado por 100 e, em seguida, dividido em dois bytes para envio, sendo reconstituído com a operação inversa no aplicativo receptor (Algoritmo 4.1).

Algoritmo 4.1 – Tarefa periódica de envio de heartbeat

```
1 async def send_heartbeat_task():
2     while True:
3         _memory_usage_A = int(memory_usage())
4         _memory_usage_B = int((memory_usage()*100) % 100)
```

```
5
6           message = [
7                WriteList.HEARTBEAT,
8                _memory_usage_A,
9                _memory_usage_B
10          ]
11
12          bt.write_characteristic.write(bytes(message), send_update=True)
13          await asyncio.sleep(1)
14          send_actuators_state()
15          await asyncio.sleep(1)
16
17          classreadlist
```

A estratégia de triagem dos dados recebidos parte da utilização do primeiro byte do pacote como identificador, permitindo a categorização e o encaminhamento adequado de cada mensagem. Para facilitar essa operação e melhorar a legibilidade do código, foram implementadas duas classes – `ReadList` e `WriteList` – que reservam os 256 valores possíveis para identificadores (Algoritmo 4.2). Essa abordagem permite que cada valor atribuído seja mapeado a uma ação específica, otimizando o fluxo de comunicação entre o *firmware* e o aplicativo.

Para exemplificar, na função `def send_heartbeat_task()` (Algoritmo 4.1), a variável `message` representa um *array* construído por meio da classe `WriteList`, de onde é extraído o identificador correspondente. Este identificador é utilizado no aplicativo receptor para direcionar o tratamento da mensagem, garantindo a correta interpretação dos dados enviados.

Algoritmo 4.2 – Definição de constantes de identifação de pacote

```
1 class ReadList:
2     CHANGE_PARAMETERS = 0
3     SEND_PARAMETERS = 1
4     START_PROCESS = 255
5
6 class WriteList:
7     SEND_PARAMETERS = 1
8     SENSOR_VALUES = 2
```

```
9       SEND_ACTUATORS = 3
10      HEARTBEAT = 255
```

A lógica de comunicação implementada no aplicativo Android segue o mesmo princípio de identificação adotado no firmware. O primeiro byte do pacote recebido é utilizado como identificador, sendo interpretado para direcionar o tratamento adequado dos dados (Algoritmo 4.3). Por exemplo, quando esse identificador assume o valor 255, entende-se que o pacote refere-se ao sinal de *heartbeat* enviado periodicamente pelo ESP32, contendo também o percentual de uso da memória RAM (Algoritmo 4.4).

Algoritmo 4.3 – Seção de triagem dos dados recebidos (aplicativo)

```
1  onReadUpdate = { data ->
2      val convertedData = convertBytesESPToIntKotlin(data)
3
4      if (convertedData[0] == 1) {
5          println("Change Parameters: $convertedData")
6          _parametersReceived.value = ParametersMapper
7              .receiveParameters(convertedData)
8          sharedRepository.updateParameters(_parametersReceived.value)
9      }
10
11     if (convertedData[0] == 255) {
12         onPulseReceived()
13         val memUseData = convertedData[1].toFloat()
14             + convertedData[2].toFloat() / 100
15         updateMemoryUsage(memUseData)
16     }
17
18     if (convertedData[0] == 2) {
19
20         val humidity = convertedData[1].toFloat()
21             + convertedData[2].toFloat() / 100
22         val temperature = convertedData[3].toFloat()
23             + convertedData[4].toFloat() / 100
24
25         val eco2 = (convertedData[5] * 10000) + (convertedData[6] * 100)
26             + convertedData[7]
27
```

```
28          val sensorData = SensorReadData(
29              humidity = humidity,
30              temperature = temperature,
31              eco2 = eco2
32          )
33
34          updateSensorRead(sensorData)
35      }
36
37      if (convertedData[0] == 3) {
38
39          val actuatorState = ActuatorState(
40              entrada = convertedData[1] == 1,
41              saida = convertedData[2] == 1,
42              rotacao = convertedData[3] == 1,
43              resistencia = convertedData[4] == 1,
44              bombaAr = convertedData[5] == 1
45          )
46
47          updateActuatorState(actuatorState)
48      }
49  }
```

Algoritmo 4.4 – Tratamento do pacote identificado com ’255’

```
1
2  if (convertedData[0] == 255) {
3      onPulseReceived()
4      val memUseData = convertedData[1].toFloat()
5          + convertedData[2].toFloat() / 100
6      updateMemoryUsage(memUseData)
7  }
```

Nesse contexto, o aplicativo interpreta os bytes seguintes como valores numéricos que representam, respectivamente, a parte inteira e a parte decimal da porcentagem. Esses valores são somados após a conversão para o formato numérico adequado, resultando em um valor final com duas casas decimais (Algoritmo 4.4). Esse número é então utilizado para atualizar a interface do usuário com a informação atualizada sobre o consumo de memória do dispositivo.

Além da extração dos dados de uso de memória, o recebimento do identificador 255 também aciona uma função interna responsável por sinalizar a chegada do *heartbeat*. Esse mecanismo funciona como um pulso periódico enviado pelo firmware, cuja função principal é indicar que o dispositivo segue ativo e em comunicação com o aplicativo. No aplicativo, esse pulso ativa um recurso visual na interface: uma indicação em forma de círculo que pisca a cada novo recebimento. Esse indicador atua como um monitor de conexão em tempo real, permitindo ao usuário verificar de forma intuitiva se a comunicação Bluetooth está estável. Caso o pulso deixe de ser recebido por um intervalo prolongado, o usuário deve observar que ocorreu a perda de sinal ou falha de comunicação, evitando a falsa impressão de que o sistema permanece operacional quando, na verdade, pode estar congelado ou desconectado.

Esse modelo de tratamento baseado em identificadores garante clareza na separação das funções e facilita futuras expansões, mantendo o código modular e de fácil manutenção.

4.1.2 Parâmetros de malteação

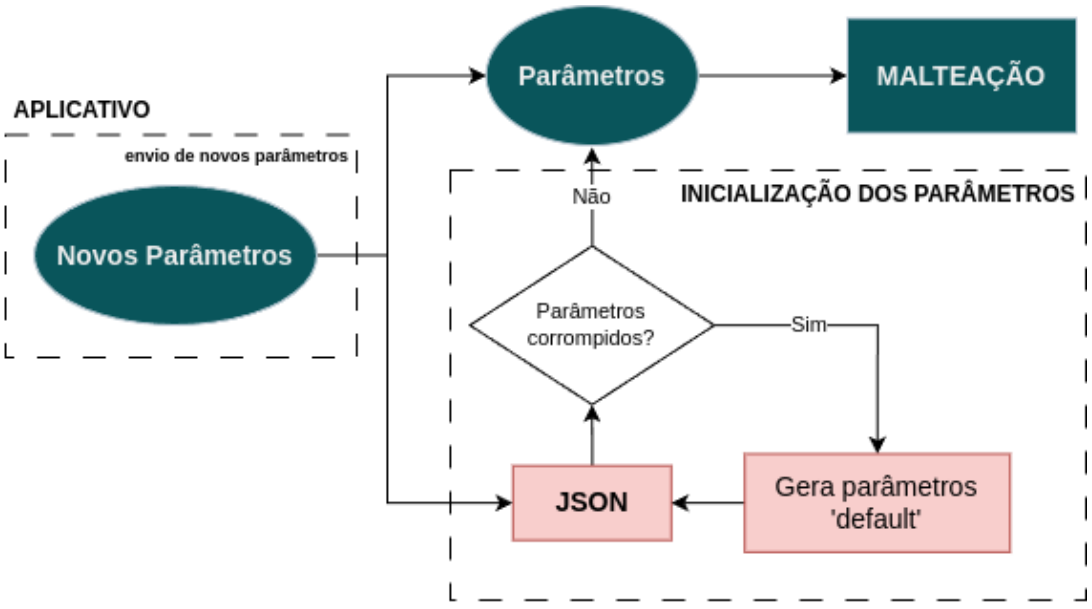
Os parâmetros de malteação correspondem às variáveis de controle utilizadas para configurar e ajustar o processo, como tempos de etapa, temperatura alvo e outros valores relevantes. Esses dados são essenciais para garantir a execução correta das etapas automatizadas.

Durante a inicialização do sistema, o firmware realiza a leitura de um arquivo de configuração persistente, armazenado localmente na memória do ESP32. Esse arquivo, no formato JSON, contém os valores salvos da última configuração utilizada. O conteúdo lido é mapeado internamente para as variáveis de controle do sistema, tornando-as acessíveis para os demais módulos do firmware.

Além da leitura, o sistema também permite a atualização desses parâmetros via comunicação Bluetooth. Sempre que o aplicativo envia novos valores, eles são armazenados temporariamente, validados e, posteriormente, gravados no arquivo de configuração. Isso garante que as alterações permaneçam ativas mesmo após desligamentos ou reinicializações.

A lógica geral de carregamento e persistência dos parâmetros é representada na Figura 14, que ilustra o fluxo completo de inicialização e atualização dos dados operacionais do malteador.

Figura 14 – Fluxo de inicialização e atualização dos parâmetros de malteação.



Fonte: Autoria própria.

Os parâmetros de malteação foram otimizados para que seus valores pudessem ser representados dentro da faixa de um único byte, permitindo até 256 variações distintas por parâmetro. Essa decisão buscou equilibrar a precisão necessária para o controle do processo com a eficiência na comunicação via Bluetooth, já que o envio de todos os parâmetros em um único pacote reduz o tempo de transmissão e minimiza o risco de perda de pacotes.

Embora essa abordagem limite a resolução de cada parâmetro, ela se mostra adequada frente às grandezas envolvidas no processo. Caso, em futuras otimizações, identifique-se a necessidade de representar valores com maior precisão, a arquitetura atual do sistema permite a adição de bytes extras sem impactos significativos na estrutura dos módulos.

A conversão dos valores numéricos para um único byte é feita por meio de uma transformação linear, na qual o valor transmitido (inteiro entre 0 e 255) é multiplicado por um fator de escala e somado a um valor mínimo admissível. Esse procedimento garante que a representação final esteja sempre dentro da faixa operacional válida

para cada variável.

As Tabelas 2, 3 e 4 apresentam os coeficientes utilizados para a reconstrução dos valores reais a partir dos dados recebidos via Bluetooth. Para cada parâmetro, aplica-se a seguinte fórmula:

$$\textit{valor real} = (\textit{byte recebido} \times \text{MULT\_TEN}) + \text{MIN\_VALUE}$$

Esse mesmo modelo é utilizado na conversão inversa, durante a serialização dos dados antes da transmissão, garantindo consistência entre os módulos do firmware e do aplicativo.

Tabela 2 – Coeficientes de reconstrução — Etapa de Maceração

Parâmetro	**MULT_TEN**	**MIN_VALUE**
Tempo Submerso	1.0	1.0
Volume de Água	10.0	200.0
Tempo de Descanso	0.1	0.0
Número de Ciclos	1.0	1.0

Fonte: Autoria própria.

Tabela 3 – Coeficientes de reconstrução — Etapa de Germinação

Parâmetro	**MULT_TEN**	**MIN_VALUE**
Nível de Rotação	1.0	0.0
Tempo Total	1.0	24.0
Volume de Água	1.0	100.0
Adição de Água	1.0	10.0

Fonte: Autoria própria.

Tabela 4 – Coeficientes de reconstrução — Etapa de Secagem

Parâmetro	**MULT_TEN**	**MIN_VALUE**
Temperatura	1.0	40.0
Tempo Total	0.1	1.0

Fonte: Autoria própria.

Na Tabela 5, são listados todos os parâmetros disponíveis no sistema, acompanhados de suas respectivas faixas de valores operacionais, definidas conforme os limites estabelecidos.

Tabela 5 – Faixa de valores possíveis para cada parâmetro.

Parâmetro	Etapa	Faixa de Valores	Unidade
Tempo Submerso	Maceração	1 – 256	horas
Volume de Água	Maceração	200 – 2750	mL
Tempo de Descanso	Maceração	0.0 – 25,5	horas
Número de Ciclos	Maceração	1 – 256	ciclos
Nível de Rotação	Germinação	0 – 255	–
Tempo Total	Germinação	24 – 279	horas
Volume de Água	Germinação	100 – 355	mL
Adição de Água	Germinação	10 – 265	minutos
Temperatura	Secagem	40 – 295	°C
Tempo Total	Secagem	1.0 – 26.5	horas

Fonte: Autoria própria.

4.1.3 Algoritmo da malteação

O controle automatizado da malteação foi implementado com base em uma lógica assíncrona, permitindo a execução simultânea de múltiplas tarefas relacionadas ao processo. A arquitetura do algoritmo principal é centrada em um sistema de estados, no qual cada etapa do processo (maceração, germinação e secagem) é tratada como um estágio independente, com mecanismos de inicialização, monitoramento e interrupção.

A execução do processo é iniciada a partir de um comando enviado pelo aplicativo, o que ativa a função responsável pela coordenação da malteação. Essa função inicializa o estado global e delega a execução para funções específicas de cada etapa, que operam com base nas configurações previamente definidas (parâmetros).

Durante toda a execução, o sistema monitora constantemente um sinal de interrupção, que pode ser acionado a qualquer momento via comunicação Bluetooth. Essa lógica garante que o processo possa ser encerrado de forma segura e imediata, sem comprometer a integridade das etapas já concluídas.

4.1.3.1 Maceração

A etapa de maceração é composta por dois fluxos assíncronos que atuam de forma paralela: o controle térmico, responsável por manter a temperatura adequada do sistema, e o controle de tempo, que regula a execução dos ciclos de hidratação.

Cada ciclo da maceração é formado por quatro subetapas executadas sequencialmente:

1. **Enchimento**: o reservatório é preenchido com água até o volume desejado.
2. **Imersão**: os grãos permanecem submersos pelo tempo programado.
3. **Drenagem**: a água é escoada do sistema.
4. **Repouso**: há uma pausa antes do próximo ciclo.

Essas fases são repetidas conforme o número de ciclos configurado pelo usuário. A execução da etapa de maceração pode ser representada de forma resumida na Tabela 6.

Tabela 6 – Fluxo de controle da etapa de maceração

Fase	Descrição
Inicialização	Define o estado do sistema como `steeping`, carrega os parâmetros de ciclo e inicia tarefas assíncronas.
Controle térmico	Mantém a temperatura dentro do intervalo alvo por meio de resfriamento ativo. Executado em paralelo.
Controle de tempo	Executa a sequência de subetapas (enchimento, imersão, drenagem, repouso) para cada ciclo.
Interrupção	A cada subetapa, o sistema verifica continuamente se houve solicitação de parada.
Finalização	Ao término dos ciclos ou interrupção, o sistema atualiza o estado para inativo.

Fonte: Autoria própria.

O controle de válvulas, tanto para enchimento quanto para drenagem, segue um modelo baseado no tempo de abertura, adicionando o volume necessário para a operação. Apesar de simplificado, esse método pode ser ajustado posteriormente com base

em calibrações que relacionem diretamente o tempo de válvula aberta ao volume de água processado. Esse aspecto é fundamental para permitir futuras melhorias sem a necessidade de reescrita da estrutura de controle.

A lógica de controle de temperatura foi implementada como uma tarefa paralela, executada em segundo plano com checagens periódicas e resposta imediata ao sinal de interrupção. Embora o protótipo atual ainda não possua capacidade física de controle térmico ativo (resfriamento), essa tarefa já está incorporada de forma modular no firmware, permitindo que sensores e atuadores possam ser integrados futuramente sem necessidade de grandes modificações estruturais. Essa abordagem garante escalabilidade e separação de responsabilidades, mantendo a coesão da arquitetura.

Ao final da execução dos ciclos de maceração, o sistema redefine o estado da etapa como inativo e passa automaticamente para o próximo estágio do processo, aguardando o início da germinação.

4.1.3.2 Germinação

A etapa de germinação (*germination*) mantém a estrutura introduzida na maceração, com múltiplas tarefas assíncronas atuando de forma coordenada. Ao entrar nesse estágio, o sistema atualiza o estado global e ativa quatro tarefas principais, que operam em paralelo:

- **Controle de tempo**: cronômetro responsável por definir a duração total da germinação.
- **Controle de temperatura**: ajusta ou monitora a temperatura ambiente.
- **Controle de umidade**: umidifica os grãos periodicamente.
- **Controle de rotação**: ativa mecanismos de movimentação dos grãos para arejamento.

Essas tarefas compartilham uma variável de sinalização comum, que permite a interrupção imediata da etapa caso o processo seja cancelado via aplicativo. A Tabela 7

resume a lógica operacional da etapa de germinação.

Tabela 7 – Fluxo de controle da etapa de germinação.

Fase	Descrição
Inicialização	Atualiza o estado global para `germination` e carrega os parâmetros definidos.
Controle temporal	Inicia o cronômetro principal da etapa. Ao fim do tempo total, sinaliza término da germinação.
Controle ambiental	Controla e monitora temperatura e umidade em paralelo ao tempo.
Controle de rotação	Aciona o sistema de movimentação dos grãos em intervalos regulares.
Interrupção	Todas as tarefas verificam sinal de parada a cada ciclo.
Finalização	Ao término do tempo ou interrupção, o sistema limpa o estado e prepara a etapa seguinte.

Fonte: Autoria própria.

A umidificação periódica dos grãos é feita por meio de ciclos adicionam de água. O algoritmo alterna entre a abertura de válvulas para umidificar o ambiente e pausas programadas para não encharcar os grãos. Essa tarefa permanece ativa enquanto o estágio estiver em execução, repetindo os ciclos conforme o necessário. O controle de temperatura segue o mesmo princípio adotado na etapa anterior, sendo executado como tarefa independente que verifica periodicamente o estado do sistema, permitindo ajustes térmicos futuros.

O reservatório de água do protótipo foi planejado para suportar 5 litros, estima-se que este valor seja suficiente para o processo. Apesar disso, caso seja necessário, a implementação de um sensor de distância ultrassom para acompanhamento do nível do tanque será simples dada a arquitetura do firmware.

Além disso, durante a germinação, o sistema realiza rotações constantes, para o revolvimento dos grãos com o objetivo de garantir a aeração e evitar o emaranhamento das radículas. Existe, também, leituras de CO_2 durante essa etapa para que seja possível emitir um alerta para o operador caso se atinja valores prejudiciais ao crescimento dos grãos.

4.1.3.3 Secagem

Encerrada a germinação, inicia-se a última etapa do processo: a secagem (*kilning*). Nesta fase, o objetivo é reduzir a umidade dos grãos e interromper a atividade enzimática, preservando as características desejadas do malte. A lógica geral mantém o padrão assíncrono das etapas anteriores, com três tarefas paralelas operando de forma coordenada: controle de tempo, temperatura e rotação.

A Tabela 8 detalha o fluxo operacional da etapa de secagem:

Tabela 8 – Fluxo de controle da etapa de secagem

Fase	Descrição
Inicialização	Atualiza o estado global para `kilning`
Controle temporal	Gerencia a duração total do estágio através de contagem regressiva baseada nos parâmetros
Controle térmico	Mantém temperatura elevada, utilizando ciclos de aquecimento e monitoramento contínuo
Controle de rotação	Ativa mecanismos de revolvimento em intervalos programados para homogeneização
Interrupção	Verificação contínua de sinal de parada durante todas as subetapas
Finalização	Desativa sistemas térmicos e mecânicos, atualiza estado para inativo

Fonte: Autoria própria.

O controle de tempo define a duração total do estágio, monitorando continuamente a contagem regressiva com base nos parâmetros definidos pelo usuário. Já a temperatura, nesse estágio, assume papel ainda mais crítico, visto que a secagem envolve temperaturas elevadas e potencialmente danosas ao produto se não forem adequadamente controladas.

Por fim, o controle de rotação continua presente, com o intuito de garantir a homogeneidade da secagem e evitar a formação de zonas com retenção de umidade. Essa movimentação periódica dos grãos deve assegurar uma secagem eficiente e uniforme.

A estrutura assíncrona aplicada em todas as etapas da malteação permite ao sistema operar com múltiplos controles simultâneos, com segurança, responsividade e flexibili-

dade. A separação lógica das funções facilita tanto a manutenção do código quanto a sua expansão, criando uma base para futuras implementações automatizadas, como a inclusão de um programa de torra.

4.1.4 Sensores e Atuadores

Nesta implementação, foram utilizados dois sensores, ambos com arquitetura baseada no protocolo I^2C responsáveis pela aquisição de dados para o processo:

Tabela 9 – Especificação dos sensores

Sensor	Parâmetros	Função no processo
AHT20	Temperatura e umidade relativa	Monitoramento contínuo das condições ambientais durante todas as etapas da malteação
ENS160	eCO_2 (equivalente de CO_2)	Detecção e monitoramento da atividade metabólica dos grãos na fase de germinação

Fonte: Autoria própria.

A leitura dos sensores ocorre de maneira contínua e assíncrona na tarefa `sensors_task`, garantindo atualização constante dos dados. Após cada leitura, os valores de temperatura, umidade e eCO_2 são armazenados em variáveis globais, facilitando o acesso por outras tarefas do sistema. Para fins de transmissão Bluetooth, os valores são convertidos em bytes e enviados periodicamente para o aplicativo dentro dessa própria função.

O tratamento de falhas também foi considerado: caso a leitura de qualquer sensor falhe, o sistema atribui valor zero como padrão, garantindo que o fluxo da comunicação continue sem interrupções.

Em paralelo à leitura de sensores, o sistema também gerencia cinco atuadores principais: duas válvulas (entrada e saída de água), um motor de rotação, uma resistência térmica e uma bomba de ar. A lógica de controle desses dispositivos é encapsulada na classe `Actuator` (Algoritmo 4.5), desenvolvida para simplificar a manipulação dos pinos GPIO e abstrair as diferenças de lógica de acionamento.

Algoritmo 4.5 – Classe Actuator - MicroPython

```
1 from machine import Pin
```

```
2
3  class Actuator:
4      def __init__(self, name: str, pin_number: int,
5                   active_high: bool = False):
6          """
7          name: Nome do atuador
8          pin_number: Numero do pino GPIO
9          active_high: Se True, ativa com nivel alto (1).
10         Se False, ativa com nivel baixo (0).
11             Default: False (logica invertida)
12         """
13         self.name = name
14         self.pin = Pin(pin_number, Pin.OUT)
15         self.active_high = active_high
16         self.state = False  # Inicia desligado
17         self.off()
18
19     def on(self):
20         self.pin.value(1 if self.active_high else 0)
21         self.state = True
22         print(f"[ACTUATOR] {self.name} ligado")
23
24     def off(self):
25         self.pin.value(0 if self.active_high else 1)
26         self.state = False
27         print(f"[ACTUATOR] {self.name} desligado")
28
29     def toggle(self):
30         if self.state:
31             self.off()
32         else:
33             self.on()
34
35     def is_on(self):
36         return self.state
```

Cada atuador é instanciado em `actuators` com seu respectivo nome e número de pino, e armazenado em um dicionário global para facilitar o acesso dinâmico por outras

partes do sistema. A classe (Algoritmo 4.5) disponibiliza métodos como `on()`, `off()` e `toggle()`, além de uma função de verificação de estado atual. Dessa forma, o acionamento de cada dispositivo torna-se simples, confiável e rastreável (via terminal).

Além do controle direto, foi implementada uma função de envio do estado atual dos atuadores via Bluetooth, permitindo que o aplicativo visualize, em tempo real, quais dispositivos estão ativos no momento.

4.2 APLICATIVO ANDROID

4.2.1 Tela Inicial: Conexão

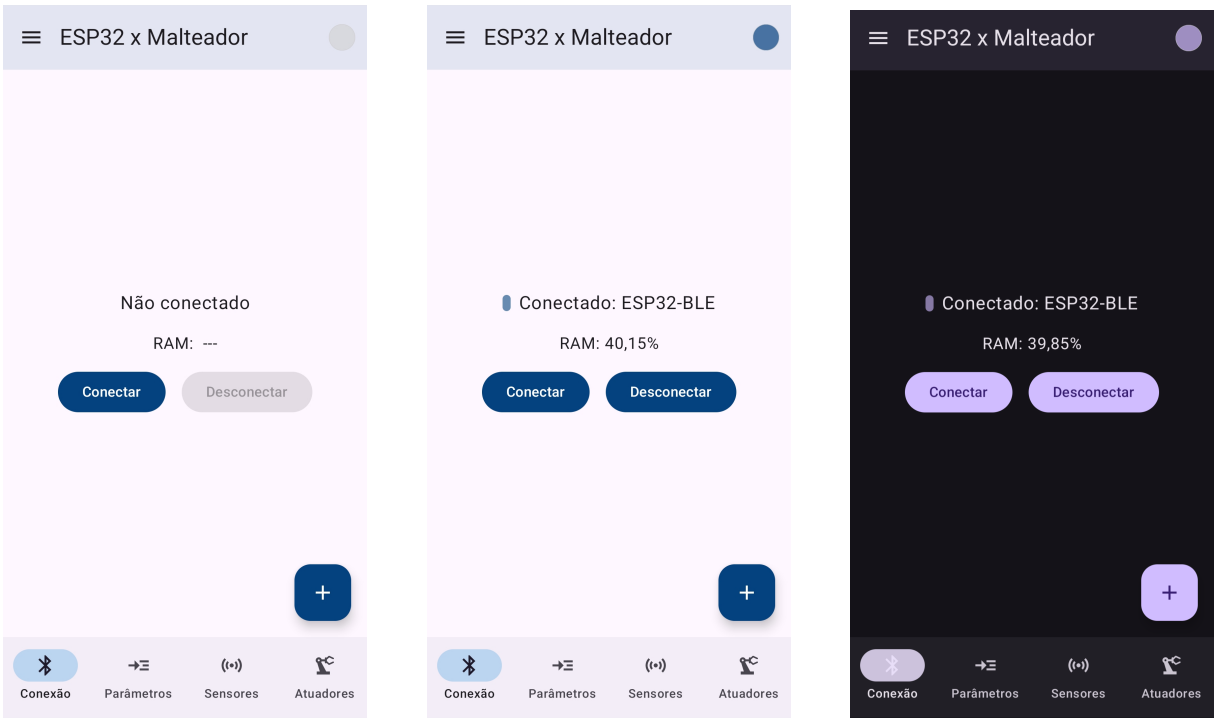
Ao inicializar o aplicativo, o usuário é direcionado à interface de conexão. Nesta tela, é exibido centralmente o status "Não conectado", acompanhado de um indicador de estado pulsante no canto superior direito, que alterna entre azul e cinza a cada segundo, quando conectado. A conexão com o dispositivo ESP32, previamente pareado via Bluetooth, é estabelecida mediante acionamento do botão "Conectar", conforme ilustrado na Figura 15.

Ainda, na estrutura geral do *Scaffold* (layout do aplicativo), destaca-se o *FloatingActionButton* (FAB) no canto inferior direito. Este componente, ao ser acionado, expande um menu com funcionalidades essenciais, incluindo:

- *Verificar Parâmetros*, para sincronização dos parâmetros com o ESP32;
- *Carregar Receita*, para autopreencher os parâmetros no aplicativo;
- *Iniciar*, para enviar os parâmetros e iniciar o processo de malteação.

No canto superior esquerdo, o botão de navegação proporciona acesso ao *Drawer*, um painel lateral com duas opções: "Início"(retorno à tela principal) e "Configurações"(Figura 16). A tela de configurações, em sua versão atual, permite a alternância entre temas claro e escuro (Figura 15), com previsão de expansão para personalizações avançadas, como definição de nomes de dispositivos e ajustes dos valores de UUID Bluetooth.

Figura 15 – Interface inicial do aplicativo.



(a) Desconectado (b) Conectado (c) Tema escuro

Fonte: Autoria própria.

Figura 16 – Drawer do aplicativo.

(a) Botão de acesso ao drawer (b) Drawer aberto (c) Tela de configurações acessada pelo drawer

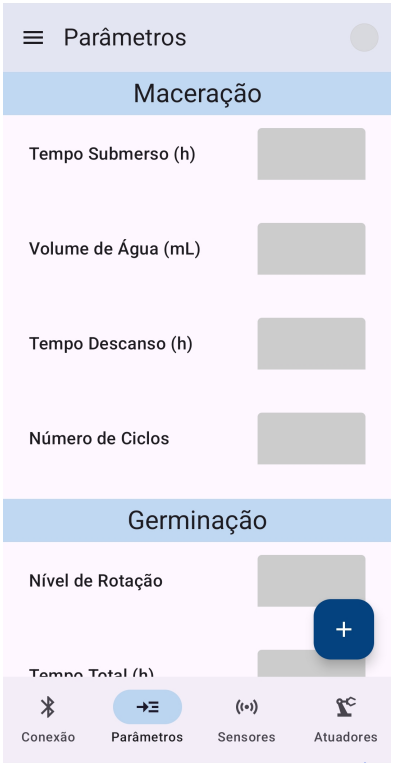
Fonte: Autoria própria.

4.2.2 Tela de Parâmetros

A interface de parâmetros opera em dois modos distintos, conforme o estado de conexão. No modo desconectado (Figura 17), os campos de entrada permanecem inativos. Quando conectado, os campos assumem codificação cromática dinâmica:

tonalidade amarela para indicar valores não sincronizados ou nulos, e azul para valores válidos sincronizados com o ESP32 (Figura 18). Valores fora da faixa operacional (**??**) ou inconsistentes geram feedback visual impedindo o envio (Figura 18), prevenindo possíveis erros de extrapolação dos bytes na comunicação Bluetooth.

Figura 17 – Tela de parâmetros em modo desconectado com campos inativos.



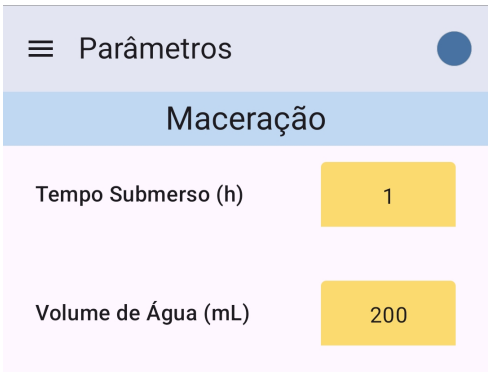
Fonte: Autoria própria.

Para melhor experiência do usuário/operador, o sistema permite o armazenamento das configurações atuais de parâmetros como perfis personalizados ("receitas"), mediante nomeação em diálogo dedicado (Figura 19). Essas receitas podem ser recuperadas através do menu do FAB, que exibe uma lista de opções salvas.

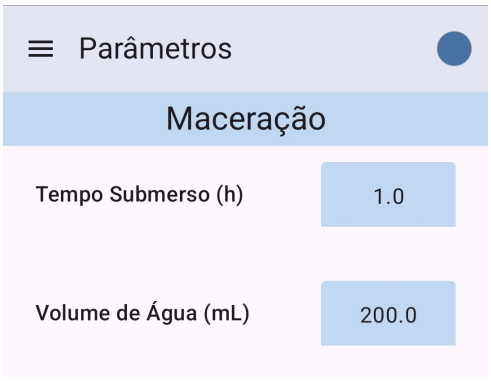
4.2.3 Telas de Monitoramento: Sensores e Atuadores

As telas de sensores (Figura 20) e atuadores (Figura 21) cumprem função de monitoramento passivo, sem permitir interações diretas. A primeira exibe dados em tempo real de temperatura, umidade e concentração de CO_2, enquanto a segunda apresenta o estado operacional dos componentes físicos (válvulas, resistência, bomba de ar), com indicadores *ON/OFF* gráficos. Ambas integram-se à *BottomAppBar*, mantendo coerência na navegação e permitindo rápida alternância entre telas.

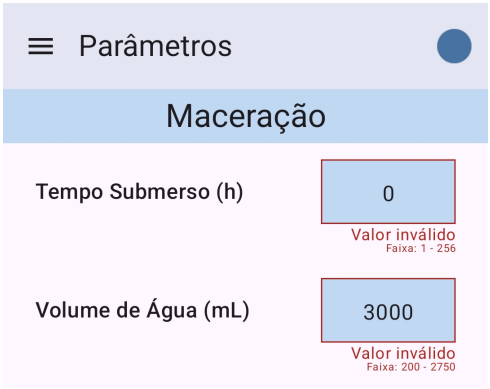
Figura 18 – Estados dos campos de parâmetros.



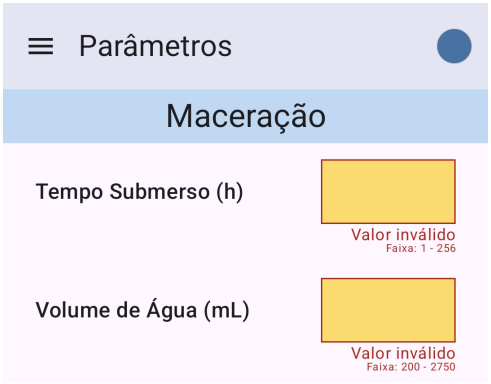
(a) Campos de parâmetros em estado não sincronizado



(b) Campos de parâmetros válidos sincronizados



(c) Feedback visual para valores fora da faixa operacional



(d) Feedback visual para valores nulos

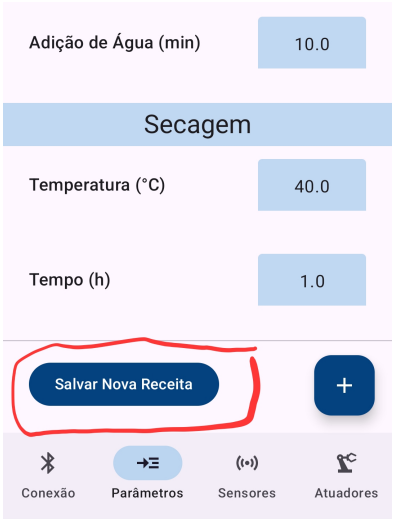
Fonte: Autoria própria.

4.3 LIMITAÇÕES E TRABALHOS FUTUROS

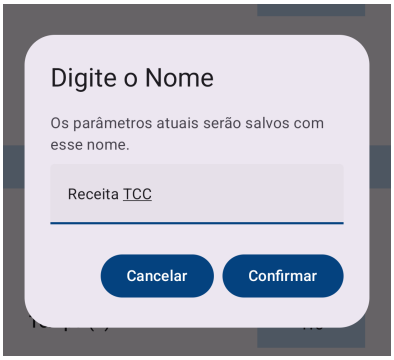
Embora a arquitetura de software esteja praticamente pronta, o presente trabalho apresenta limitações no que diz respeito à validação prática do sistema. O protótipo físico desenvolvido anteriormente, durante a IC, permanece incompleto em sua estrutura elétrica e nos componentes de automação, o que inviabilizou a realização de testes físicos com os controles implementados.

Como consequência, os testes foram realizados sem a ativação real dos atuadores. Os sensores AHT20 e ENS160 foram conectados ao sistema e realizaram leituras ambientais reais . O comportamento do código foi verificado por meio de mensagens no terminal, confirmando que os trechos responsáveis pelo acionamento de dispositivos estão sendo corretamente executados. Essa abordagem teve como objetivo validar

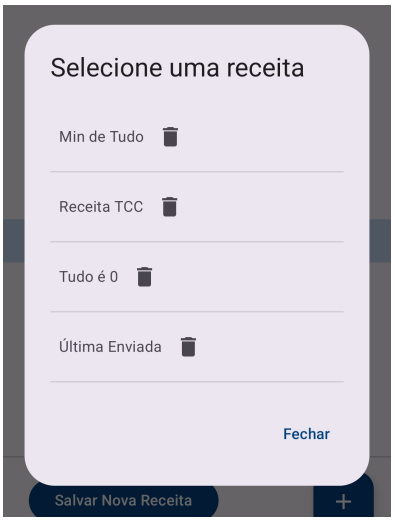
Figura 19 – Salvamento e seleção de receitas.



(a) Botão de salvamento de receitas



(b) Entrada de nome para salvamento de receita personalizada



(c) Seleção de receitas salvas

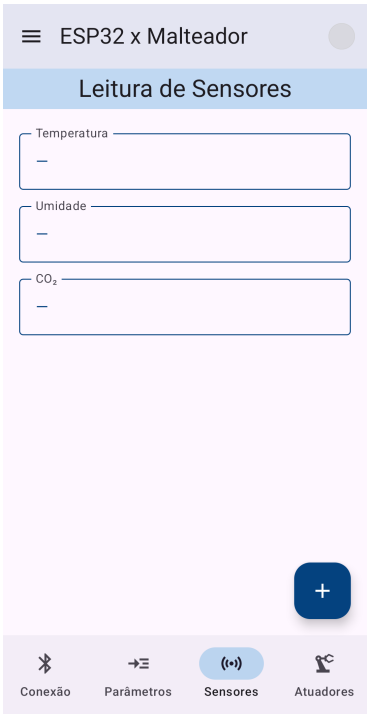
Fonte: Autoria própria.

a lógica de controle assíncrona e a estabilidade da comunicação via Bluetooth. A simulação demonstrou-se eficaz para comprovar o funcionamento da arquitetura em condições normais de operação.

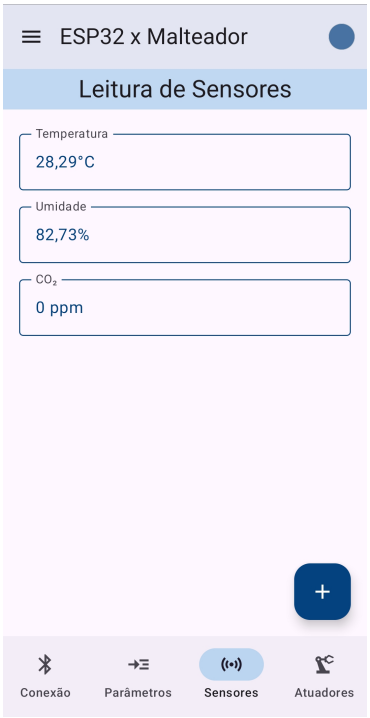
Para trabalhos futuros, recomenda-se:

- Finalização do protótipo físico com sensores e atuadores totalmente integrados;
- Avaliação da eficiência do sistema em testes com diferentes perfis de malteação;
- Implementação de novos recursos, como controle de torrefação e exportação de dados históricos.

Figura 20 – Interface de sensores.



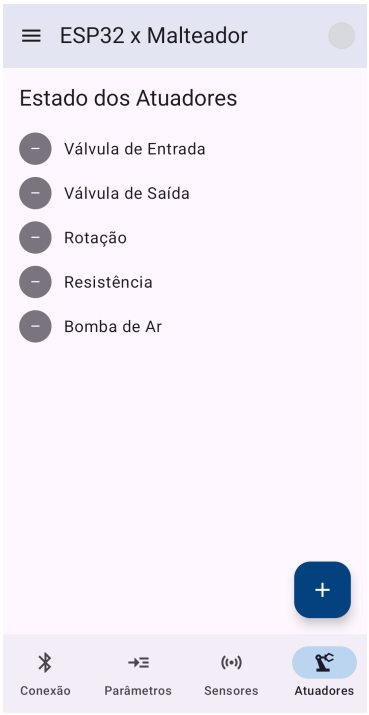
(a) Interface de sensores em modo desconectado



(b) Interface de sensores em modo conectado

Fonte: Autoria própria.

Figura 21 – Interface de atuadores.



(a) Estado dos atuadores em modo desconectado

(b) Interface de atuadores em modo conectado

Fonte: Autoria própria.

5 CONCLUSÃO

Este Trabalho de Conclusão de Curso teve como objetivo principal o desenvolvimento de uma solução de software composta por um firmware e um aplicativo Android para automatizar o processo de malteação em ambiente laboratorial. A proposta surgiu como resposta à demanda do LACEMP-IFES, que busca um equipamento acessível para a realização de experimentos controlados de malteação, especialmente em condições climáticas desfavoráveis.

O sistema foi implementado utilizando o microcontrolador ESP32-C3, programado em MicroPython, com algoritmos estruturados de forma assíncrona, capazes de controlar as etapas de maceração, germinação e secagem. O aplicativo Android, desenvolvido em Kotlin com uso do Jetpack Compose, permite a configuração remota de parâmetros, o monitoramento em tempo real de sensores e atuadores, bem como o armazenamento local de receitas para diferentes perfis de malteação. A comunicação entre os dispositivos foi estabelecida via Bluetooth Low Energy (BLE).

Os resultados alcançados demonstraram que a arquitetura modular adotada é satisfatória, sendo flexível e escalável. As funcionalidades essenciais para o controle do processo foram implementadas com sucesso, e o sistema provou ser capaz de manter a comunicação estável, registrar dados operacionais via terminal e responder a comandos externos de forma confiável. Ainda que o protótipo físico original desenvolvido durante a IC possua limitações que impedem a integração imediata do controle térmico real, a lógica de controle foi devidamente implementada e encontra-se preparada para operação assim que o hardware for ajustado.

Com isso, o presente trabalho entrega não apenas uma plataforma funcional de automação voltada ao malteador laboratorial, mas também uma base modular e documentada de desenvolvimento para sistemas embarcados com ESP32. O firmware e o aplicativo Android, organizados em repositórios públicos, foram estruturados de forma a facilitar sua adaptação para outras aplicações acadêmicas, especialmente na prototipação de equipamentos laboratoriais de baixo custo, como tituladores automáticos, espectrofotômetros e controladores de processo. Assim, além de atender a uma demanda específica do LACEMP-IFES, este trabalho se propõe a servir como ponto de partida

para futuros projetos orientados na interseção entre Química e IoT, contribuindo com uma trilha prática e acessível para estudantes que desejem explorar a automação em ambientes educacionais e de pesquisa.

Como trabalhos futuros, recomenda-se a montagem e validação experimental do sistema completo com sensores e atuadores reais, a implementação de um módulo adicional para controle da etapa de torrefação (maltes especiais) e a ampliação da interface do aplicativo com funcionalidades voltadas à exportação de dados via Wi-Fi. Essas melhorias ampliarão o potencial da plataforma para experimentos reprodutíveis, análises mais refinadas e aplicação direta em práticas laboratoriais, fortalecendo sua utilidade tanto em pesquisas quanto no ensino técnico sobre a malteação.

Apesar dessas conquistas, destaca-se que a validação prática do sistema em hardware físico completo ainda não foi realizada, sendo este um passo fundamental para futuras aplicações experimentais no LACEMP. A modularidade da arquitetura permite que novos dispositivos sejam integrados facilmente, garantindo escalabilidade. Com os ajustes necessários no protótipo físico, o sistema está pronto para ser incorporado em experimentos reais de malteação, contribuindo para pesquisas sobre novas variedades de grãos, controle de processos e desenvolvimento de tecnologias acessíveis para a indústria cervejeira.

REFERÊNCIAS

ABLESON, F.; KING, C.; ORTIZ, C. E. **Android in action**. [S.l.]: Simon and Schuster, 2011.

Android Developers. **Android Studio**. 2023. Disponível em: <https://developer.android.com/studio>.

Android Developers. **Bluetooth overview**. 2024. Disponível em: <https://developer.android.com/develop/connectivity/bluetooth>.

ANNAMAA, A. **Thonny - Python IDE for Beginners**. 2024. Acessado em 05 de junho de 2025. Disponível em: <https://thonny.org/>.

BAMFORTH, C. W. Beers: Chemistry of brewing a2. In: **Encyclopedia of Food Sciences and Nutrition - Second Edition**. Oxford: [s.n.], 2003. p. 440–447.

BAXTER, E.; REEVES, S.; BAMFORTH, C. The effects of increased steeping temperatures on enzyme development in malt. **Journal of the Institute of Brewing**, Wiley Online Library, v. 86, n. 4, p. 182–185, 1980.

Bluetooth. **The Bluetooth® Low Energy Primer**. 2024. Acessado em: 4 jun. 2025. Disponível em: <https://www.bluetooth.com/wp-content/uploads/2022/05/the-bluetooth-le-primer-v1.2.0.pdf>.

Bluetooth SIG. **GATT Specification Supplement**. 2025. Acessado em: 4 jun. 2025. Disponível em: <https://btprodspecificationrefs.blob.core.windows.net/gatt-specification-supplement/GATT_Specification_Supplement.pdf>.

BOBADE, H.; GUPTA, A.; SHARMA, S. Beta-glucan. In: **Nutraceuticals and health care**. [S.l.]: Elsevier, 2022. p. 343–358.

BOLTON, W. **Instrumentation and control systems**. [S.l.]: Newnes, 2021.

BRIGGS, D. E. *et al.* **Brewing: Science and Practice**. [S.l.]: Elsevier, 2004.

BRITO, J. J. d. **Automação de reator químico usando Micropython e sistema supervisório**. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2020.

CECCARONI, D. *et al.* Rice malting optimization for the production of top-fermented gluten-free beer. **Journal of the Science of Food and Agriculture**, v. 99, n. 6, p. 2726–2734, 2019.

CENCI, I. d. O. *et al.* **Cevada e Malteação**. 1. ed. Curitiba: Appris Editora, 2021.

COGHE, S. *et al.* Sensory and instrumental flavour analysis of wort brewed with dark specialty malts. **Journal of the Institute of Brewing**, v. 110, n. 2, p. 94–103, 2004.

COLEMAN, C. **A Practical Guide to BLE Throughput**. 2019. Acessado em 05 de junho de 2025. Disponível em: <https://interrupt.memfault.com/blog/ble-throughput-primer>.

Developers Android. **DataStore**. 2025. Acessado em 05 de junho de 2025. Disponível em: <https://developer.android.com/topic/libraries/architecture/datastore?hl=pt-br>.

Developers Android. **Salvar dados em um banco de dados local usando o Room**. 2025. Acessado em 05 de junho de 2025. Disponível em: <https://developer.android.com/training/data-storage/room?hl=pt-br>.

DHINGRA, S. *et al.* Internet of things mobile–air pollution monitoring system (iot-mobair). **IEEE Internet of Things Journal**, IEEE, v. 6, n. 3, p. 5577–5584, 2019.

DOUGLAS, P.; FLANNIGAN, B. A microbiological evaluation of barley malt production. **Journal of the Institute of Brewing**, Wiley Online Library, v. 94, n. 2, p. 85–88, 1988.

Espressif Systems. **ESP32-C3 Series Datasheet Version 2.1**. [S.l.], 2025. Accessed on 2025-06-03. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32-c3_datasheet_en.pdf>.

FARZANEH, V. *et al.* The impact of germination time on the some selected parameters through malting process. **International journal of biological macromolecules**, Elsevier, v. 94, p. 663–668, 2017.

FERENCZ, K.; DOMOKOS, J. Rapid prototyping of iot applications for the industry. In: IEEE. **2020 IEEE international conference on automation, quality and testing, robotics (AQTR)**. [S.l.], 2020. p. 1–6.

FOX, G. P. Chemical composition in barley grains and malt quality. In: **Genetics and improvement of barley malt quality**. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. p. 63–98.

Google Material Design Team. **Material Design 3**. 2024. Acessado em 05 de junho de 2025. Disponível em: <https://m3.material.io/>.

GRIFFITHS, M. Malt kilning for flavour and colour. **Brew. Guardian**, v. 121, p. 29–34, 1992.

GUPTA, M.; ABU-GHANNAM, N.; GALLAGHAR, E. Barley for brewing: Characteristic changes during malting, brewing and applications of its by-products. **Comprehensive Reviews in Food Science and Food Safety**, v. 9, n. 3, p. 318–328, 2010.

HEYDON, R.; HUNN, N. Bluetooth low energy. **CSR Presentation, Bluetooth SIG https://www. bluetooth. org/DocMan/handlers/DownloadDoc. ashx**, v. 32, 2012.

JEMEROV, D.; ISAKOVA, S. **Kotlin in action**. [S.l.]: Simon and Schuster, 2017.

KOISTINEN, V. M. *et al.* Side-stream products of malting: a neglected source of phytochemicals. **npj Science of Food**, v. 4, n. 1, p. 21, 2020.

KOLBAN, N. Kolban's book on esp32. **Texas, USA**, 2017.

KOVALOVA, O. *et al.* **Development of oat malt production technology using plasma-chemically activated aqueous solutions**. 2024. (Informações de publicação não fornecidas).

KUNZE, W. **Technology Brewing and Malting: Yeast**. [S.l.]: VLB, 1996.

LEWIS, M. J.; YOUNG, T. W. **Brewing**. [S.l.]: Springer Science & Business Media, 2012.

LUARASI, L.; TROJA, R.; PINGULI, L. Microbiological safety and quality evaluation of the raw materials used in beer production. In: **(Evento não especificado)**. [S.l.: s.n.], 2016.

MALLETT, J. **Malte: um guia prático do campo à cervejaria**. [S.l.]: Editora Krater, 2022.

MAYER, H. *et al.* Production of a saccharifying rice malt for brewing using different rice varieties and malting parameters. **Journal of Agricultural and Food Chemistry**, v. 62, n. 23, p. 5369–5377, 2014.

MicroPython Contributors. **bluetooth — low-level Bluetooth**. 2025. Acessado em 05 de junho de 2025. Disponível em: <https://docs.micropython.org/en/latest/library/bluetooth.html>.

MicroPython Team. **MicroPython Download - ESP32-C3**. 2024. Acessado em 05 de junho de 2025. Disponível em: <https://micropython.org/download/ESP32_GENERIC_C3/>.

MicroPython Team. **aioble - Bluetooth Low Energy API for MicroPython**. 2025. Acessado em 05 de junho de 2025. Disponível em: <https://github.com/micropython/micropython-lib/tree/master/micropython/bluetooth/aioble>.

MOHANASUNDARAM, R. *et al.* Water quality monitoring and control in urban areas in real-time via iot and mobile applications. In: IEEE. **2024 3rd International Conference on Artificial Intelligence For Internet of Things (AIIoT)**. [S.l.], 2024. p. 1–5.

MONTANUCI, F. D. *et al.* Effect of steeping time and temperature on malting process. **Journal of Food Process Engineering**, Wiley Online Library, v. 40, n. 4, p. e12519, 2017.

NASCIMENTO, J. F. do; CICHACZEWSKI, E. Internet das coisas (iot) aplicada ao monitoramento do nível de água em reservatórios domésticos. **Caderno Progressus**, v. 1, n. 2, p. 34–48, 2021.

NURSTEN, H. E. **The Maillard reaction: chemistry, biochemistry, and implications**. [S.l.]: Royal Society of Chemistry, 2005.

PETTERS, H.; FLANNIGAN, B.; AUSTIN, B. Quantitative and qualitative studies of the microflora of barley malt production. **Journal of Applied Microbiology**, Blackwell Science Ltd Oxford, UK, v. 65, n. 4, p. 279–297, 1988.

PITZ, W. An analysis of malting research. **Journal of the American Society of Brewing Chemists**, Taylor & Francis, v. 48, n. 1, p. 33–44, 1990.

PLAUSKA, I.; LIUTKEVIČIUS, A.; JANAVIČIŪTĖ, A. Performance evaluation of c/c++, micropython, rust and tinygo programming languages on esp32 microcontroller. **Electronics**, MDPI, v. 12, n. 1, p. 143, 2022.

REYNOLDS, T.; MACWILLIAM, I. C. Water uptake and enzymic activity during steeping of barley. **Journal of the Institute of Brewing**, v. 72, n. 2, p. 166–170, 1966.

RODRIGUES, L. D. *et al.* Controle de temperatura e de umidade de uma composteira utilizando o esp32. In: SBC. **Congresso Brasileiro de Agroinformática (SBIAGRO)**. [S.l.], 2021. p. 53–61.

ROP, O.; MLCEK, J.; JURIKOVA, T. Beta-glucans in higher fungi and their health effects. **Nutrition reviews**, Oxford University Press Oxford, UK, v. 67, n. 11, p. 624–631, 2009.

SANTANA, L. **Como o Google está usando a linguagem de programação Kotlin para eliminar bugs de código**. SempreUpdate, 2020. Acessado em: 28 mai. 2025. Disponível em: <https://sempreupdate.com.br/android/como-o-google-esta-usando-a-linguagem-de-programacao-kotlin-para-eliminar-bugs-de-codigo/>.

SANTOS, J. W.; JUNIOR, R. C. d. L. **Sistema de automatização residencial de baixo custo controlado pelo microcontrolador esp32 e monitorado via smartphone**. Dissertação (B.S. thesis) — Universidade Tecnológica Federal do Paraná, 2019.

SEBORG, D. E. *et al.* **Process dynamics and control**. [S.l.]: John Wiley & Sons, 2016.

SHINDE, K. S.; BHAGAT, P. H. Industrial process monitoring using Iot. In: IEEE. **2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)**. [S.l.], 2017. p. 38–42.

SILLS, B. *et al.* **Android Programming: The Big Nerd Ranch Guide**. 5. ed. Atlanta, GA: Big Nerd Ranch, LLC, 2023. Fifth edition, second printing, February 2023. ISBN 978-0137645633.

SKENDI, A.; PAPAGEORGIOU, M. Influence of kilning temperature on chemical composition of a greek barley malt and its wort properties. **Millenium-Journal of Education, Technologies, and Health**, n. 7, p. 49–58, 2018.

TANGANELLI, G.; VALLATI, C.; MINGOZZI, E. Rapid prototyping of iot solutions: A developer's perspective. **IEEE Internet Computing**, IEEE, v. 23, n. 4, p. 43–52, 2019.

TANGNI, E.; LARONDELLE, Y. Malts, moulds and mycotoxins. In: **Bacteria, yeasts and moulds in malting and brewing: Proceedings of the Xth symposium "Chair J de Clerck", Leuven (Belgium)**. [S.l.: s.n.], 2002.

TOLLERVEY, N. H. **Programming with MicroPython: embedded programming with microcontrollers and Python**. [S.l.]: "O'Reilly Media, Inc.", 2017.

TURNER, H. M. *et al.* Effect of steeping regime on barley malt quality and its impacts on breeding program selection. **Journal of the American Society of Brewing Chemists**, v. 77, n. 4, p. 267–281, 2019.

VISAKH, P. *et al.* Starch-based bionanocomposites: Processing and properties. **Polysaccharide building blocks: A sustainable approach to the development of renewable biomaterials**, Wiley Online Library, p. 287–306, 2012.

WeAct Studio. **ESP32C3 Core Board - Documentation**. 2022. Acessado em 05 de junho de 2025. Disponível em: <https://github.com/WeActStudio/WeActStudio.ESP32C3CoreBoard>.

WILHELMSON, A. *et al.* Oxygen deficiency in barley (hordeum vulgare) grain during malting. **Journal of agricultural and food chemistry**, ACS Publications, v. 54, n. 2, p. 409–416, 2006.

WOFFENDEN, H. M. *et al.* Effect of kilning on the antioxidant and pro-oxidant activities of pale malts. **Journal of Agricultural and Food Chemistry**, v. 50, n. 17, p. 4925–4933, 2002.