Desenvolvimento de um Sistema Embarcado para coleta de dados de vibração em Drones utilizando a Densidade de Máximos

MATEUS ANTONIO DA SILVA

CENTRO DE INFORMÁTICA
UNIVERSIDADE FEDERAL DA PARAÍBA

João Pessoa, 2024

MATEUS ANTONIO DA SILVA

Desenvolvimento de um Sistema Embarcado para coleta de dados de vibração em Drones utilizando a Densidade de Máximos

Monografia apresentada ao curso Engenharia de Computação
do Centro de Informática, da Universidade Federal da Paraíba,
como requisito para a obtenção do grau de Bacharel em
Engenharia de Computação

Orientador: Dr. Alisson Vasconcelos De Brito

Junho de 2024

Catalogação na publicação
Seção de Catalogação e Classificação

S586d Silva, Mateus Antonio da.
Desenvolvimento de um sistema embarcado para coleta de dados de vibração em drones utilizando a densidade de máximos / Mateus Antonio da Silva. - João Pessoa, 2024.
62 f. : il.

Orientação: Alisson Brito.
TCC (Graduação) - UFPB/CI.

1. ESP32. 2. SAC-DM. 3. Detecção de falhas. 4. Veículos aéreos não tripulados. 5. Drones. I. Brito, Alisson. II. Título.

UFPB/CI CDU 004:629.735

Elaborado por Michelle de Kássia Fonseca Barbosa - CRB-738

"Procure uma necessidade e comece a ter ideias para atender essa necessidade. Uma ideia vai levar a outra e sem perceber, conseguiu! Viu a necessidade? Atenda!" ROBÔS. Direção: Chris Wedge, Carlos Saldanha. Produção: Blue Sky Studios, 20th Century Animation. Estados Unidos: 20th Century Fox, 2005. Disney+.

AGRADECIMENTOS

Dedico esse espaço para agradecer a todos que contribuíram para o encerramento deste ciclo.

Em primeiro lugar a Deus, pela minha vida, e por me permitir ultrapassar todos os obstáculos encontrados ao longo de toda a graduação, com saúde e determinação.

Aos meus pais e irmão, que me incentivaram desde o início a seguir o caminho que escolhi, dando todo o suporte necessário para que eu chegasse até aqui. O apoio provido por eles foi essencial durante a jornada.

Aos meus amigos mais próximos que foram extremamente importantes nesse percurso. Mateus, Luiz, Lívia, João Victor, Gustavo, Lenildo, Alberto, Vinícius e Laura. Seu suporte e apoio vieram de diversas formas e sou muito grato por isso.

Aos colegas e amigos da graduação, que de diversas formas colaboraram para meu crescimento. Não caberiam aqui tantos nomes que gostaria de citar, então destaco os grupos para que saibam o quão importante foram para mim. O IEEE e seus membros, que me fizeram perceber o quão importante é ajudar os outros, realizar trabalhos voluntários e dar um retorno a comunidade que a Universidade está inserida. Dentro do IEEE, destaco a RAS, por ter mostrado o quanto gosto de ensinar, principalmente aprender, para compartilhar conhecimento e ajudar os outros. Os meus colegas de turma, que estavam juntos nos momentos de risadas e nos momentos de desespero durante as fases da graduação. Aos amigos e pesquisadores do LASER, que estavam sempre lá, para trabalhar e conversar, tornando tudo mais leve.

Agradeço também aos professores do LASER, principalmente Alisson e Anand, que foram verdadeiros guias em diversos momentos, contribuindo verdadeiramente para o meu crescimento acadêmico.

Por fim, agradeço à minha namorada e colegas de trabalho de Campinas, que na reta final contribuiram também com conhecimento técnico e apoio, influenciando na escrita deste trabalho.

RESUMO

Os Veículos Aéreos Não Tripulados (VANTs), também conhecidos como drones, são amplamente utilizados em uma variedade de aplicações devido à sua versatilidade e capacidade de coleta de dados em tempo real. No entanto, o monitoramento do desempenho desses dispositivos durante o voo é crucial para garantir sua operação segura e eficiente. A pesquisa e desenvolvimento de meios para o aumento da confiabilidade deles é essencial para garantir não só a segurança do equipamento, que costuma ser caro, como também a segurança das pessoas que operam e estão envolvida no seu uso. Este trabalho propõe o desenvolvimento de um sistema embarcado baseado na Teoria do Caos, especificamente na Análise de Sinal Baseada no Caos utilizando Densidade de Máximos (SAC-DM), para coletar e processar dados de vibração em tempo real. O sistema é projetado utilizando um kit de desenvolvimento com ESP32 e uma Unidade de Medição Inercial (IMU), além de outros componentes, com o intuito de identificar padrões de vibração indicativos de possíveis falhas no drone, aumentando assim a confiabilidade e a segurança operacional. Além disso, as informações serão enviadas para uma plataforma na nuvem para análises e diagnósticos remotos.

Palavras-chave: <ESP32>, <SAC-DM>, <Detecção de Falhas>, <VANT>.

ABSTRACT

Unmanned Aerial Vehicles (UAVs), also known as drones, are widely utilized across various applications owing to their versatility and real-time data acquisition capabilities. However, monitoring the performance of these devices during flight is very important to ensure safe and efficient operation. Research and development of ways to increase their reliability is essential to ensure not only the safety of the equipment, which tends to be expensive, but also the safety of the people who operate it and are involved in its use. This paper proposes the development of an embedded system grounded in Chaos Theory, specifically Chaos Signal Analysis using Maximum Density (SAC-DM), to gather and process vibration data in real-time. The system is designed employing an ESP32 development kit and an Inertial Measurement Unit (IMU), alongside other components, aimed at identifying vibration patterns indicative of potential drone failures, thereby enhancing reliability and operational safety. Additionally, the collected data will be transmitted to a cloud platform for remote analysis and diagnostics.

Key-words: <UAV>, <ESP32>, <SAC-DM>, <Failure Detection>.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API – *Application Programming Interface*

CPU - *Central Process Unit*

ESC - *Eletcronic Speed Controller*

ESP-IDF - *Espressif IoT Development Framework*

GPIO - *General Purpose Input/Output*

HTTP - *Hypertext Transfer Protocol* IDE - *Integrated Development Environment*

IMU - *Intertial Measurement Unit*

IOT - *Internet Of Things*

I2C – *Inter-Integrated Circuit*

JSON – *JavaScript Object Notation*

LCD - *Liquid Crystal Display*

LED - *Light Emitting Diode*

MEMS - *Microelectromechanical Systems*

SAC-DM - *Signal Analysis based on Chaos using Density of Maxima*

SCL – *Serial Clock*

SDA – *Serial Data*

SoC - *System-on-Chip*

SQL – *Structured Query Language*

UART - *Universal Asynchronous Receiver / Transmitter*

VANT - Veículo Aéreo Não Tripulado

Sumário

1 INTRODUÇÃO

Nos últimos anos, os Veículos Aéreos Não Tripulados (VANTs), popularmente conhecidos como drones, têm ganhado destaque em diversas aplicações devido à sua versatilidade e capacidade de coleta de dados em tempo real. Inicialmente pensados para fins militares, hoje em dia são empregados em diversos setores, desde o entretenimento até a agricultura, logística, entre muitos outros. No entanto, o monitoramento da saúde desses dispositivos durante o voo tem sido um grande desafio. Vibrações excessivas podem comprometer não apenas a qualidade dos dados coletados, mas também a integridade estrutural do drone, representando um risco significativo para sua operação e a segurança das pessoas ao redor (GHAZALI; RAHIMAN, 2022). Dito isso, o monitoramento em tempo real desses veículos é essencial para a detecção precoce de falhas mecânicas durante os voos.

Normalmente um piloto de Drone consegue evitar certos tipos de falhas durante um voo, como por exemplo nível baixo de bateria, perda de comunicação e condições de tempo que afetem o veículo. Pelo acompanhamento de dados que são fornecidos pelas controladoras, são problemas facilmente contornáveis (GHAZALI; RAHIMAN, 2022). No entanto, falhas originárias de componentes da estrutura como hélices danificadas, folgas de parafusos ou até frame quebrado, são mais difíceis de identificar. Para que seja possível evitar estes tipos de falhas, a manutenção deve ser feita regularmente pelo piloto, o que não costuma ocorrer com tanta frequência. Vários desses problemas podem ser identificados através de dados de vibração que podem ser coletados durante o voo.

Tendo em vista o propósito de aumentar a confiabilidade dos drones, várias pesquisas são realizadas com o intuito de diagnosticar e detectar falhas, baseadas em dados gerados em tempo real durante um voo. Alguns trabalhos utilizam uma análise baseada em sons, emitidos pelos motores que operam em alta velocidade (ALTINORS et al., 2021). Existe também métodos de detecção baseados na velocidade e consumo de corrente dos motores, providos pelos Controladores Eletrônicos de Velocidade (ESC) (SAIED et al., 2017). Porém, um dos métodos mais eficazes e comuns de se encontrar é o baseado em vibração. Baseados na nos dados captados por um dispositivo de Unidade de Medida Inercial (IMU), esse tipo de método se baseia principalmente na diferença de vibração causada pelo desbalanceamento de hélices nos drones (BONDYRA et al., 2017). Os IMU's possuem acelerômetros que são capazes de fornecer a aceleração nos eixos X, Y e Z. Desta forma, é possível aplicar a técnica que melhor se adeque para processá-los e gerar o diagnóstico desejado.

Uma técnica de processamento de sinais foi desenvolvida para simplificar o tratamento de dados caóticos, como os mencionados anteriormente. Esta técnica é conhecida como Análise de Sinal Baseada no Caos utilizando Densidade de Máximos (SAC-DM). A

partir de um IMU gerando os dados de vibração nos eixos X, Y e Z, o SAC-DM é capaz de realizar a contagem de picos desses sinais dentro de uma determinada amostra, a fim de calcular a densidade de máximos desses picos. Desta forma, o comportamento caótico do sistema pode ser identificado a partir dos resultados da correlação obtida da densidade de máximos (SOUZA et al., 2020).

Neste contexto, este trabalho propõe o desenvolvimento de um sistema embarcado destinado a ser integrado em VANTs, com o objetivo de coletar e processar dados de vibração em tempo real. Utilizando um algoritmo baseado na Teoria do Caos, nesse caso o SAC-DM. O sistema será capaz de identificar padrões de vibração indicativos de possíveis falhas no drone. Além disso, as informações serão encaminhadas para uma plataforma na nuvem, permitindo a realização de análises e diagnósticos remotos. Essa abordagem visa aprimorar a confiabilidade e segurança operacional dos drones, representando um avanço significativo no monitoramento da saúde desses dispositivos.

Encontram-se descritos no próximo capítulo o objetivo geral e objetivos específicos deste trabalho. No capítulo três apresenta-se os conceitos gerais e a revisão da literatura sobre as ferramentas de desenvolvimento, plataformas utilizadas e elementos de software e hardware do sistema embarcado. O capítulo quatro apresenta a metodologia do trabalho, nesse caso como foi desenvolvido e estruturado todo o software e comunicação entre os componentes do sistema. O capítulo cinco apresenta os resultados obtidos neste projeto e o capítulo seis apresenta as discussões, conclusões e as ideias de trabalhos futuros.

2 ESCOPO DO TRABALHO

A seguir serão apresentados as problemáticas abordadas neste trabalho, os objetivos gerais e específicos e uma breve descrição do que será abordado no decorrer deste trabalho.

2.0.1 Objetivo geral

Desenvolver um sistema embarcado qualificado para ser acoplado em um VANT, com capacidade para coletar dados de vibração em tempo real, processar utilizando algoritmo baseado na Teoria do Caos e encaminhar essas informações para uma plataforma na nuvem, para tratamentos posteriores, com o objetivo de realizar a detecção e diagnóstico de falhas em drones.

2.0.2 Objetivos específicos

- Desenvolver uma arquitetura de software escalável para coleta dos dados.
- Implementaçar um componente de código para execução do algoritmo baseado na Teoria do Caos.
- Identificar e configurar plataformas e serviços na nuvem para persistência dos dados coletados.

3 CONCEITOS GERAIS E REVISÃO DA LITERATURA

A seguir são identificados os conceitos necessários para entender o que compõe o sistema embarcado proposto neste trabalho. Desta forma, são apresentados alguns dos elementos essenciais de hardware e software utilizados no desenvolvimento do dispositivo.

3.1 Sistemas Embarcados

Sistemas Embarcados são dispositivos programados para servir uma aplicação específica ao qual foi designado. Dependendo de onde é empregado, vários tipos de dispositivos podem ser considerados sistemas embarcados, desde microcontroladores de 8 bits até celulares. Geralmente esse tipo de sistema atua sob certas circunstâncias como processadores baratos, baixo consumo de energia, CPU com recursos limitados e suporte a periféricos, como sensores e atuadores (WHITE, 2011).

As linguagens de programação mais comuns para sistemas embarcados são o C e C++. Na grande maioria das vezes, as ferramentas de compilação suportam elas, embora também seja possível programar com outras linguagens como Rust e Micropython. A forma como eles são programados, também ajudam a identificá-los como sistemas embarcados, pois utilizam cross compilers. Os cross compilers são ferramentas que rodam no desktop, mas que criam códigos que rodam em dispositivos embarcados (WHITE, 2011). Estas ferramentas normalmente são disponibilizadas pela fabricante do chip ao qual está sendo trabalhado.

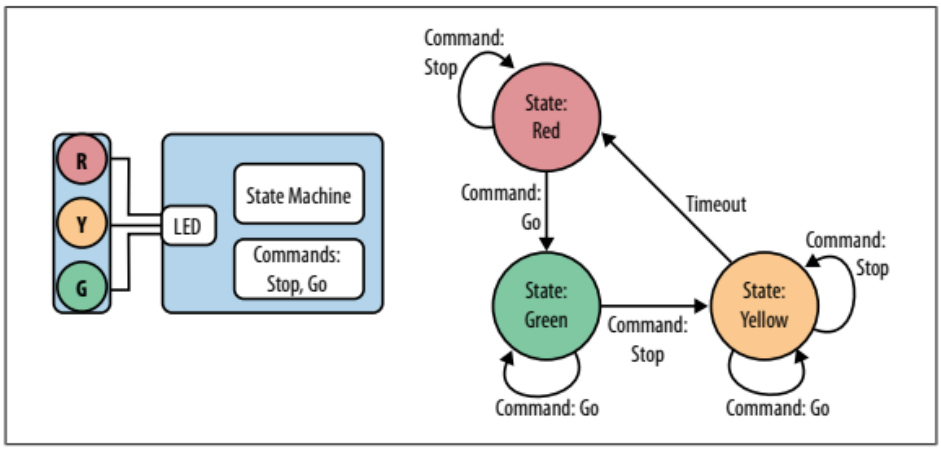
3.1.1 Máquinas de Estados

Uma forma relativamente simples de construir um código complexo de sistemas embarcados é através de máquinas de estados. Elas mantém o sistema organizado, enquanto uma ou mais atividades podem estar acontecendo em paralelo. É um padrão de código que que através de estados, determinados a partir dos eventos do sistema, ajuda a definir qual vai ser o próximo estado, ou seja, quais serão as próximas ações a serem tomadas na lógica implementada (WHITE, 2011).

Na figura 18 é possível ver um exemplo de uma máquina de estados. Nesse sistema, criado para representar a lógica que poderia ser implementada para um sinal de trânsito, existem três estados: Vermelho, Amarelo e Verde. Esses estados possuem um fluxo bem determinado, que vai do vermelho para o verde, do verde para o amarelo e do amarelo de volta para o vermelho. A qualquer momento podem ser enviados comandos ao sistema, que pode se comportar de diferentes formas de acordo com o estado atual. Quando o sinal está vermelho, um comando de *stop*, que faz o sinal parar, não afeta em nada, pois já está

no sinal de parado. Entretanto, o comando de *go* faz o estado mudar de vermelho para verde. No sinal verde, o comando de *go* não influencia no estado, pois o verde já sinaliza isso, mas o de stop faz mudar para amarelo, pois é obrigatório que antes do vermelho o sinal seja amarelo para alertar. No estado amarelo, nenhum comando de *stop* ou *go* o faz mudar, pois o alerta possui um tempo mínimo para ser exibido antes do fluxo parar. Por isso, o estado só é alterado depois de um temporizador.

Figura 1: Diagrama de estados de um sinal de trânsito.



Fonte: (WHITE, 2011)

Assim como em um sistema embarcado comum, o sinal de trânsito possui casos a serem tratados dependendo do que está acontecendo no momento. Esse tratamento não foge da normalidade de implementações com condicionais, loops de repetição, entre outros. No entanto, quando a implementação é baseada em uma máquina de estados, tudo se torna organizado, fácil de manter e facilita a implementação de melhorias, dando espaço para o sistema crescer.

3.1.1.1 Implementação de uma máquina de estados

A lógica por trás de uma máquina de estados remete a uma grande verificação de qual o estado atual do sistema, para que o haja uma reação de acordo com os eventos que ocorrem. Nesse caso, podemos ver no código de exemplo 1 abaixo que, para cada mudança de estado da cor do sinal de trânsito, existe uma série de ações a serem realizadas, ao mesmo tempo que uma mudança de estado é verificada sempre dentro do loop. Cada estado é responsável por alterar as variáveis do sistema para que ele funcione como deveria, ou seja, o desenvolvedor é responsável pela coerência entre as partes do funcionamento.

Código Fonte 1: Exemplo de implementação de máquina de estados

```
while (1) {
        look for event
        switch (state) {
                case (green light):
                        if (event is stop command)
                                turn off green light
                                turn on yellow light
                                set state to yellow light
                                start timer
                        break;
                case (yellow light):
                        if (event is timeout)
                                turn off yellow light
                                turn on red light
                                set state to red light
                        break;
                case (red light):
                        if (event is go command)
                                turn off red light
                                turn on green light
                                set state to green light
                        break;
                default (unhandled state)
                        error!
    }
}
```

Fonte: (WHITE, 2011)

3.2 Plataforma como Serviço

O modelo de computação na nuvem chamado de Plataforma como serviço, ou PaaS (do inglês Platform-as-a-Service), é uma ótima opção quando se precisa desenvolver um projeto ao qual o foco está no resultado entregue pelo produto, e não no ambiente de desenvolvimento e implementação da infraestrutura de rede/nuvem para o projeto. Esse tipo de modelo é oferecido pelos maiores líderes no mercado da tecnologia, como Amazon Web Services (AWS), Google Cloud, IBM Cloud, Microsoft Azure, entre outros.

Tem como objetivo fornecer diversos serviços como banco de dados, sistema operacional pronto para uso, gerenciamento de aplicativos, servidores, entre outras ferramentas (IBM, 2023). Cada empresa possui suas particularidades, mas no geral, é possível utilizar tudo isso sem custo, para testar produtos e soluções, até que quando estiver pronto para ser lançado, a paltaforma seja paga conforme o uso distribuido entre os produtos.

Este modelo traz uma série de vantagens no desenvolvimento de projetos. Lançamento mais rápido no mercado, já que o hardware e software já estão prontos e podem ser configurados com apenas alguns cliques. Mais velocidade no desenvolvimento e menos risco em teste de funcionalidades, além de prover uma grande escalabilidade. No geral, aumenta a flexibilidade no desenvolvimento e diminui o custo de produção do projeto (IBM, 2023).

3.2.1 Supabase

O Supabase é um PaaS open source, criado para ser uma alternativa ao Firebase, que por sua vez também é um PaaS pertencente ao Google, comumente utilizado em projetos de sistemas embarcados conectados na nuvem. Por ser open source, possui uma comunidade grande e ativa, portanto possui suporte a diversas plataformas de desenvolvimento e está em constante crescimento. Embora seja uma alternativa ao Firebase, hoje em dia oferece diversos diferenciais com soluções próprias e inovadoras. Algumas das funcionalidades disponibilizadas no Supabase são: Banco de dados Relacional Postgres, Extensões de banco de dados, Sistema de Autenticação, API's geradas automaticamente para facilitar o uso das aplicações, Armazenamento de Arquivos, Bibliotecas de Clients para uso da plataforma, entre outros (SUPABASE, 2023).

3.2.1.1 Banco de Dados Postgres

O PostgreSQL é um banco de dados relacional open source que possui um sistema capaz de guardar com segurança e eficiência os mais variados tipos de dados. Criado em 1986, é desenvolvido e mantido há mais de 35 anos, com uma comunidade ativa e recebendo constantes melhorias até os dias atuais. Possui uma boa reputação pela boa confiabilidade, arquitetura, integridade de dados, robustez e funcionalidades disponíveis. Além de grátis, possui diversas extensões, permitindo que o usuário consiga implementar diversos sistemas de formas diferentes e em linguagens de programação diferentes, sem afetar a integração com a base de dados (POSTGRESQL, 2024).

Todo projeto criado no Supabase possui uma base de dados PostgreSQL dedicada a ele. Sendo uma das base de dados mais confiáveis e escaláveis que existe, foram criados serviços na nuvem do Supabase para facilitar seu uso por sistemas e aplicações. Desta forma, sempre que uma base é definida, ao mesmo tempo também é criada automatica-

mente uma API que permite acessar, adicionar e modificar os dados dentro desta base (SUPABASE, 2023).

3.3 Tecnologias de Internet das Coisas

O conceito de Internet das Coisas traz um paradigma multidisciplinar, ao qual os sistemas embarcados estão inseridos como dispositivos de comunicação conectados na internet. Esses dispositivos são responsáveis por criar novos serviços e aplicações nas áreas da saúde, casas e cidades inteligentes, indústria, entre vários outros. Tudo isso é obtido a partir da interação entre o dispositivo e o mundo real, através de sensores e atuadores, que desempanham um papel fundamental no funcionamento dos sistemas embarcados (SAMIE et al., 2016).

3.3.1 Sensores e Atuadores

A tomada de decisão e funcionamento de sistemas embarcados dependem diretamente do papel entre sensores e atuadores. Os atuadores tomam ações baseado nas entradas do sistema, informações capturadas pelos sensores, no ambiente em que atua. Enquanto os sensores são lidos, os atuadores são escritos, e o que acontece entre eles é o processamento e lógica do algoritmo baseado no objetivo do projeto (MOHAMED, 2019). Este trabalho tem como principais sensores e atuadores o Botão, LED e uma Unidade de Medida Inercial.

3.3.1.1 IMU

Um Sistema Micro-Eletro-Mecânico (MEMS) baseado em Unidade de Medição Incercial (IMU) são muito utilizados em aplicações que se preocupam com monitoramento, diagnóstico e/ou controle, transporte e navegação de sistemas. Os dados fornecidos por esses dispotivivos, que geralmente são chamados de IMU, são obtidos através de um grupo de sensores como acelerômetros, giroscópios e magnetômetros. Geralmente este tipo de dispositivo é utilizado para fornecer informações para filtros e algoritmos que precisam analisar os dados da situação do sistema embarcado em tempo real e diagnosticar algum tipo de falha ou comportamento (CAPRIGLIONE et al., 2021).

Nas áreas de atuação relacionadas a Drones, que envolve tanto os Veículos Aéreos Não Tripulados (VANTs) e Aeronaves Remotamente Pilotadas (RPAs), é obrigatório que esses sistemas possuam um sistema confiável, pois uma falha no meio da operação pode causar sérios riscos. Por isso, IMU's são extremamente importantes para o monitoramento desses sistemas, pois são capazes de medir alguns parâmetros físicos como a aceleração, utilizando um acelerômetro triaxial, a taxa de mudança angular, utilizando um giroscópio triaxial e em alguns casos, quando o IMU tem disponível, também o campo magnético

através de um magnetômetro triaxial. Eles dominam o mercado por serem baratos, práticos, possuirem um baixo consumo de energia e apresentar uma boa acurácia, além de serem chips de fácil integração em projetos (CAPRIGLIONE et al., 2021).

3.4 SAC-DM

A técnica de Análise de Sinal Baseada no Caos usando Densidade de Máximos, também conhecida como SAC-DM, é utilizada para analisar os sinais de vibração de um drone. Utilizando a contagem de picos dentro de uma determinada amostra de dados coletados, é possível determinar a média da densidade dos máximos dessas amostras. Desta forma, é possível obter uma correlação entre a densidade de máximos e o comportamento caótico do sistema. Os dados coletados, mencionados anteriormente, no contexto de vibração de um drone, são os de acelerômetro, obtidos através de um IMU, nos eixos X, Y e Z (SOUZA et al., 2021).

No algoritmo 1 podemos ver a lógica de funcionamento da técnica, que utiliza os resultados de leitura do acelerômetro para o cálculo dos máximos.

Algoritmo 1: SAC-DM

Function `SAC-DM`($sinal_x, sinal_y, sinal_z, tamanhoAmostra$)**:**

$taxaLimite \longleftarrow 1.10$;
$n \longleftarrow sampleSize$;
$picos_x \longleftarrow 0$;
$picos_y \longleftarrow 0$;
$picos_z \longleftarrow 0$;
$leituras \longleftarrow 0$;
while *lerSinais* **do**
  $acc_x[0] \longleftarrow acc_x[1]$;
  $acc_x[1] \longleftarrow acc_x[2]$;
  $acc_x[2] \longleftarrow sinal_x$;
  $acc_y[0] \longleftarrow acc_y[1]$;
  $acc_y[1] \longleftarrow acc_y[2]$;
  $acc_y[2] \longleftarrow sinal_y$;
  $acc_z[0] \longleftarrow acc_z[1]$;
  $acc_z[1] \longleftarrow acc_z[2]$;
  $acc_z[2] \longleftarrow sinal_z$;
  $leituras \longleftarrow leituras + 1$;
  if $leituras > 2$ **then**
    if $acc_x[1] > acc_x[0]$ & $acc_x[1] > acc_x[2]$ **then**
      $picos_x \longleftarrow picos_x + 1$
    end
    if $acc_y[1] > acc_y[0]$ & $acc_y[1] > acc_y[2]$ **then**
      $picos_y \longleftarrow picos_y + 1$
    end
    if $acc_z[1] > acc_z[0]$ & $acc_z[1] > acc_z[2]$ **then**
      $picos_z \longleftarrow picos_z + 1$
    end
  end
  if $leituras == n$ **then**
    $rho_x \longleftarrow picos_x/tamanhoAmostra$;
    $rho_y \longleftarrow picos_y/tamanhoAmostra$;
    $rho_z \longleftarrow picos_z/tamanhoAmostra$;
    $leituras \longleftarrow 0$;
    $picos_x \longleftarrow 0$;
    $picos_y \longleftarrow 0$;
    $picos_z \longleftarrow 0$;
  end
end
return $SAC - DM$
End Function

Neste algoritmo é possível observar algumas variáveis importantes para o funcionamento da solução. A princípio são esperadas as três entradas mencionadas, valores de aceleração nos eixos X, Y e Z, providas pelo sensor, além do tamanho da amostra (quantidade desses dados que será coletada em um intervalo de tempo). Por padrão este intervalo de tempo é fixo em um segundo, ou seja, o tamanho da amostra determina a frequência de coleta de dados dentro deste intervalo. Com a entrada e tamanho de amostra definidos, dentro do intervalo de cada segundo que se passa, o sistema coleta os valores de aceleração e a cada três coletas, continuamente, é verificado se o valor central é maior que o anterior e o seguinte a ele. Se for maior, isso caracteriza um pico. A partir disto, no final do intervalo de coleta da amostra, a densidade de picos daquele intervalo é calculada e armazenada no que chamamos de *rho X*, *rho Y* e *rho Z*. Essas resultantes podem ser utilizadas para a análise do comportamento caótico.

3.4.1 Sistema de Detecção de falhas de Drones utilizando Densidade de Máximos

A detecção de falhas baseado na densidade de máximos é uma técnica testada e validada em Drones de pequeno porte e bancada de testes. O desenvolvimento do protótipo utiliza um sensor de baixo custo (MPU6886) e um microcontrolador simples com conexão a Wi-Fi (ESP8266). Para execução dos testes em um Drone, alguns pontos precisam estar bem definidos, como a frequência de coleta de dados, tamanho da amostra e qual tipo de falha se almeja detectar com o sistema embarcado. Portanto, a indução a falhas específicas é necessária para a comparação com os resultados esperados, para que possam ser analisados os efeitos da densidade dos picos em cada situação (SOUZA et al., 2021).

3.5 Meios e Protocolos de Comunicação

Uma das principais premissas por trás da Internet das Coisas é ter dispositivos embarcados se comunicando com outros sistemas. A ideia dessa comunicação é trafegar dados através de uma rede, utilizando protocolos de comunicação (NIKOLOV, 2018). Os protocolos de comunicação definem as regras que os dispositivos devem seguir para concluir a transferência de dados entre as partes, seja através de uma rede com fio ou sem fio. A utilização dos meios de comunicação e protocolos corretos para cada tipo de aplicação é importante para se trabalhar com os dados do sistema embarcado da melhor forma possível.

3.5.1 Comunicação Serial

A comunicação serial consiste na conversação entre portas seriais para transmitir dados entre si, através de cabos e/ou fios, utilizando especificamente um protocolo de comunicação assíncrono para a transmissão dos bits. As portas transferem esses dados normalmente através de USB, bastante conhecido desde sua criação. No entanto, também é encontrado de outras formas, como em portas RS-232 em computadores ou através de pinos, como vemos normalmente em sistemas embarcados. Os sistemas embarcados costumam possuir uma ou mais portas seriais, para se comunicar com computadores, periféricos ou até outros sistemas embarcados (AXELSON, 2007). A representação virtual das portas seriais podem ser facilmente identificadas no Windows como as portas COM, também no Linux como as portas /dev/tty ou /dev/ACM.

O uso da comunicação serial oferece diversas vantagens, pois como são capazes de transmitir praticamente qualquer tipo de dados, diversos tipos de aplicações podem se beneficiar disso, por exemplo coletando informações de diversos tipos de sensores ou enviando comandos para diferentes tipos de atuadores. Além disso, é barato e possui amplo suporte não só entre aparelhos, mas também de sistemas operacionais. Dispositivos que possuem portas assíncronas normalmente possuem um periférico de hardware chamado Universal Asynchronous Transmitter/Receiver (UART), que lida com diversas necessidades e regras da comunicação (AXELSON, 2007). Aplicações de sistemas embarcados utilizam também com frequência para testes e debug de aplicações, visualizando todo o tráfego de dados e logs de execução em tempo real, em um monitor serial que exibe tudo na tela, em tempo real.

3.5.1.1 UART

O UART é uma funcionalidade provida por um hardware capaz de lidar com a comunicação serial de maneira assíncrona. Durante a troca de dados, encarrega-se de todas as regras que os dispositivos em questões estabelecem para se entenderem, como taxa de transferência de bits por segundo, tamanho dos frames de dados, paridade, bits de controle, entre outros. Uma vez que os dois dispositivos que vão se comunicar configuram esses parâmetros, eles são capazes de enviar e/ou receber os dados um para o outro. O chip da ESP32 possui três controladores de UART, como pode ser visto na figura 4 indicado na cor cinza, também chamados de porta UART. Além disso, o ESP-IDF também oferece as próprias bilbiotecas para cuidar de todo o tratamento de software e hardware para o desenvolvedor (ESP-IDF, 2024).

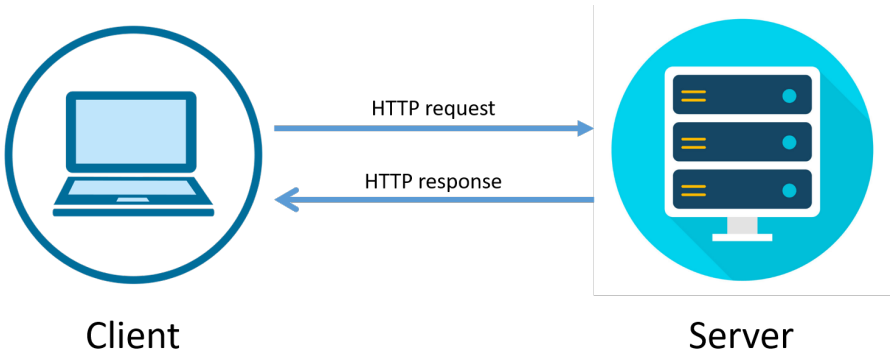
3.5.2 Wi-Fi

Wi-Fi é um tipo de tecnologia de rede sem fio que possibilita a conexão à Internet de diversos tipos de dispositivos, comumente encontrado em placas de Internet das Coisas. Essa tecnologia possibilita a comunicação e troca de informações entre esses dispositivos, formando uma rede. A conexão com a Internet é estabelecida através de um roteador sem fio, onde, ao acessar o Wi-Fi, os dispositivos compatíveis se conectam a este roteador, criando uma interface com a Internet (CISCO, 2020).

3.5.2.1 Protocolo HTTP

HTTP, ou *Hypertext Transfer Protocol*, é um protocolo essencial para a transferência de dados na Internet, formando a base para a conexão entre clientes e servidores. Funcionando no modelo cliente-servidor, o HTTP torna possível a comunicação entre dispositivos, permitindo que usuários acessem informações online. Desenvolvido por Tim Berners-Lee e sua equipe no CERN entre 1989 e 1991, o HTTP evoluiu ao longo do tempo, com versões subsequentes como HTTP/1.0, HTTP/1.1, HTTP/2 e HTTP/3. Sua importância reside na facilitação da solicitação e resposta de dados, formando a estrutura essencial para a navegação na web (SANTANA, 2023). Como pode ser visto na figura 2, qualquer dispositivo conectado na internet que foi configurado para ser utilizadoc como cliente, pode enviar e receber dados através de requisições a algum servidor, seja ele local ou na nuvem. Para cada requisição, há uma resposta, seja de sucesso ou erro, confirmando o envio ou recebimento dos dados.

Figura 2: Modelo cliente-servidor do protocolo HTTP



Fonte: (BYTESOFGIGABYTES, 2020)

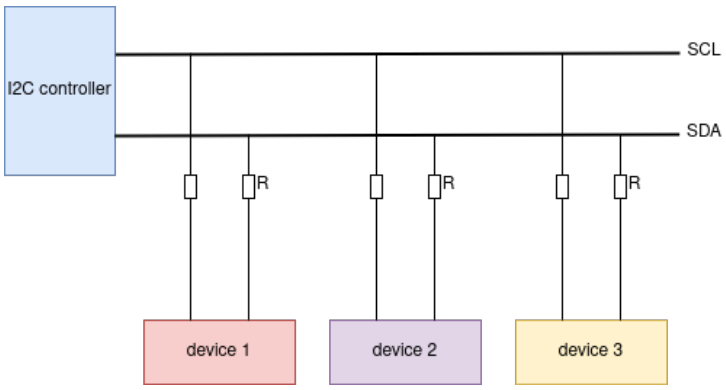
3.5.3 I2C

O I2C, ou *Inter-Integrated Circuit*, é um protocolo que define o funcionamento de um barramento de comunicação que utiliza dois fios. Inventado pela Phillips no início dos

anos 90, é um protocolo muito utilizado para conectar periféricos de baixa velocidade a placas-mãe, microcontroladores e outros dispositivos. Para o devido funcionamento, tanto o dispositivo de controle (*Master*) quanto o periférico (*Slave*) devem ter suporte ao I2C através do hardware. Os dois fios são representados como SDA e SCL (CAMARA, 2013).

O fio SCL é responsável pelo sinal de *clock* do barramento, enquanto o SDA pela transmissão de dados. A figura 3 apresenta como o protocolo I2C é utilizado em um sistema. Através de uma lógica bem definida pelo protocolo, o sistema com o I2C possui capacidade para multiplos *Masters* e/ou *Slaves*. Nesse caso, o *Master* (I2C controller) pode ser visto como um sistema embarcado qualquer. Ele é capaz de controlar um ou mais dispositivos *Slave*, enviando comandos e recebendo dados providos por eles (ESP-IDF, 2024).

Figura 3: Esquema de fios entre *Master* e *Slaves* do protocolo I2C



Fonte: (ESP-IDF, 2024)

4 METODOLOGIA

Neste capítulo e descrito todo o processo envolvido com o desenvolvimento do sistema embarcado. Como mencionado no capítulo inicial, o objetivo é implementar um sistema que pode ser acoplado a um drone, capaz de coletar dados de vibração em tempo real, processá-los em um algoritmo baseado na teoria do caos e encaminhá-los para uma plataforma que permita o acesso para posteriores tratamentos.

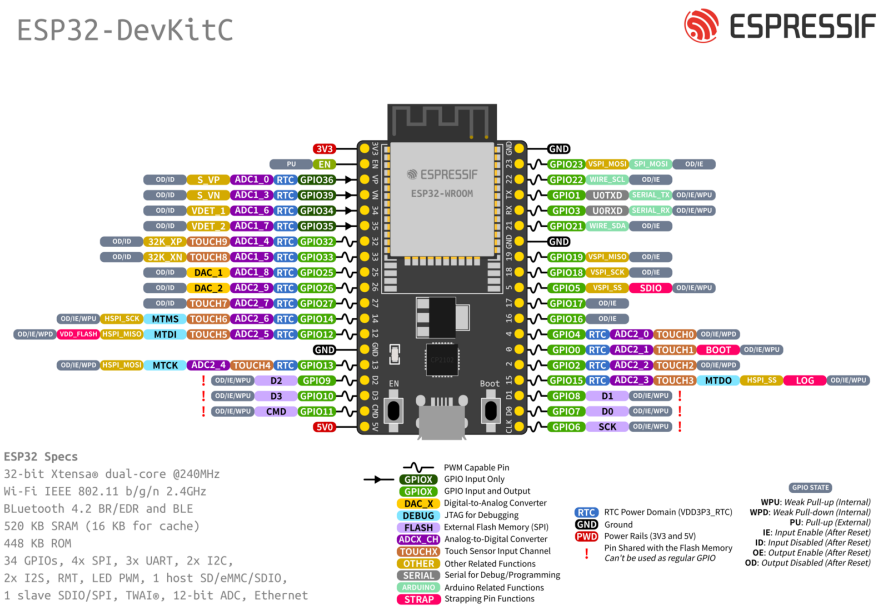
O sistema embarcado pode ser descrita através da arquitetura do projeto, máquina de estados e componentes desenvolvidos a partir das API's fornecidas pelo ESP-IDF. Com uma máquina de estados bem definida para o passo a passo da execução, cada estado é responsável pela verificação se a execução do código está ocorrendo como esperado e pela definição do próximo estado. O sistema não é completamente autônomo, então também realiza o tratamento de eventos, como clique de botão para tomar certas ações, como início e parada de coleta de dados. Há também o feedback visual, com um led indicador que fornece informações em campo se o sistema está funcionando como esperado ou não.

4.1 ESP32

O ESP32 é um *SoC* (System-on-Chip), normalmente encontrado em placas de desenvolvimento, aplicadas em projetos que exige conectividade sem fio entre o sistema embarcado e outros dispositivos. Oferece algumas características como design robusto, capaz de atuar em cenários que vão desde residenciais até na indústria, consumo de energia ultra baixo, boa integração entre o chip e diversos tipos de antenas, assim como na grande maioria do portfolio disponibiliza no mesmo chip Wi-Fi e Bluetooth (DATASHEET, 2023). Algumas das características de hardware mais comuns da ESP32 pode ser visto na figura 4.

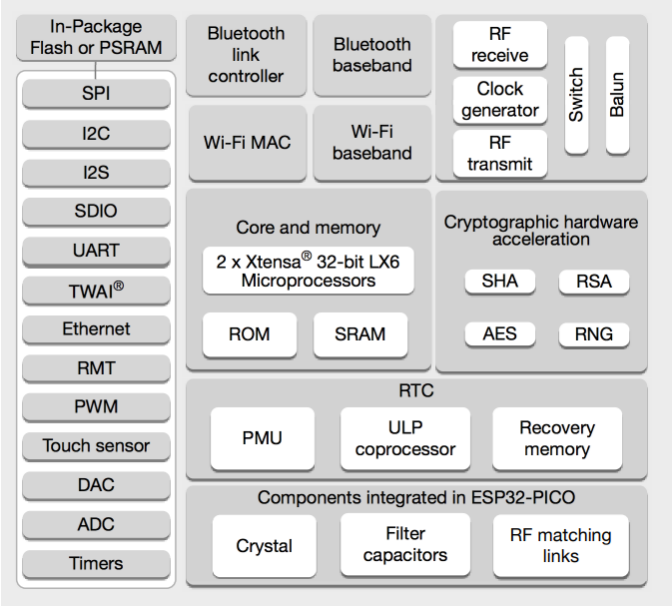
A Espressif, empresa responsável pelo desenvolvimento deste chip, disponibiliza uma série de tipos de famílias com diferentes funcionalidades. O ESP32 pode ser encontrado no mercado em formato de chip, módulo ou placa de desenvolvimento (ESP-IDF, 2024). Isso influencia diretamente na variedade de placas e possibilidades de aplicações, visto que diversas outras empresas conseguem desenvolver suas próprias placas e kits, oferecendo ao usuário final uma gama de possíveis cenários para se trabalhar. Além disso, essa disponibilidade impacta diretamente no preço, que acaba sendo bem abaixo do valor encontrado por outras marcas. No entanto, por oferecer uma arquitetura própria do CPU, o desenvolvedor deve usar no desenvolvimento algumas ferramentas fornecidas apenas pela fabricante. Na figura 5 é possível ver alguns dos periféricos e funcionalidades utilizados na maior parte dos chips ESP2 do mercado.

Figura 4: Layout de pinos do Kit de Desenvolvimento com ESP2 (DevKit).



Fonte: (ESP-IDF, 2024)

Figura 5: Diagrama de Blocos da série de SoC's ESP32.



Fonte: (DATASHEET, 2023)

4.1.1 M5STICK-C PLUS

Desenvolvido pela empresa M5Stack, o M5StickC Plus é a versão melhorada de um kit de desenvolvimento lançado anteriormente, o M5StickC. Este kit possui uma tela maior e como seu antecessor, também utiliza o *SoC* ESP32-PICO-D4. Diferente do ESP32 comum, este *SoC* é menor e normalmente utilizado em projetos que utilizam os recursos

oferecidos pelo chip, mas com um design bastante reduzido. Esse é o caso deste kit, que oferece diversas funcionalidades, como pode ser visto na tabela 1, em um encapsulamento extremamente restrito. A figura 6 apresenta uma foto de como é a aparência do kit.

Figura 6: Kit de Desenvolvimento IoT M5StickC Plus.

Fonte: Autoria própria

Tabela 1: Principais recursos disponíveis no M5StickC Plus.

Recurso	**Característica**
ESP32-PICO-D4	240MHz dual core, Wi-Fi/Bluetooth
Memória Flash	4MB
Portas	1xUSB Tipo C, 1xGROVE(I2C/UART)
Tela LCD	1,14"135*240px, ST7789v
Botão	2x Botões
LED	1xLED vermelho
Unidade de Medição Inercial (IMU)	MPU6886
Buzzer	1xBuzzer
Infra Vermelho	1xEmissor Infra Vermelho
Microfone	SPM1423
Temporizador de Tempo Real(RTC)	BM8563
Unidade de Gerenciamento de Energia (PMU)	AXP192
Bateria	120 mAh @ 3.7V
Antena	Antena 3D 2.4G
Pinos de propósito geral (GPIO)	G0, G25/G36, G26, G32, G33
Temperatura de Operação	0°C até 60°C
Peso	15g
Dimensões	48.2 x 25.5 x 13.7mm
Material do encapsulamento	Policarbonato

Fonte: (M5STACK, 2024)

O principal objetivo do kit é oferecer uma solução que seja portátil, fácil de usar e que possua recursos de Internet das Coisas que facilitem o desenvolvimento de projetos. Com ele é possível criar protótipos de diversas ideias sem a necessidade de montagem

de diferentes circuitos, reduzindo o tempo de desenvolvimento e evitando algumas dificuldades encontradas no processo de montagem com vários periféricos. Além de oferecer diversos recursos de hardware, também possui conectores que podem ser facilmente expansíveis para vários periféricos fabricados e fornecidos pela M5Stack, que são feitos para serem plugados e utilizados, de maneira simples.

4.2 ESP-IDF

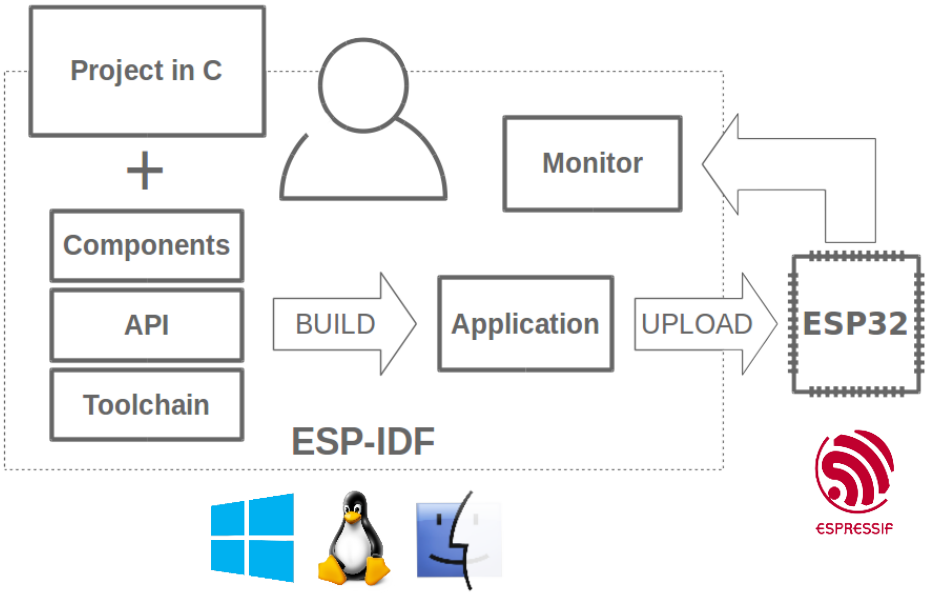
O ESP-IDF é o Framework de Desenvolvimento de IoT oficial da Espressif, que dá suporte as famílias de SoC's ESP32, ESP32-S, ESP32-C e ESP32-H. O framework disponibiliza um kit de desenvolvimento de software suficiente para realizar qualquer tipo de aplicação IoT em qualquer uma destas plataformas, usando as linguagens de programação C e C++. Também provê para o desenvolvedor uma toolchain, API's, componentes prontos para uso e métodos bem estabelecidos de criação de software para aplicações de sistemas embarcados, além de ser suportado em Windows, Linux e macOS. A figura 7 apresenta um resumo do fluxo de desenvolvimento de uma aplicação utilizando o ESP-IDF (DOCS, 2024).

Embora seja fornecido por uma empresa privada, o framework é open-source, disponível no github sob a licença Apache 2.0, que permite usar, modificar e distribuir o código disponível. Se baseia em um processo bem estabelecido de versionamento e desenvolvimento de novas funcionalidades, com um processo de qualidade rigoroso e garantia de que os usuários do framework receberão suporte e correções de problemas para as suas aplicações (ESP-IDF, 2024).

4.2.1 Ferramentas de Software

A fim de desenvolver software para ESP32, o ESP-IDF oferece algumas formas de programar. As principais são a própria IDE para desenvolvimento, baseada em Eclipse, e uma extensão para o editor de texto Visual Studio Code. Fornece também diversos *scripts* através de linhas de comando de terminal que auxiliam em todas as etapas de desenvolvimento e execução do projeto. Dentre esses *scripts*, alguns são bem úteis, ajudando o usuário por exemplo a buildar e compilar o projeto, gerar documentação automática, formatar o código para os estilos de programação padrão do framework, analisar dados de consumo de memória, entre outros. Também segue padrões de interface de programação, que garantem a segurança entre threads na execução do *software*, e de desenvolvimento nas API's das bibliotecas (ESP-IDF, 2024).

Figura 7: API.



Fonte: (DOCS, 2024)

4.2.2 API

Para praticamente todos os recursos existentes na ESP32, mostrados na figura 5, existe uma API ou mais. A API permite utilizar todos os recursos, seja relacionado a periféricos, protocolos de comunicação, componentes integrados ao chip, conectividade (Wi-Fi/Bluetooth), processador, entre outros. Grande parte do que é oferecido pelo ESP-IDF pode ser encontrado através destas API's (ESP-IDF, 2024).

4.2.3 Bibliotecas e Frameworks

Dentro do ecossistema do ESP-IDF, também existem outras bibliotecas e frameworks de desenvolvimento de projetos. Se um produto a ser desenvolvido com o ESP32, por exemplo, tiver o objetivo de ser integrado a um aparelho de som, dentro do ESP-IDF é possível encontrar o framework de desenvolvimento de áudio (ESP-ADF). Dentre as diversas opções para desenvolvimento de produtos, dois são destacados aqui: *ESP-IoT-Solution* e ESP-Arduino (ESP-IDF, 2024).

O *ESP-IoT-Solution* fornece diversos exemplos de aplicações e bibliotecas para aumentar a velocidade e eficiência do desenvolvimento de produtos que utilizam componentes comuns. Nesse caso, se um produto irá utilizar um botão, uma tela LCD e um sensor comum do mercado, o desenvolvedor não precisa criar do zero uma biblioteca para cada um destes componentes. O ESP-IoT-Solution fornece as bibliotecas com os drivers, implementações utilizando as API's do ESP-IDF e todo o necessário para que o foco seja apenas na funcionalidade e comportamento específico do produto (SOLUTION, 2024).

O ESP-Arduino é o mais conhecido, pois a grande maioria dos desenvolvedores iniciantes de Sistemas Embarcados que migram do Arduino para o ESP32, o utilizam para aprender e desenvolver as aplicações iniciais em seus projetos. Este conjunto de bibliotecas é compatível com a IDE do Arduino e oferece opções de implementações para os projetos com ESP32 com comandos que são geralmente iguais aos do Arduino. Como todos os outros frameworks e bibliotecas, utiliza as API's do ESP-IDF debaixo dos panos para realizar todas as operações. Embora essa enorme facilidade seja atraente do ponto de vista de desenvolvimento, ela traz uma série de desvantagens, pois a simplificação dos comandos e implementações faz com que o usuário perca a capacidade de configurar, personalizar e criar um produto com características únicas utilizando o ESP32. É uma solução mantida principalmente pela comunidade, que vista atrair mais usuários de ESP32 sem que precisem aprender do zero o ESP-IDF, que inicialmente possui uma curva de aprendizado que desafia os desenvolvedores menos experientes (ESP32, 2023).

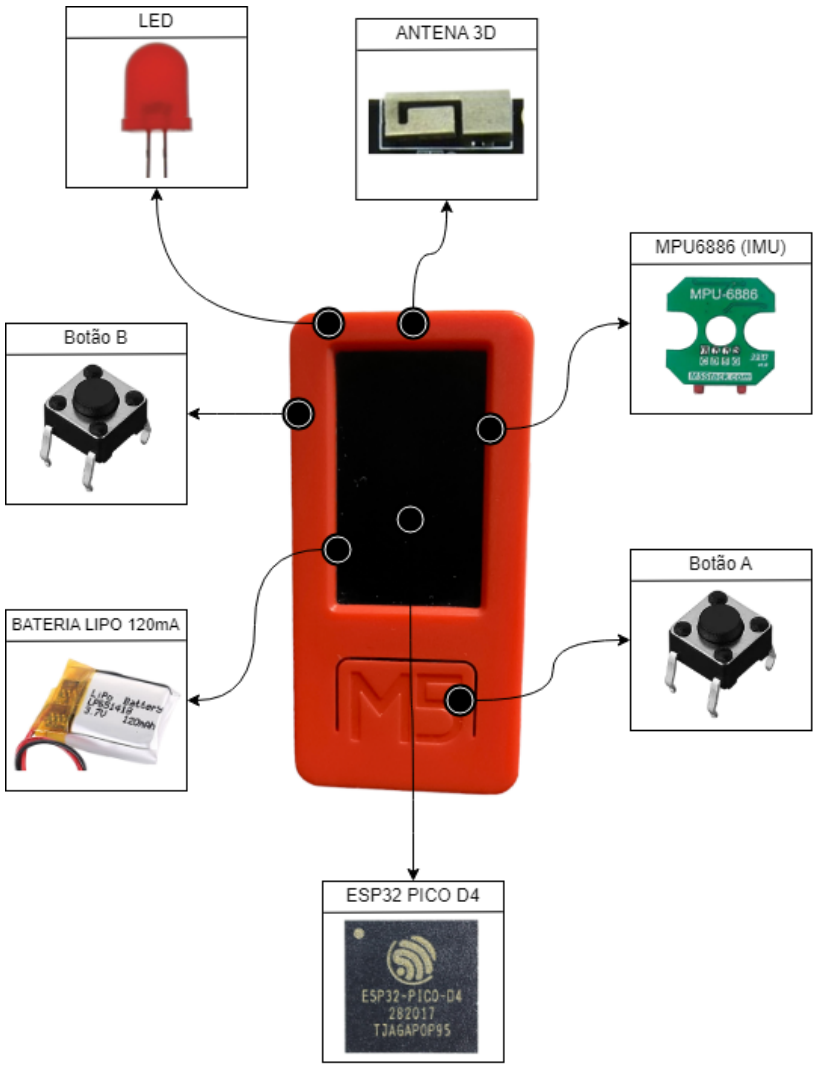
4.2.4 Componentes

Os componentes são importantes na estrutura de projetos utilizando o ESP-IDF. O framework, além de fornecer API's de hardware e sistema para o desenvolvimento de código, disponibiliza também uma variedade de componentes, que a princípio funcionam como bibliotecas. Estes componentes podem usar auxiliar na implementação das API's no código, mas também podem oferecer sub conjuntos de aplicações, como as soluções mencionadas no tópico anterior. No entanto, componentes são bem mais importantes que apenas ajudar no desenvolvimento de software. Os componentes possuem estruturas de arquivos bem definidas, que são utilizadas pelos scripts de build e compilação do framework para mapear todas as dependências do ESP32. Com isso, o sistema de build do software consegue otimizar o tempo de compilação. Além disso, existe um repositório de componentes criados pela Espressif e pela própria comunidade de desenvolvedores, conhecido como *IDF Component Registry*, que funciona como mais uma forma de facilitar o desenvolvimento de software e distribuição de soluções.

4.3 Elementos do Sistema Embarcado

A figura 8 mostra os principais elementos do kit M5StickC-Plus utilizados no projeto. Cada uma dessas partes possui um propósito bem definido e desempenha um papel fundamental para o seu pleno funcionamento.

Figura 8: Principais elementos do kit utilizados no projeto.



Fonte: Autoria própria

Na tabela 2 são identificados em quais pinos os botões e LED estão conectados internamente. São duas partes extremamente importantes para a interação com o usuário do sistema embarcado. Como um dispositivo de saída, o LED tem a responsabilidade de indicar o que está acontecendo com o sistema no momento do uso. Embora seja simples, o feedback fornecido por ele é indispensável em qualquer aparelho, por isso é tão comum e encontramos LED's em praticamente 100% dos eletroeletrônicos em qualquer lugar do mundo. Além disso, como um dispositivo de entrada de dados, os botões também interagem com o usuário para o acionamento das funcionalidades previstas pelo projeto. Assim como o LED, também são muito famosos e são responsáveis por controlar o modo de operação do sistema. Com isso, é possível iniciar e parar a coleta de dados a qualquer momento, influenciando não só na economia de bateria, mas também no controle de dados enviados até a nuvem. Posteriormente será descrito com mais detalhes o comportamento do LED e botão para cada situação do funcionamento do projeto.

Tabela 2: GPIO dos botões e LED.

ESP32	GPIO
LED Vermelho	GPIO10
Botão A	GPIO37
Botão B	GPIO39

Fonte: (M5STACK, 2024)

O MPU6886 é um IMU que pode ser utilizado através de mais de um protocolo de comunicação, mas como podemos ver na 3, ele é mapeado internamente dentro do kit nos pinos referentes ao protocolo I2C. No modo de operação utilizando o I2C, o IMU já dispõe de uma capacidade suficientemente acima do mínimo necessário para a coleta de dados do projeto. Sua operação com I2C provê uma frequência de coleta de 400kHz, enquanto a de coleta do trabalho é de apenas 500Hz. Embora possa oferecer dados de acelerômetro e giroscópio, ambos nos eixos X, Y e Z, são coletados apenas os dados do acelerômetros, necessários para os cálculos referentes a vibração de onde o sistema estará acoplado. A posição inicial escolhida para acoplar o sistema no objeto de estudo influencia nos valores de acelerômetro coletados. Por isso deve-se utilizar sempre a mesma posição e local para acoplar o kit a fim de obter sempre os dados mais próximos e precisos baseados no mesmo referencial.

Tabela 3: GPIO da interface I2C.

ESP32	SCL	SDA
IMU (MPU6886)	GPIO22	GPIO21

Fonte: (M5STACK, 2024)

A bateria fornecida pelo kit parece pequena, mas quando é levado em consideração que o chip utilizado possui um consumo extremamente baixo, ela acaba se tornando bastante eficiente para alimentar tudo que precisa para o funcionamento do sistema, sem muitas dificuldades. Além disso, internamente também existe um circuito integrado responsável pelo gerenciamento de energia e carregamento da bateria, com mais de uma interface. Desta forma, a bateria é facilmente carregada quando o kit é conectado em algum lugar através da porta USB ou quando os GPIO externos referentes a bateria são conectados, possibilitando a extensão do tempo de funcionamento durante o voo, caso o drone possua fios para alimentação externa. Por fim, a anetena 3D é apenas um recurso para aumentar a eficiência de transmissão e recepão do hardware responsável pela conexão wireless. Esse formato de antena visto na imagem é simples e capaz de melhorar significativamente o sinal, se comparado ás antenas comumente encontrada em outros kits

de desenvolvimento, que costumam ser integradas a placa de circuito.

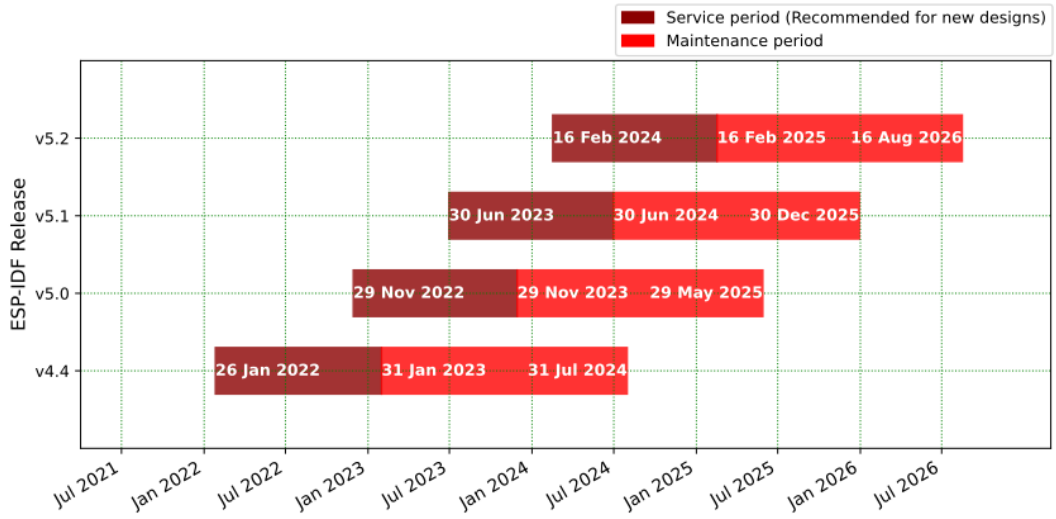
4.4 Programação do Sistema Embarcado

Nesta seção serão apresentados todos os recursos utilizados e criados a partir do ESP-IDF.

4.4.1 Versionamento do Framework

No ESP-IDF encontra-se todos os recursos e ferramentas de desenvolvimento de software necessários para desenvolver qualquer tipo de aplicação com um ESP32. Como é possível observar na figura 9, existem algumas versões disponíveis para o desenvolvimento. Embora qualquer versão possa ser utilizada em projetos com EPS32, até outras além dos que são mostrados, para a execução do que foi implementado neste trabalho, é necessário o uso das versões a partir da 5. Entre a 4.X e 5.X houveram diversas mudanças drásticas em muitas APIs, melhorias na organização do código e alterações na forma como os projetos pode ser estruturados. Além disso algumas ferramentas ficaram obsoletas.

Figura 9: Plano de lançamento das versões do ESP-IDF.



Fonte: (DOCS, 2024)

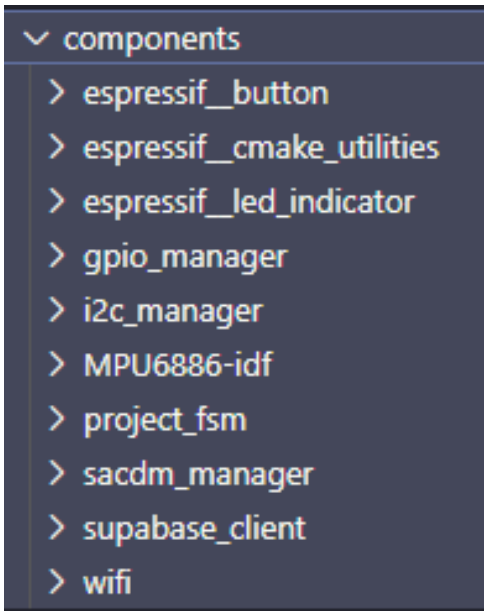
Seguindo o recomendado, para desenvolvimento de novos projetos é indicado a utilização da última versão estável, que atualmente é a 5.1, sendo esta a exata versão responsável por validar todos os testes e resultados apresentados aqui.

4.4.2 Componentes do projeto

Os componentes são elementos essenciais para qualquer projeto que utilize o ESP-IDF. A princípio podem ser vistos como um conjunto de bibliotecas, pois cada um pode conter diversos arquivos de implementação dentro, além de possuir dependências e/ou serem dependentes de outros componentes. Mas os componentes também podem ser bastante personalizáveis e dispõem de ferramentas para facilitar a organização do código e configuração dos projetos. Uma dessas ferramentas, utilizadas neste projeto, é a criação de menus de configuração.

Os menus podem ser utilizados durante a fase de configuração do projeto, antes da compilação, para manipular diversos tipos de definições e variáveis que podem ser utilizadas no projeto. Isso facilita a criação de códigos que utilizam informações externas, como chaves de API, login e senha de WiFi, entre outros tipos de informações, pois basta que o desenvolvedor que está utilizando o projeto entre no menu e atualize as informações para que o projeto funcione.

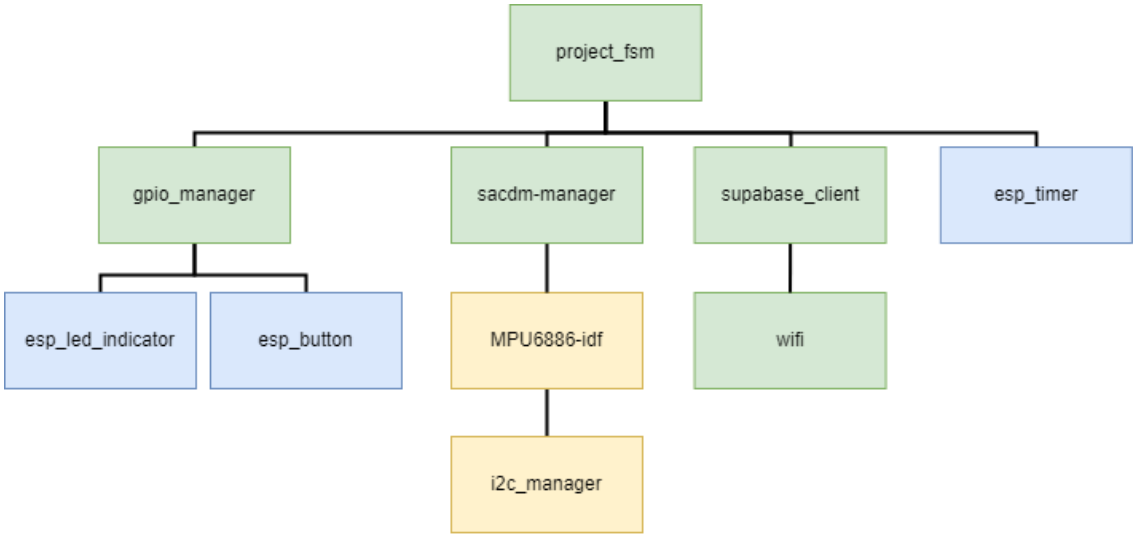
Figura 10: Diretório de componentes do projeto



Fonte: Autoria própria

A figura 10 mostra como o diretório de componentes está disposto no projeto. Embora estejam na mesma pasta, existe uma hierarquia entre eles. Devido ao tamanho do projeto, os componentes na mesma pasta permitem um acesso fácil e direto. Do contrário, poderiam ser reorganizados com suas respectivas dependências dentro um do outro.

Figura 11: Diagrama geral de componentes do projeto



Fonte: Autoria própria

A hierarquia de uso dos componentes podem ser vistos no diagrama da figura 11. Neste diagrama é possível observar três cores destacando os componentes. Os verdes representam os componentes criados pelo autor, no desenvolvimento do projeto. Os azuis são componentes oficiais, fornecidos e mantidos pela espressif. Por fim, os amarelos são criados por terceiros, mas que sofreram algumas adaptações para que funcionassem na versão atual do ESP-IDF dentro do projeto, visto que foram desenvolvidos e mantidos apenas até a versão 4.X, que por sua vez teve diversas modificações grandes em várias API's de sistema.

4.4.2.1 Componente project_fsm

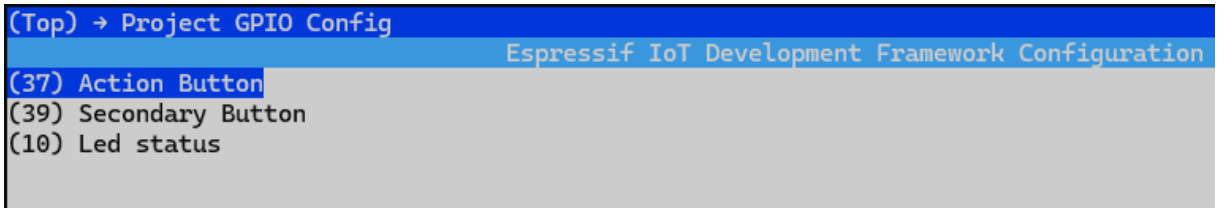
O project_fsm é o pilar da implementação do projeto. Nele existe uma função chamada *run_project_fsm()*, responsável por iniciar a máquina de estados do sistema. Por este motivo, é um componente que tem acesso a todos os outros componentes, sendo responsável por gerenciar e executar a troca de contexto de cada parte do funcionamento. Neste componente são executadas as inicializações implementadas em cada um dos outros, descritos a seguir.

4.4.2.2 Componente gpio_manager

O gpio_manager, como o nome indica, é responsável pelos elementos que estão conectados nos pinos de GPIO da placa de desenvolvimento, o LED e o botão. Cada um desses elementos possui seus próprios arquivos de configuração, inicializados individualmente. Como o diagrama anterior mostra, cada um deles utiliza outros componentes, o *esp_led_indicator* e *esp_button*. Estes auxiliam na configuração e implementação des-

sas inicializações, sendo uma interface mais amigável para o driver de gpio criado pela Espressif.

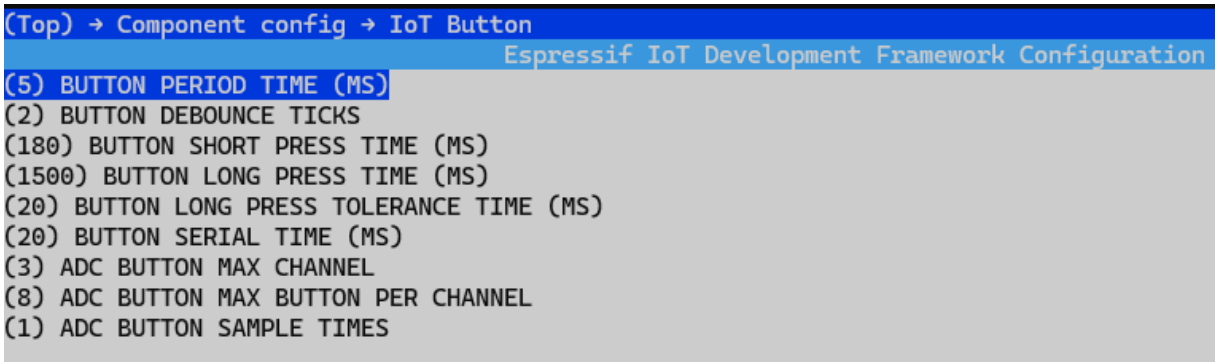
Figura 12: Menu de configuração do GPIO Manager



Fonte: Autoria própria

Na figura 12 é possível visualizar a configuração dos pinos dos botões e LED. Após a inserção do número da GPIO em que o periférico está conectado, esse valor será utilizado no código para sua inicialização.

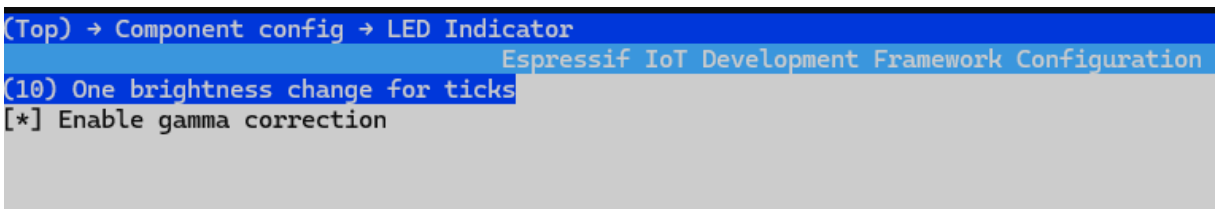
Figura 13: Menu de configuração do componente de botão da Espressif



Fonte: Autoria própria

Na figura 13 encontram-se todas as configurações do componente de botão da Espressif, onde se configura alguns parâmetros de calibração para o botão, além de alguns valores referentes aos eventos de clique do botão.

Figura 14: Menu de configuração do componente de LED da Espressif



Fonte: Autoria própria

Assim como o menu anterior, a figura 14 também mostra algumas configurações

do componente da Espressif referente ao LED, com alguns detalhes que influenciam na sua correção e temporização de brilho.

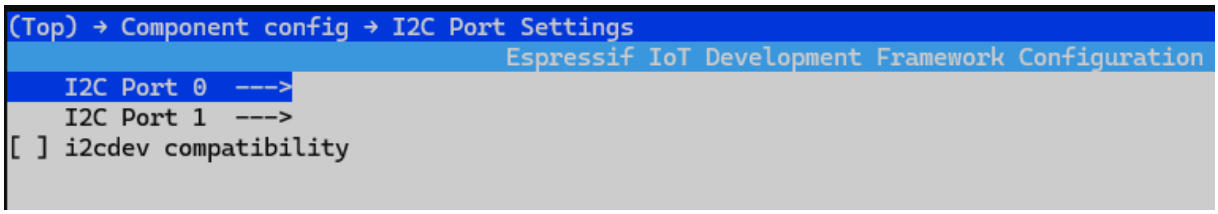
4.4.2.3 Componente MPU6886-idf

Este componente é responsável por inicializar o barramento onde o IMU está conectado, além de passar todas as configurações necessárias para que ele seja capaz de começar a gerar os dados de acelerômetro. A partir do momento que é inicializado, estes dados podem ser obtidos por qualquer outro componente do sistema através de algumas funções que retornam os valores em tempo real, podendo ser utilizados quando e como forem necessários.

4.4.2.4 Componente i2c_manager

O papel do i2c_manager é dar suporte ao funcionamento do componente anterior, o MPU6886-idf. Como o IMU utilizado no projeto utiliza o barramento I2C para se comunicar com a ESP32, este componente garante o uso e configuração deste barramento.

Figura 15: Menu de configuração do componente de I2C



Fonte: Autoria própria

Na figura 15 podemos ver as configurações das portas I2C. Por padrão, utiliza-se a porta 0.

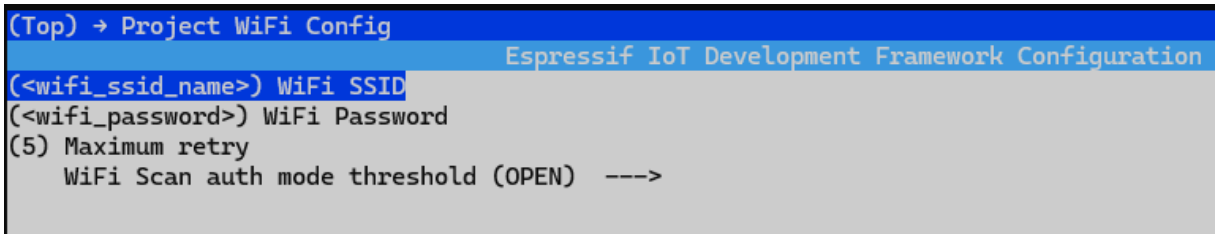
4.4.2.5 Componente sacdm_manager

No sacdm_manager é onde se encontra a lógica do algoritmo do SAC-DM. A implementação deste componente permite a obtenção dos dados de acelerômetro, para que sejam aplicados no algoritmo e assim possa gerar os resultados da quantidade de densidade de máximos.

4.4.2.6 Componente wifi

Como um dispositivo conectado, o wifi é essencial para o gerenciamento da conexão com a internet e envio de dados do projeto para a nuvem.

Figura 16: Menu de configuração do componente de WiFi

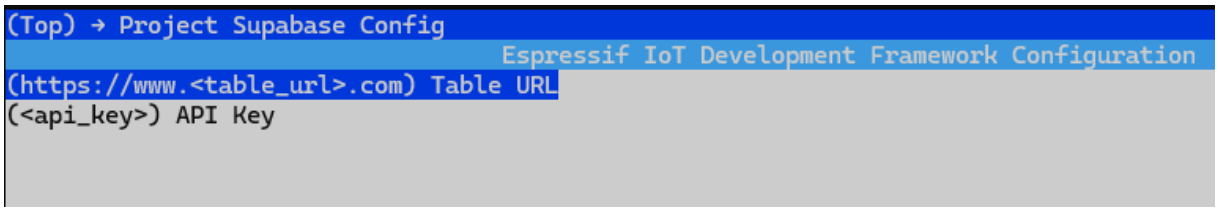


Para a configuração do WiFi, duas informações precisam ser fornecidas pelo desenvolvedor para que funcione corretamente, o nome da rede e a senha. Como é possível ver na figura 16, isso pode ser adicionado no menu do componente, desta forma essa informação não precisa ser escrita diretamente no código e é preservada apenas em um arquivo utilizado na compilação do projeto.

4.4.2.7 Componente supabase_client

O supabase_client utiliza algumas API's de requisições para web da Espressif para se conectar com a plataforma Supabase na nuvem. Depende diretamente do funcionamento do WiFi. A responsabilidade deste componente é não só se conectar, mas também organizar as informações em um formato específico para que sejam armazenadas da forma correta no banco de dados.

Figura 17: Menu de configuração do componente cliente do Supabase



Assim como no WiFi, algumas informações "secretas" precisam ser inseridas para que a conexão funcione. Da mesma forma, para que não sejam escritas diretamente no código e cada desenvolvedor consiga inserir o seu, na figura 17 podemos observar como são inseridos os dados necessários no menu para a configuração do cliente de conexão do Supabase.

4.4.2.8 Componente esp_timer

Este componente está diretamente associado ao sacdm_manager, pois a coleta e cálculo dos dados tem tempos bem definidos para que a frequência das amostras seja sem-

pre a mesma. O esp_timer é um temporizador de alta resolução fornecido pela Espressif e ajuda a garantir que isso seja possível.

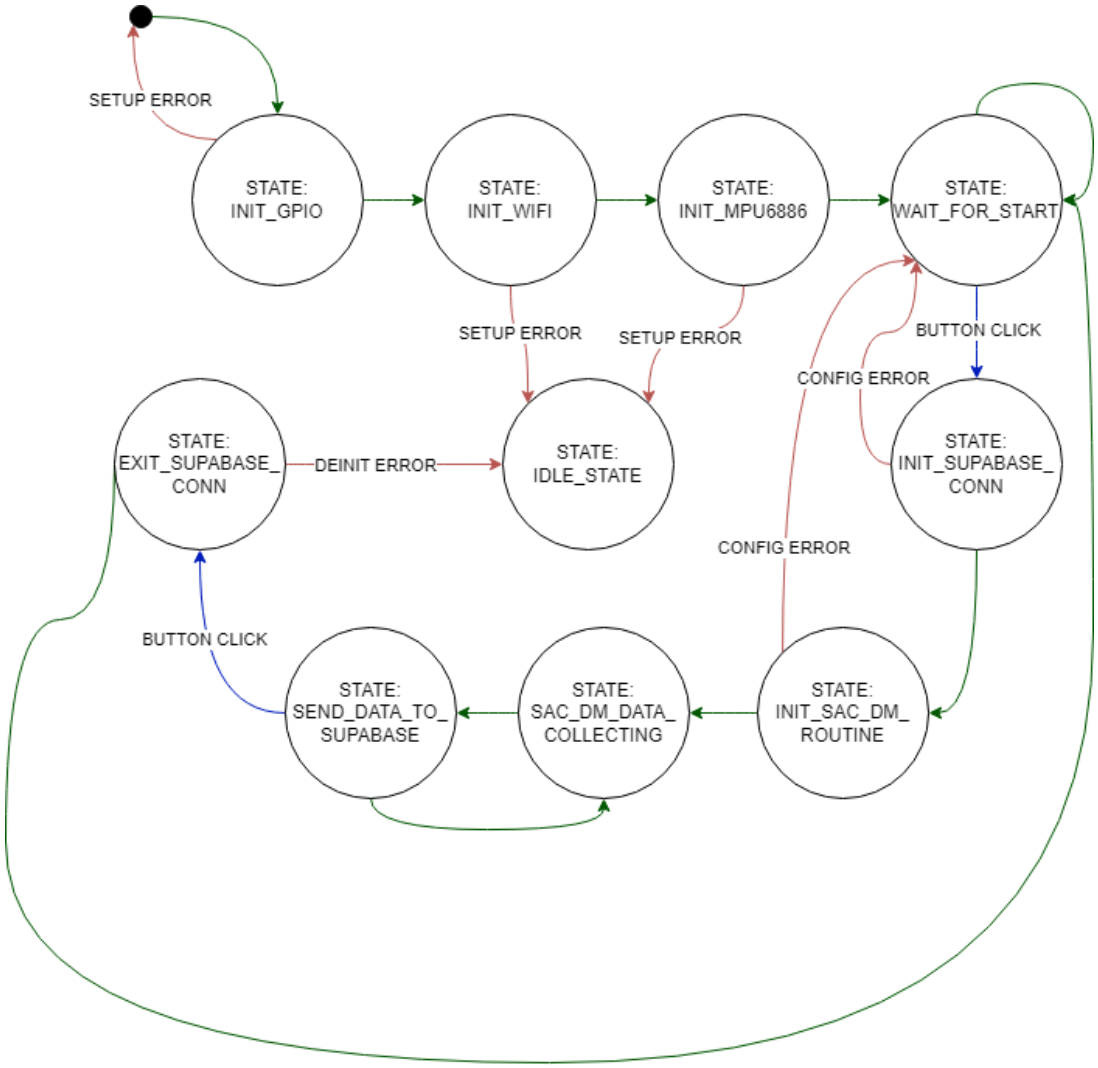
4.5 Máquina de Estados do projeto

A máquina de estados do projeto é considerada como o núcleo de funcionamento dele. Toda a organização e execução do que foi implementado nos componentes passam por cada estado da máquina. Por este motivo, quando ocorre algum tipo de problema na execução do código, é fácil identificar onde o erro ocorreu e o que exatamente o causou, visto que cada estado tem um papel específico, executa e verifica cada funcionalidade por vez.

Pode-se considerar que a lógica de todo o projeto funciona baseado em três grandes estruturas: a principal máquina de estados executada dentro do componente project_fsm e outras duas máquinas de estados do LED e Botão, executadas dentro do componente de gpio_manager.

4.5.1 Estados do projeto

Figura 18: Máquina de estados simplificada do projeto



Fonte: Autoria própria

Na figura 18 é possível identificar todos os estados do projeto. As setas verdes indicam o fluxo de funcionamento de sucesso entre um estado e outro, sendo esse o caminho esperado durante a execução. As setas vermelhas indicam o fluxo da máquina quando ocorre algum erro, e as azuis, por sua vez, indicam o fluxo que exige que uma ação seja tomada para que o próximo estado seja acionado, nesse caso a interação com o usuário do sistema embarcado para iniciar ou interromper algum processo. Caso haja algum erro, é fácil identificar qual estado está com problemas através da depuração com os logs do sistema, pois devido a forma como foi desenvolvido, cada passo é mapeado e checado com informações de sucesso e falha.

Os três primeiros estados INIT_GPIO, INIT_WIFI e INIT_MPU6886 são considerados essenciais para o funcionamento do projeto, portanto, se algum deles falhar o

sistema tenta reiniciar ou entra no estado IDLE_STATE, em que não realiza nenhuma ação devido a falta de recursos para a execução plena de suas funcinalidades. Após eles, o sistema entra no estado de WAIT_FOR_START, pronto para iniciar a execução depois que o botão principal for clicado.

Com o clique, a inicialização com a plataforma na nuvem é iniciada no estado de INIT_SUPABASE_CONN, que retornará caso a comunicação com o Supabase falhe ou seguirá para a execução do algoritmo de coleta de dados quando conseguir. No estado INIT_SAC_DM_ROUTINE, é iniciado o processo de temporização de coleta de dados, utilizando o esp_timer. Após isso, o estado SAC_DM_DATA_COLLECTING é acionado e a captura dos dados de acelerômetro no tempo exato são obtidos para a montagem da amostra de dados. Para cada amostra completada, o resultado do cálculo da densidade é calculada e enviada para o supabase no estado SEND_DATA_TO_SUPABASE. Sem mais delongas, o estado anterior de coleta é iniciado novamente e a temporização de coleta é mantida. A API da ESP32 garante que, mesmo que os dados não sejam enviados para a nuvem naquele exato momento, não há perda de tempo durante a coleta pois todo esse processo ocorre em paralelo. Com isso, mesmo que haja uma lentidão ou atraso por parte da conexão de internet, os dados podem chegar atrasados na nuvem, mas o intervalo entre eles é o mesmo de um segundo.

Caso haja um clique no botão por parte do usuário, EXIT_SUPABASE_CONN é o estado iniciado, garantindo que a execução seja encerrada corretamente com a nuvem e o sistema volta para o estado de WAIT_FOR_START, pronto para entrar novamente em execução. Sempre que uma nova conexão com o Supabase é criada, os varoles de densidade de máxima enviados são sempre -1 nos eixos X, Y e Z. Desta forma, é garantida a verificação com o banco de dados online e há uma separação clara entre o início de uma coleta e término de outra, caso ocorram várias seguidas.

4.5.2 Estados do LED e Botão

Como foi observado na figura 18, existe um fluxo de execução de código do sistema. Quando o sistema embarcado está conectado a um computador, os logs de sistema podem ser observados através do monitor e cada passo da execução pode ser lido em tempo real. Na execução do sistema em um cenário real, em um drone, isso não pode ser feito com facilidade. Por isso é necessário um feedback visual para que o usuário saiba exatamente o que está acontecendo e quando pode interagir para iniciar e/ou parar a coleta de dados.

Tabela 4: Estados do LED.

Evento da Máquina de Estados	Comportamento do LED
SUCCESS	LED pisca rapidamente (100ms)
ERROR	LED pisca duas vezes rapidamente
WAIT_FOR_START	LED aceso com metade do brilho
IDLE	LED pisca três vezes rapidamente
INVALID	LED entra em modo de fade

Fonte: Autoria própria

Na tabela 4 podemos ver o comportamento do LED para cada passo da execução. Nas linhas verdes, o estado de SUCCESS é acionado, enquanto nas linhas vermelhas, o estado de ERROR é executado. Para cada um deles, o usuário sabe exatamente o que está acontecendo e em que estado o sistema se encontra, caso precise de alguma interação.

Tabela 5: Eventos gerados pelos botões.

Nº	Estado atual	Ação baseado no click dos Botões
0	INIT_GPIO	Estado inválido, botão sendo configurado
1	INIT_WIFI	Não faz nada
2	INIT_MPU6886	Não faz nada
3	WAIT_FOR_START	Inicia aplicação, altera estado para nº4
4	INIT_SUPABASE_CONN	Limpa dados e retorna para nº3
5	INIT_SAC_DM_ROUTINE	Limpa dados e retorna para nº3
6	SAC_DM_DATA_COLLECTING	Finaliza coleta e retorna para nº8
7	SEND_DATA_TO_SUPABASE	Finaliza coleta e retorna para nº8
8	EXIT_SUPABASE_CONN	Não faz nada
9	IDLE	Reseta a ESP32, estado atual irreparável
10	default	Altera o estado para nº9

Fonte: Autoria própria

De maneira análoga ás outras máquinas de estado, o botão também tem a capacidade de realizar uma ação no sistema, baseado no estado atual. Como pode ser visto na tabela 4, a depender do estado atual, o botão tem a capacidade de alterar o fluxo de execução de acordo com o que o sistema se propõe a fazer.

4.6 Supabase

O Supabase é uma plataforma de fácil configuração. Para seu uso é apenas necessário a criação de uma conta e em seguida, um projeto para a criação dos recursos que serão utilizados.

Figura 19: Descrição da tabela do banco de dados

Fonte: Autoria própria

Com o projeto criado, automaticamente cria-se uma instância do banco de dados para o projeto. Desta forma, basta criar uma tabela nessa instância. Para este projeto, podemos ver na tabela exibida na figura 19 os atributos criados para a tabela que armazena os dados gerados pelo SAC-DM. A tabela consiste apenas de três campos: *id*, *value* e *created_at*. Os campos *id* e *created_at* são preenchidos automaticamente, com o identificador da tabela e a hora exata em que o dado foi enviado para a nuvem. O campo *value*, que é o responsável por armazenar os dados gerados pelo sistema embarcado, pode receber qualquer valor. Como esse valor pode ser utilizado por qualquer aplicação depois, seja para tratar os dados, gerar gráficos, entre outras possibilidades, ele pode ser enviado para a tabela do banco de dados em qualquer formato. No projeto atual, ele é enviado no formato JSON.

5 APRESENTAÇÃO E DISCUSSÃO DOS RESULTADOS

De acordo com a metodologia proposta para o sistema embarcado, serão apresentados neste capítulo os resultados do desenvolvimento. A seguir se encontra a descrição e observações sobre o funcionamento do sistema embarcado.

5.1 Versionamento do ESP-IDF

Figura 20: Tabela de compatibilidade com versões do ESP-IDF.

Chip	v4.3	v4.4	v5.0	v5.1	v5.2
ESP32	supported	supported	supported	supported	supported
ESP32-S2	supported	supported	supported	supported	supported
ESP32-C3	supported	supported	supported	supported	supported
ESP32-S3		supported	supported	supported	supported
ESP32-C2			supported	supported	supported
ESP32-C6				supported	supported
ESP32-H2				supported	supported
ESP32-P4					preview

Fonte: (DOCS, 2024)

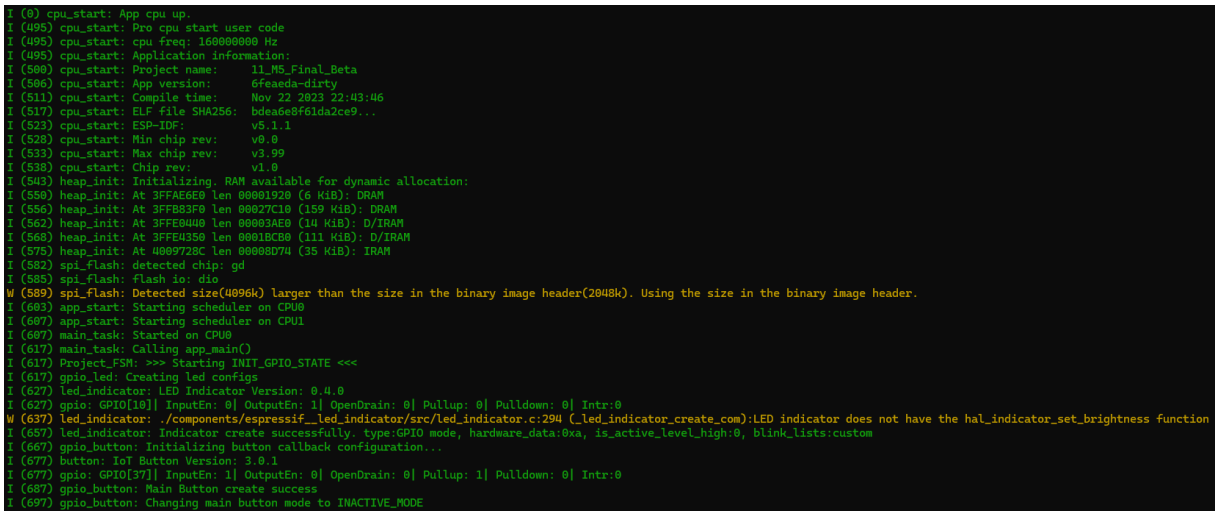
Baseado no que foi visto na seção sobre o versionamento do framework, o projeto foi desenvolvido utilizando a versão *5.1* do Framework, embora possa ser compilado com qualquer versão, desde que seja *5.X*. Como pode ser visto na figura 20, estas versões são compatíveis com basicamente todos os modelos de ESP32 já lançados. Desta forma, praticamente qualquer kit ou modelo criado com ESP32 é capaz de executar o código que foi desenvolvido, possibilitando a criação de variações de projeto e estrutura, sem se prender a apenas o kit utilizado no trabalho.

5.2 Fluxo de funcionamento

Para a validação do fluxo de funcionamento do código, foi utilizado a ferramenta *Monitor* do kit de ferramentas disponíveis dentro do ESP-IDF. O *Monitor* é responsável por exibir todas as mensagens de log do sistema que o próprio desenvolvedor colocou, através de uma comunicação serial (USB). Desta forma, a depuração do sistema pode

ser feita antes de enviá-lo para testes em campo. A seguir encontram-se todos os logs de execução do sistema, considerando um fluxo de funcionamento dentro do esperado conforme foi mencionado anteriormente na seção da Máquina de Estados. Dentre os logs, o número que se encontra entre parêntesis é o tempo de execução daquele log, dado em milissegundos, desde que o sistema iniciou. Portanto, o log de sistema indicado pelo *cpu_start* marca o começo de execução do código.

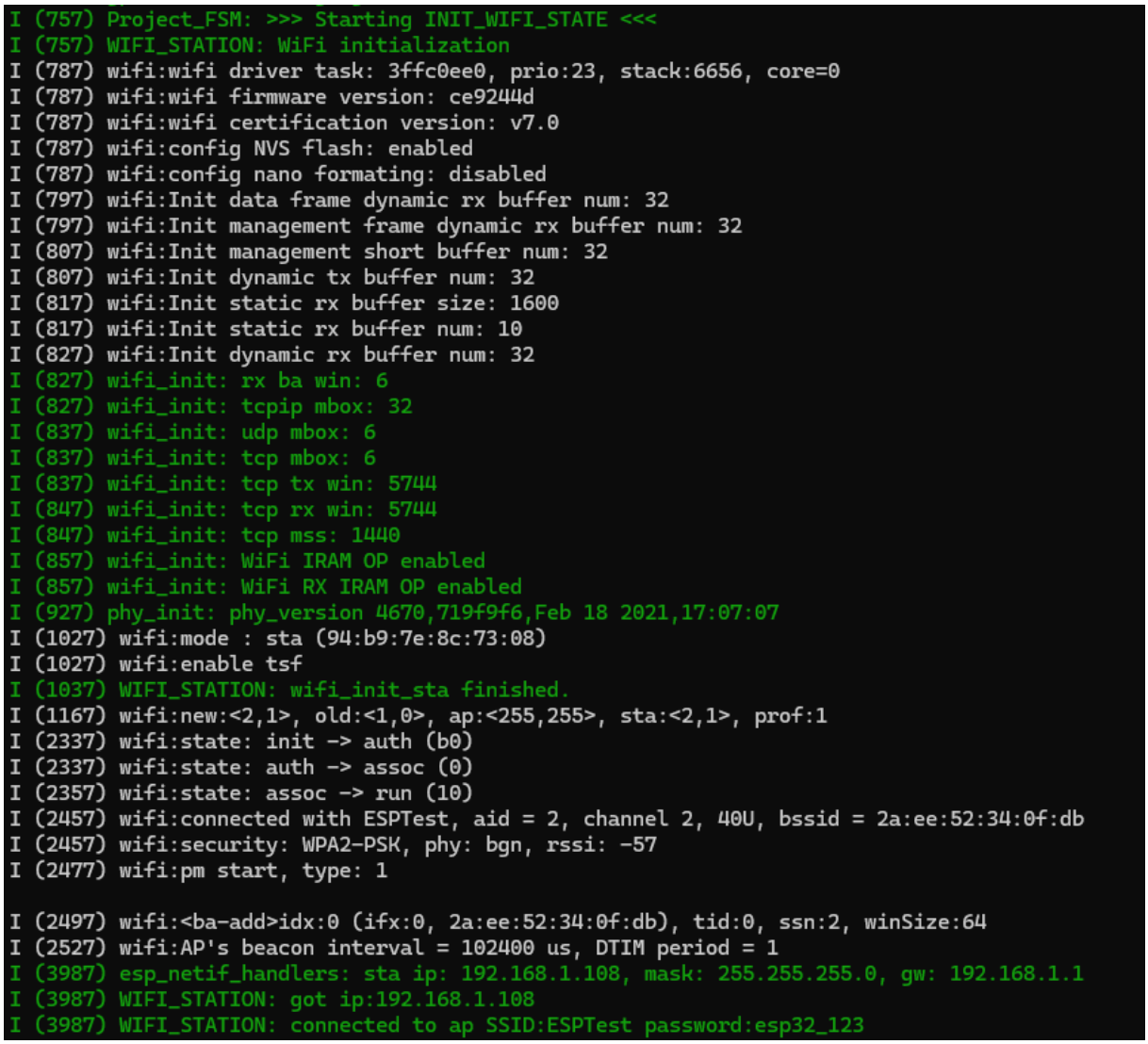
Figura 21: Logs de inicialização do GPIO.



Fonte: Autoria própria

No momento em que se inicia o estado *INIT_GPIO_STATE*, como indicado na figura 21, são criadas as configurações do LED e de Botão, através dos seus respectivos componentes. Após o término das configurações o Botão é imediatamente levado ao estado de inativo, pois seu uso ainda não é necessário, visto que ainda faltam outras inicializações. Por esse motivo, como pode ser visto na tabela 5, nesse primeiro momento ele não é capaz de realizar nenhuma ação.

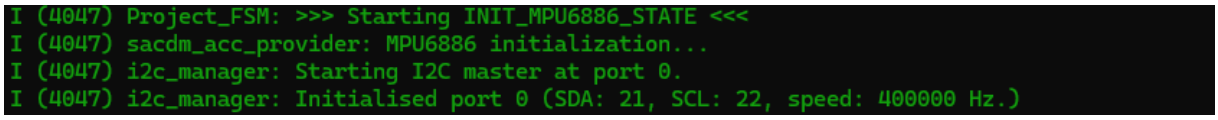
Figura 22: Logs de inicialização do WiFi.



Fonte: Autoria própria

Com a GPIO configurada corretamente, o estado *INIT_WIFI_STATE* é iniciado, dando início ao processo de configuração e conexão da ESP32 com o WiFi que foi previamente estabelecido no menu. Como está indicado nos logs da figura 22, quando a inicialização do drive de wifi no modo *station* é finalizada no tempo de execução 1037, se inicia o processo de busca e conexão com o roteador local, que nesse caso possui o nome do wifi ESPTest, criado para os testes do projeto.

Figura 23: Logs de inicialização do IMU.



Fonte: Autoria própria

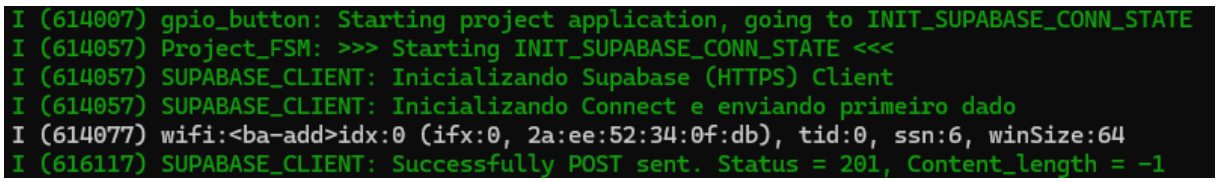
Como a figura 23 indica, após a conexão com o wifi se dá a configuração do IMU. Isso ocorre pois nesta versão do projeto, o wifi é necessário para o envio de dados do projeto para a nuvem. Por isso, ocorre algumas tentativas de conexão antes do IMU, pois sem ela não haveria o envio de dados. Com a porta I2C devidamente configurada, é possível acessar os dados fornecidos pelo MPU6886.

Figura 24: Logs de espera para início de execução.

Fonte: Autoria própria

A partir deste momento, como pode ser visto na figura 24, se inicia o estado *INIT_WAIT_FOR_START_STATE*. O botão que antes estava inativo, passa a funcionar na espera do clique do usuário para que possa ser iniciada a coleta e envio de dados.

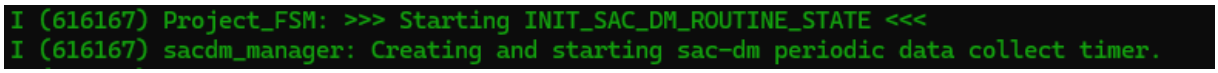
Figura 25: Logs de inicialização da conexão com a nuvem.



Fonte: Autoria própria

Após o clique inicia-se o *INIT_SUPABASE_CONN_STATE* através da máquina de estados do botão. A partir deste momento, o cliente responsável por se comunicar com o Supabase é configurado para garantir que o envio de dados funcionará corretamente.

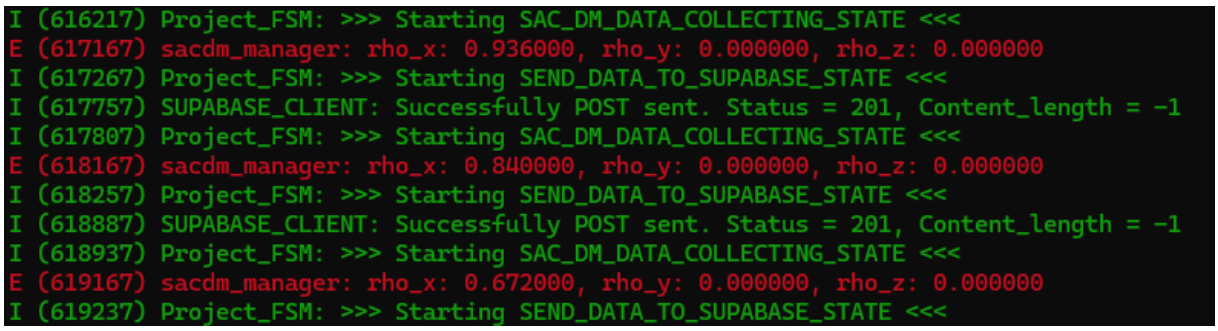
Figura 26: Logs de inicialização de rotina do algoritmo SAC-DM.



Fonte: Autoria própria

Nesse momento, são configuradas as variáveis do algoritmo e temporizador responsável pela garantia de coleta de dados nos intervalos corretos.

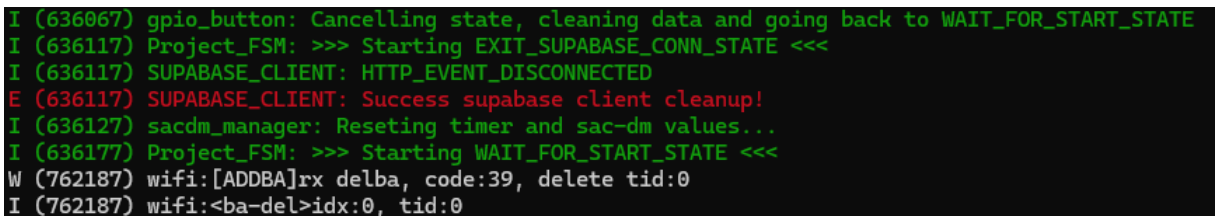
Figura 27: Logs de envio de dados para a nuvem.



Fonte: Autoria própria

Com as variáveis e temporizador do algoritmo prontos para serem executados, se inicia o processo de coleta. A frequência de captura de dados do IMU é configurado no algoritmo como padrão, de 500Hz. É possível observar na figura 27 que o intervalo de tempo entre o cálculo e envio dos dados *rho_x*, *rho_y* e *rho_z* é de exatamente 1000 milissegundos. Esses dados são formatados em uma estrutura JSON e enviados para a nuvem.

Figura 28: Logs de término de execução.



Fonte: Autoria própria

Por fim, quando o botão é clicado novamente, se dá o término da execução. Como

mostrado na figura anterior, a conexão com o Supabase é encerrada, os dados de conexão são limpos, temporizador e variáveis do SAC-DM são resetados e o estado de espera para início de coleta é iniciado novamente.

5.3 Conexão com a nuvem

Com a tabela do banco de dados criada na nuvem, uma API é gerada automaticamente e através de requisições HTTP os dados são enviados para a plataforma.

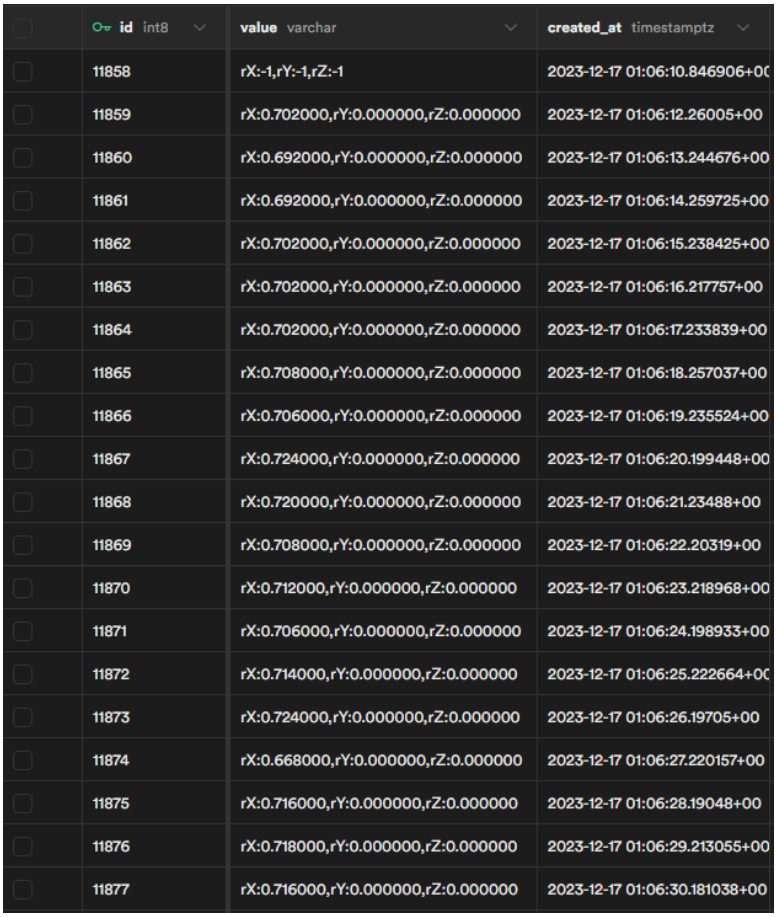
Figura 29: Formato de requisição de envio de dados para o Supabase.

```
curl - X POST 'https://<url_aleatoria>.supabase.co/rest/v1/<nome_da_tabela>'
     - H "apikey: SUPABASE_KEY"
     - H "Authorization: Bearer SUPABASE_KEY"
     - H "Content-Type: application/json"
     - H "Prefer: return=minimal"
     - d '{ "rho_x": float value, "rho_y": float value, "rho_z": float value }'
```

Fonte: Autoria própria

Como pode ser visto na figura 29, através de uma requisição POST basta preencher os campos seguindo o formato gerado automaticamente pela plataforma, com a chave gerada e entregue. Os dados devem estar no formato correto para que as informações sejam armazenadas corretamente.

Figura 30: Tabela de dados do Supabase.



id int8	value varchar	created_at timestamptz
11858	rX:-1,rY:-1,rZ:-1	2023-12-17 01:06:10.846906+0(
11859	rX:0.702000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:12.26005+00
11860	rX:0.692000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:13.244676+00
11861	rX:0.692000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:14.259725+00
11862	rX:0.702000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:15.238425+00
11863	rX:0.702000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:16.217757+00
11864	rX:0.702000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:17.233839+00
11865	rX:0.708000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:18.257037+00
11866	rX:0.706000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:19.235524+00
11867	rX:0.724000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:20.199448+00
11868	rX:0.720000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:21.23488+00
11869	rX:0.708000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:22.20319+00
11870	rX:0.712000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:23.218968+00
11871	rX:0.706000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:24.198933+00
11872	rX:0.714000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:25.222664+0(
11873	rX:0.724000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:26.19705+00
11874	rX:0.668000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:27.220157+00
11875	rX:0.716000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:28.19048+00
11876	rX:0.718000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:29.213055+00
11877	rX:0.716000,rY:0.000000,rZ:0.000000	2023-12-17 01:06:30.181038+00

Fonte: Autoria própria

Na figura 30 é possível ver como os dados ficam armazenados na tabela após a coleta e envio das coletas do sistema embarcado. De maneira análoga ao envio, a plataforma também fornece em sua API uma forma fácil de recuperar estas informações, podendo este resultado ser utilizado em qualquer aplicação para visualização e/ou tratamento.

5.4 Sistema Embarcado

Figura 31: Sistema aguardando início da coleta de dados.

Fonte: Autoria própria

Após alguns testes de funcionamento do sistema embarcado, foi possível determinar que o tempo médio de coleta de dados ininterruptas com o kit M5StickC Plus é de 1 hora e 15 minutos. Como exemplo do kit no estado de espera, pode-se observar na figura 32 como o LED fica aceso para indicar o aguardo do clique.

Figura 32: M5StickC Plus embarcada em um Drone DJI Tello.

Fonte: Autoria própria

Por ser leve e possuir uma bateria interna, é fácil embarcá-lo em diversos tipos de Drone. A figura acima mostra um exemplo prático de um Drone cujo modelo é o DJI Tello, que por si só já é extremamente leve e não aceita muita carga além do seu próprio peso, mas que consegue suportar o kit utilizado neste projeto. É necessário ressaltar que para este drone especificamente, o sistema pode influenciar na performance do voo e duração da bateria do Drone, mas para drones maiores a influência do sistema embarcado é mínima.

No que diz respeito a comunicação do sistema, comparado a outras tecnologias de comunicação sem fio, o Wi-Fi possui algumas limitações em relação a distância de comunicação, que para aplicação com Drones parece ser inviável. No entanto, para validação e consolidação do sistema embarcado proposto em testes indoor, é mais que suficiente para o correto funcionamento e comunicação com a nuvem. Além disso, por ter sido desenvolvido em uma estrutura baseada em componentes, o componente de comunicação Wi-Fi pode ser facilmente substituído por outro componente e módulos de comunicação que eventualmente possam fazer mais sentido, a depender da comunicação exigida pela aplicação do drone.

Figura 33: Diferentes modelos de chips utilizados na validação.

Fonte: Autoria própria

Como um dos principais destaques mencionados, a compatibilidade do código com os diferentes modelos de ESP32 foi validada para diferentes variações do chip. As variações utilizadas podem ser vistas na figura 33, sendo eles o M5StickC Plus, M5StickC Plus 2 e o Devkit mais comum, facilmente encontrado no mercado. Cada um desses kits possui uma variação diferente de ESP32, capaz de rodar o código do projeto sem nenhuma configuração ou modificação adicional.

6 CONCLUSÕES E TRABALHOS FUTUROS

O principal objetivo do desenvolvimento deste trabalho foi a implementação de um sistema embarcado robusto e de fácil aplicação em um Drone. Devido a distância do autor do trabalho ao laboratório onde diversos Drones podem ser testados, não foi possível testar em diversos modelos de Drone, apenas pôde ser validado com o Tello apresentado anteriormente. No entanto, por ser uma evolução de um outro sistema embarcado desenvolvido e testado anteriormente, é possível afirmar que seu uso é igualmente ou até mais prático, devido aos componentes integrados no sistema.

A coleta e fornecimento de dados pôde ser feita normalmente, gerando os resultados esperados do SAC-DM. No entanto, para que o Sistema Embarcado implementado seja melhor aplicado, é necessária a criação de casos de teste para cada aplicação e cada tipo de Drone. Por exemplo, podem ser coletados dados de teste em um Drone com uma hélice perfeitamente estável e alinhada, como uma dessas hélices pode ser propositalmente danificada com o intuito de registrar a diferença nos resultados. A posição do sistema embarcado também influencia nos resultados, podendo também ser alterada para que os dados coletados gerem diferentes comportamentos e o melhor deles possa ser escolhido para validação.

A frequência de coleta de dados pode ser facilmente alterada, aumentando ou diminuindo o tamanho da amostra no componente relacionado ao algoritmo do SAC-DM. Desta forma, a sensibilidade do sistema será alterada e pode ser capaz de captar mais ou menos variações no Drone, possibilitando uma "calibração"do sistema através de software. Em teoria isso pode ser ajustado até o limite máximo do IMU de 400kHz, mas na prática existem limitações de software e envio de dados que podem não suportar frequências tão altas de coleta, portanto os limites máximos e mínimos de frequência de coleta devem ser testados para melhor obtenção dos resultados.

Como foi mostrado, todo o software é compatível com diversas versões do ESP32. Para trabalhos futuros poderiam ser adicionados mais componentes para o sistema, como suporte a outros sensores, mais atuadores para interação com usuário. Para tornar o software escalável e fácil de utilizar, poderia ser criada uma camada HAL (Hardware Abstraction Layer), a fim de suportar os diversos sensores existentes no mercados e não seria necessário se limitar a apenas o que está ingerado ao kit (MPU6886). Desta forma, seria possível utilizar qualquer modelo de ESP32, com qualquer sensor, adicionando apenas um componente novo, dando a liberdade de confeccionar as próprias placas ou utilizar outros kits que forem melhor aplicados no Drone ou projeto em questão.

Durante o desenvolvimento do trabalho a empresa M5Stack foi comprada pela Espressif, o que significa que a integração entre os produtos e kits além de melhores, se tornarão oficiais. Isso possibilita que um componente do framework conhecido como BSP

(Board Support Package) seja utilizado em todos os projetos, facilitando a integração de novos componentes, importação de drivers e generalização de hardware que facilitarão ainda mais projetos como este. O uso desse sistema pode acelerar e contribuir para a criação de cada vez mais protótipos funcionais sem a necessidade de criar tudo do zero.

REFERÊNCIAS

ALTINORS, A.; YOL, F.; YAMAN, O. A sound based method for fault detection with statistical feature extraction in uav motors. *Applied Acoustics*, v. 183, p. 108325, 2021. ISSN 0003-682X. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0003682X21004199>.

AXELSON, J. *Serial Port Complete*. 2007. Disponível em: <https://wiki.sj.ifsc.edu.br/images/7/73/USART.pdf>. Acesso em: 15 Fev de 2024.

BONDYRA, A. et al. Fault diagnosis and condition monitoring of uav rotor using signal processing. In: *2017 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*. [S.l.: s.n.], 2017. p. 233–238.

BYTESOFGIGABYTES. *How HTTP request and response works – Bytesof-Gigabytes*. 2020. Disponível em: <https://bytesofgigabytes.com/networking/how-http-request-and-response-works/>.

CAMARA, R. *Barramento e Protocolo I2C*. 2013. Disponível em: <http://www.univasf.edu.br/ romulo.camara/novo/wp-content/uploads/2013/11/Barramentoe-Protocolo-I2C.pdf > Acesso em: 15 Fev de 2024.

CAPRIGLIONE, D. et al. Experimental Analysis of Filtering Algorithms for IMU-Based Applications Under Vibrations. *IEEE Transactions on Instrumentation and Measurement*, v. 70, p. 1–10, 2021. ISSN 1557-9662. Conference Name: IEEE Transactions on Instrumentation and Measurement. Disponível em: <https://ieeexplore.ieee.org/abstract/document/9292928>.

CISCO. *O que é Wi-Fi? - Definição e tipos*. 2020. Disponível em: <https://www.cisco.com/c/pt_br/products/wireless/what-is-wifi.html>.

DATASHEET, E. *ESP32 Series Datasheet*. 2023. Disponível em: <https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf>. Acesso em: 26 Set de 2023.

DOCS, E. E.-I. *ESP-IDF Programming Guide for ESP32*. 2024. Disponível em: <https://docs.espressif.com/projects/esp-idf/en/v5.1.1/esp32/esp-idf-en-v5.1.1-esp32.pdf>. Acesso em: 26 Set de 2023.

ESP-IDF. *ESP-IDF Official Site*. 2024. Disponível em: <https://www.espressif.com/en/products/sdks/esp-idf>. Acesso em: 26 Set de 2023.

ESP32, A. *Arduino Core for ESP*. 2023. Disponível em: < https://github.com/espressif/arduino-esp32>. Acesso em: 26 Set de 2023.

GHAZALI, M. H. M.; RAHIMAN, W. An investigation of the reliability of different types of sensors in the real-time vibration-based anomaly inspection in drone. *Sensors*, v. 22, n. 16, 2022. ISSN 1424-8220. Disponível em: <https://www.mdpi.com/1424-8220/22/16/6015>.

IBM. *PaaS*. 2023. Disponível em: < https://www.ibm.com/br-pt/topics/paas>. Acesso em: 26 Set de 2023.

M5STACK. *M5StickC-Plus Library*. 2024. Disponível em: <https://github.com/m5stack/M5StickC-Plus>. Acesso em: 03 Jan de 2024.

MOHAMED, K. S. IoT Physical Layer: Sensors, Actuators, Controllers and Programming. In: MOHAMED, K. S. (Ed.). *The Era of Internet of Things: Towards a Smart World*. Cham: Springer International Publishing, 2019. p. 21–47. ISBN 978-3-030-18133-8. Disponível em: <https://doi.org/10.1007/978-3-030-18133-8_2>.

NIKOLOV, N. Research of the communication protocols between the iot embedded system and the cloud structure. In: IEEE. *2018 IEEE XXVII International Scientific Conference Electronics-ET*. [S.l.], 2018. p. 1–4.

POSTGRESQL. *What is PostgreSQL*. 2024. Disponível em: <https://www.postgresql.org/ > Acesso em: 16 Fev de 2024.

SAIED, M. et al. Fault diagnosis and fault-tolerant control of an octorotor uav using motors speeds measurements. *IFAC-PapersOnLine*, v. 50, n. 1, p. 5263–5268, 2017. ISSN 2405-8963. 20th IFAC World Congress. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2405896317308315>.

SAMIE, F.; BAUER, L.; HENKEL, J. IoT technologies for embedded computing: a survey. In: *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*. New York, NY, USA: Association for Computing Machinery, 2016. (CODES '16), p. 1–10. ISBN 978-1-4503-4483-8. Disponível em: <https://doi.org/10.1145/2968456.2974004>.

SANTANA, B. *O Que é HTTP e Como Ele Permite o Acesso ao Seu Site*. 2023. Disponível em: <https://www.hostinger.com.br/tutoriais/http>.

SOLUTION, E. I. *Espressif IoT Solution*. 2024. Disponível em: < https://github.com/espressif/esp-iot-solution/tree/release/v1.1>. Acesso em: 26 Set de 2023.

SOUZA, J. S. et al. Motor speed estimation and failure detection of a small UAV using density of maxima. *Frontiers of Information Technology & Electronic Engineering*, v. 22, n. 7, p. 1002–1009, jul. 2021. ISSN 2095-9230. Disponível em: <https://doi.org/10.1631/FITEE.2000149>.

SOUZA, J. Silva de et al. Motor speed estimation and failure detection of small uav using density of maxima. *Frontiers of Electrical and Electronic Engineering in China*, v. 1, p. 1–5, 07 2020.

SUPABASE. *Supabase*. 2023. Disponível em: < https://supabase.com/docs>. Acesso em: 26 Set de 2023.

WHITE, E. *Making Embedded Systems: Design Patterns for Great Software*. O'Reilly Media, Incorporated, 2011. (Oreilly and Associate Series). ISBN 9781449302146. Disponível em: <https://books.google.com.br/books?id=bI8w17SyNdYC>.