UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA

DAVI DA SILVA ESTRELA

Projeto de um computador de voo destinado à coleta de dados e à ativação de eventos em dispositivos de trajetória balística.

Uberlândia - MG
2024

DAVI DA SILVA ESTRELA

Título do trabalho: Projeto de um computador de voo destinado à coleta de dados e à ativação de eventos em dispositivos de trajetória balística.

Projeto de Fim de Curso apresentado à Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica.

Área de concentração: Engenharia Mecatrônica.

Orientador: Roberto Mendes Finzi Neto.

Uberlândia - MG

2024

Ficha Catalográfica Online do Sistema de Bibliotecas da UFU
com dados informados pelo(a) próprio(a) autor(a).

E82
2024

Estrela, Davi da Silva, 2001-
Projeto de um computador de voo destinado à coleta de dados e à ativação de eventos em dispositivos de trajetória balística. [recurso eletrônico] / Davi da Silva Estrela. - 2024.

Orientador: Roberto Mendes Finzi Neto.
Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Uberlândia, Graduação em Engenharia Mecatrônica.
Modo de acesso: Internet.
Inclui bibliografia.

1. Mecatrônica. I. Finzi Neto, Roberto Mendes ,1974-, (Orient.). II. Universidade Federal de Uberlândia. Graduação em Engenharia Mecatrônica. III. Título.

CDU: 621.03

Bibliotecários responsáveis pela estrutura de acordo com o AACR2:

Gizele Cristine Nunes do Couto - CRB6/2091
Nelson Marcos Ferreira - CRB6/3074

DAVI DA SILVA ESTRELA

Título do trabalho: Projeto de um computador de voo destinado à coleta de dados e à ativação de eventos em dispositivos de trajetória balística.

Projeto de Fim de Curso apresentado à Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia como requisito parcial para obtenção do título de bacharel em Engenharia Mecatrônica.

Área de concentração: Engenharia Mecatrônica.

Uberlândia, 27 de setembro de 2024.

Banca Examinadora:

Documento assinado digitalmente
ROBERTO MENDES FINZI NETO
Data: 29/10/2024 09:18:00-0300
Verifique em https://validar.iti.gov.br

Roberto Mendes Finzi Neto – Doutorado (UFU)

Documento assinado digitalmente
ALEXANDRE ZUQUETE GUARATO
Data: 09/11/2024 15:34:27-0300
Verifique em https://validar.iti.gov.br

Alexandre Zuquete Guarato – Doutorado (UFU)

José Jean Paul Zanlucchi de Souza Tavares – Doutorado (UFU)

Documento assinado digitalmente
JOSE JEAN PAUL ZANLUCCHI DE SOUZA TAVAR
Data: 09/11/2024 20:48:24-0300
Verifique em https://validar.iti.gov.br

Este trabalho é dedicado a todos os meus entes queridos que me acompanharam durante a jornada, especialmente meus amigos da EPTA, que compartilharam da minha emoção e do meu esforço. Mas, acima de todos, é para meus pais que eu sinto uma especial gratidão. Eles me deram apoio e confiança, me permitindo superar os obstáculos e nunca desistir, mesmo nos momentos mais difíceis.

RESUMO

Este trabalho apresenta o desenvolvimento de um computador de voo destinado a foguetes de modelismo, com foco na coleta de dados e na ativação de eventos durante o voo. O objetivo principal é fornecer uma base clara e didática para futuros desenvolvimentos de sistemas aviônicos, facilitando o acesso às informações necessárias para a elaboração de projetos nessa área. Além de abordar os requisitos de hardware e software, o documento apresenta técnicas de calibração de sensores, como acelerômetros e giroscópios, e detalha a lógica de funcionamento do sistema. Foram realizadas simulações utilizando MATLAB e OpenRocket para validar a eficácia do sistema em diferentes cenários de voo. Este trabalho visa servir como ponto de partida para futuras pesquisas e projetos acadêmicos, contribuindo para o avanço técnico no campo do foguetemodelismo.

Palavras-chave: foguetemodelismo, aviônica, sistema embarcado, captura de dados, trajetória balística.

ABSTRACT

This study presents the development of a flight computer for model rocketry, focusing on data acquisition and event triggering during flight. The main objective is to provide a clear and comprehensive foundation for the development of avionics systems, facilitating access to the necessary knowledge for future projects in this field. The document covers hardware and software requirements, as well as sensor calibration techniques for accelerometers and gyroscopes, detailing the system's logic. Simulations were performed using MATLAB and OpenRocket to validate the system's effectiveness in various flight scenarios. This work aims to serve as a starting point for future research and academic projects, contributing to the technical advancement of model rocketry.

Keywords: model rocketry**,** avionics, embedded system, data acquisition, ballistic trajectory

SUMÁRIO

1. INTRODUÇÃO AOS SISTEMAS AVIÔNICOS EM FOGUETEMODELISMO 12

1.1. Objetivos propostos 19

1.2. Fundamentação teórica básica 19

1.3. Metodologia adotada 20

2. REQUISITOS DE HARDWARE 21

2.1. Requisitos do Sistema 21

2.2. Instrumentação de Sensores 22

2.3. Armazenamento de Dados 23

2.4. Sinalização audiovisual 23

2.5. Acionamento de Carga Pirotécnica 23

2.6. Exportação de Dados 23

2.7. Proteção contra Água 24

2.8. Interface Homem-Máquina (IHM) 24

3. REQUISITOS DE SOFTWARE 25

3.1. Saída do Modo de Suspensão 25

3.2. Modo de Configuração 25

3.3. Detecção de Lançamento 26

3.4. Filtragem de Dados 27

3.5. Detecção de Desligamento durante Voo 27

3.6. Gerenciamento de Arquivos de Dados de Voo 27

3.7. Dados de Parâmetros 28

4. PROJETO DE HARDWARE 29

4.1. Escolha dos componentes 29

4.2. Comunicação I2C 41

4.2.1. História e Funcionamento do Protocolo I2C 41

4.2.2. Funcionamento Básico 42

4.2.3. Vantagens e Desvantagens 44

4.2.4. Conexão Pull-Up 44

4.2.5. Componentes e Aplicações 45

4.3. Barramento de comunicação SPI 45

4.3.1. História e funcionamento do protocolo SPI 45

4.3.2. Funcionamento Básico 46

4.3.3. Etapas da comunicação SPI 46
4.3.4. Vantagens e Desvantagens 47
4.3.5. Aplicações e Componentes 48
4.3.6. Configuração Pull-Up no protocolo SPI 48
4.4. Modelagem da placa de circuitos através do Altium Designer 48
4.4.1. Circuito de alimentação do sistema 50
4.4.2. Circuito do processador 52
4.4.3. Circuito do barômetro 53
4.4.4. Circuito acelerômetro 54
4.4.5. Circuito de acionamento da carga pirotécnica 55
4.4.6. Circuito das sinalizações sonora e luminosa 56
4.4.7. Circuito para a utilização do cartão de memória 56
4.4.8. Circuito dos componentes da IHM 57
4.4.9. Modelagem 3D completa da placa de circuito 58
5. PROJETO DE SOFTWARE 60
5.1. Linguagem de programação utilizada no projeto 61
5.2. Utilização do acelerômetro 63
5.2.1. Inicialização do Acelerômetro 63
5.2.2. Protocolo de Comunicação 63
5.2.3. Leitura dos Dados 63
5.3. Utilização do barômetro 64
5.3.1. Inicialização e Configuração 64
5.3.2. Leitura dos Dados 64
5.3.3. Procedimento de Leitura 64
5.4. Controle das indicações sonora e luminosa 65
5.4.1. Inicialização e Configuração dos Componentes 65
5.5. Utilização dos elementos da IHM 66
5.5.1. Display I2C 66
5.5.2. Chave Rotativa com Encoder Integrado 66
5.5.3. Interface 66
5.5.4. Funcionamento 67
5.5.5. Lógica para leitura da chave rotativa 68
5.5.6. Algoritmo para utilização do display 68
5.6. Utilização do cartão de memória 69

5.7. Métodos para acionamento da recuperação **70**
5.8. Acionamento barométrico **71**
5.9. Máquina de estados do sistema **71**
6. INSTRUMENTAÇÃO DOS SENSORES **74**
6.1. Calibração do barométrico **75**
6.1.1. Princípios de funcionamento de barômetros digitais 75
6.1.2. Métodos de calibração de barômetros digitais 76
6.2. Calibração do acelerômetro **76**
6.2.1. Princípios de funcionamento de acelerômetros digitais 77
6.2.1. Métodos de calibração de acelerômetros e giroscópios digitais 77
7. TESTES FUNCIONAIS DO SISTEMA **79**
7.1. Teste do barômetro **79**
7.2. Teste do acelerômetro **83**
7.3. Testes de tratamento de erros **84**
7.4. Teste da IHM **85**
7.5. Teste da máquina de estados **88**
7.6. Testes das condições de acionamento da carga pirotécnica **90**
7.7. Simulação do funcionamento do sistema completo **92**
8. DISCUSSÃO **96**
9. CONCLUSÃO **98**
10. REFERÊNCIAS **100**
APÊNDICE **102**
APÊNDICE A – CodigoTCC.ino (Arquivo para a execução do sistema) **102**
APÊNDICE B – IHM.ino (Controle do display) **104**
APÊNDICE C – SDCard.ino (Controle da leitura e escrita na memória) **116**
APÊNDICE D – controleEventos.ino (Controle da ativação de eventos durante o voo) **120**
APÊNDICE E – EstadosVoo.ino (Controle dos estados do sistema) **123**
APÊNDICE F – IHM.ino (Controle do display, para simulação) **126**
APÊNDICE G – controleEventos.ino (Controle da ativação de eventos durante o voo, para simulação) **135**

1. INTRODUÇÃO AOS SISTEMAS AVIÔNICOS EM FOGUETEMODELISMO

Ao longo da história humana, foram desenvolvidos diversos dispositivos capazes de se deslocar pelo ar, como helicópteros, aviões, mísseis, foguetes etc. Consequentemente, um aparato tecnológico precisou ser desenvolvido para auxiliar esses dispositivos a cumprirem seus propósitos. Dessa forma, a grande maioria dos equipamentos contava com um sistema eletromecânico (mais antigo) ou eletroeletrônico (moderno). Assim, consolidou-se o termo aviônica, que se refere a todos os aparatos elétricos, eletrônicos, de sensoriamento, de atuadores, de comunicação e instrumentais presentes em tais máquinas (HELFRICK, 2012).

No tocante ao foguetemodelismo, a aviônica mostra-se intimamente relacionada ao cumprimento da missão em cada lançamento, pois todo foguete precisa realizar uma gama de tarefas básicas. Essas tarefas vão desde validar seu deslocamento e coletar dados, até realizar eventos durante o voo, como liberar um satélite em uma altitude pré-determinada ou acionar seu sistema de recuperação, abrindo seu paraquedas. Tudo isso é possível graças a um sistema aviônico integrado que controla todos os eventos do dispositivo durante o voo.

A aviônica no foguetemodelismo está diretamente ligada aos sensores presentes no foguete. Dentre os mais comuns estão:

- Barômetro: capaz de efetuar leituras barométricas como pressão, temperatura e umidade, havendo a possibilidade de ser utilizado como altímetro (Imagem 01);
- Acelerômetro e giroscópio: sensores capazes de aferir dados dinâmicos de maneira instantânea, como aceleração linear na direção dos três eixos principais (x, y e z) e velocidade de rotação em torno deles. Esses sensores são excelentes para obter dados de atitude do voo (Imagem 02);
- Sensores da classe “tilt”: utilizados para obter dados de inclinação e vibração de maneira mais simplificada (Imagem 03);
- Módulo de GPS: Sistema Global de Posicionamento (Imagem 04);
- Rádios de transmissão (Imagem 05);
- Saídas pirotécnicas para acionamento de cargas explosivas, cargas de ar comprimido, servomotores, dentre outros aparatos que variam de acordo com o propósito do foguete.

Imagem 01 – Sensor barométrico.

Fonte: Sensor de pressão barométrico BMP180 [03 de agosto de 2024]. Disponível em: www.eletrogate.com.

Imagem 02 – Sensor acelerômetro.

Fonte: Acelerômetro e giroscópio de 3 eixos 6 DOF MPU6050 [03 de agosto de 2024]. Disponível em: www.eletrogate.com.

Imagem 03 – Sensor Tilt.

Fonte: Módulo de sensor de inclinação SW-520D [03 de agosto de 2024]. Disponível em: www.eletrogate.com.

Imagem 04 – Módulo GPS.

Fonte: Módulo GPS NEO-6M com antena [03 de agosto de 2024]. Disponível em: www.eletrogate.com.

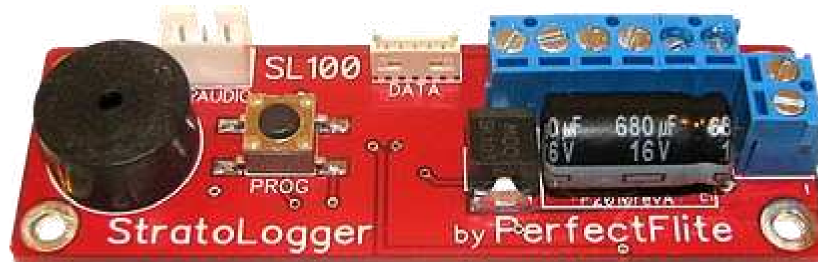
Imagem 05 – Módulo comunicação de rádio LoRa.

Fonte: Módulo RF wireless LoRa 433MHz [03 de agosto de 2024]. Disponível em: www.eletrogate.com.

Contudo, o sistema aviônico não é projetado para ser uma solução única e geral para o controle do foguete. É comum o emprego de pelo menos dois sistemas aviônicos independentes como redundância. Dessa forma, o sistema principal sempre possui um substituto para assumir em caso de falha. A redundância pode parecer algo a ser evitado em projetos, seja para torná-los mais otimizados ou mais compactos, mas em dispositivos voadores, nos quais milhares de dados estão sendo processados por segundo, qualquer falha de funcionamento tem um grande potencial para causar a perda catastrófica do equipamento. Por isso, o emprego de sistemas redundantes e independentes apresenta-se como uma solução indispensável para todos os

lançamentos. As Imagens 6 a 8 apresentam exemplos de sistemas aviônicos que podem atuar tanto como sistema principal como redundância.

Imagem 6 – Sistema aviônico (COTS STRATOLOGGER).



Fonte: Stratologger SL100 Altimeter [03 de agosto de 2024]. Disponível em: www.perfectflite.com.

Imagem 07 – Sistema aviônico (RRC3 Altimeter)

Fonte: RRC3 Altimeter [03 de agosto de 2024]. Disponível em: www.missileworks.com.

Imagem 08 – Sistema aviônico (EPTA 2023)

Fonte: O Autor.

Notoriamente, existem acionamentos cruciais para garantir o sucesso do lançamento e do pouso, e dentre eles está o acionamento da recuperação do dispositivo. Atualmente, existem diversas maneiras de fazê-lo, como utilizando airbags, fazendo o foguete pousar em pé usando

vetores de empuxo (Imagem 09), suportes anti-impacto (Imagens 10 e 11) projetados para fora do foguete no momento da descida ou, o mais tradicional, a abertura de um paraquedas (Imagem 12) em algum momento após alcançar o apogeu da trajetória. Todos esses métodos são exemplos de mecanismos de recuperação acionados pela aviônica embarcada no foguete para garantir a integridade do dispositivo durante o pouso.

Imagem 09 – Empuxo vetorizado.

Fonte: Thrust vector control - Series [03 de agosto de 2024]. Disponível em: www.youtube.com/@BPSspace.

Imagem 10 – Suporte anti-impacto (visão lateral).

Fonte: Thrust vector control - Series [03 de agosto de 2024]. Disponível em: www.youtube.com/@BPSspace.

Imagem 11 – Suporte anti-impacto (visão completa)

Fonte: Thrust vector control - Series [03 de agosto de 2024]. Disponível em: www.youtube.com/@BPSspace.

Imagem 12 – Paraquedas para foguete modelismo

Fonte: Custom BPS parachutes [03 de agosto de 2024]. Disponível em: BPS.Space.

Entretanto, a aviônica não lida apenas com o hardware e mecanismos empregados, também precisa desenvolver o software embarcado para garantir o controle de todas essas ações. Assim, ela assegura a correta leitura dos dados, análise, transmissão e acionamento de eventos. Dessa forma, garante a robustez do sistema como um todo, tratando erros e aplicando um algoritmo capaz de suportar a dinâmica de um lançamento.

Além disso, os sistemas aviônicos contam com o grande auxílio de sistemas de simulação, como softwares privados ou simulações numéricas, para validar o funcionamento correto de seus sistemas. Nessas simulações, é possível criar um cenário de lançamento controlado e variar os parâmetros conforme a necessidade da missão, testando assim a eficiência do sistema e as condições de acionamento de eventos, por exemplo. Após isso, é possível comparar o desempenho da simulação com os dados reais coletados durante o voo, refinando ainda mais o modelo virtual do lançamento e possibilitando uma otimização adicional do sistema aviônico.

1.1. Objetivos propostos

Diante do exposto, este trabalho tem como objetivo central a elaboração de um documento que sirva como base e guia para o desenvolvimento de sistemas aviônicos voltados ao foguetemodelismo, através do desenvolvimento de um projeto de própria autoria que seja economicamente viável. Observou-se uma grande dificuldade em encontrar fontes didáticas e acessíveis que ofereçam instruções para o desenvolvimento de projetos como este. Assim, o presente trabalho visa suprir essa lacuna ao apresentar, de maneira clara e didática, os componentes, as técnicas e a lógica de funcionamento de um sistema funcional de aviônica.

Ao longo do documento, são descritos os principais sensores e atuadores utilizados, além das técnicas de processamento de dados e controle para garantir a eficácia e segurança do sistema durante o voo. O objetivo final é fornecer um ponto de partida sólido para o desenvolvimento de futuros trabalhos no campo do foguetemodelismo, destacando também a busca por uma solução economicamente viável no desenvolvimento de uma placa de circuito impresso, adaptada para reduzir custos, sem comprometer a funcionalidade e a eficiência do sistema.

1.2. Fundamentação teórica básica

A aviônica em sistemas de foguetes tem a responsabilidade de coordenar todos os sistemas elétricos e eletrônicos, desde os sensores que monitoram variáveis ambientais e de movimento, até os atuadores que controlam eventos críticos como a abertura de paraquedas. Em foguetes de foguetemodelismo, a integração de componentes eletrônicos e sensores permite não apenas monitorar o voo, mas também automatizar o acionamento de mecanismos de recuperação, essenciais para garantir a segurança do dispositivo e sua reutilização.

O uso de sensores precisos, como barômetros para a medição de altitude e acelerômetros para o monitoramento da aceleração em vários eixos, é essencial para o controle eficaz do foguete. A instrumentação correta e a calibração desses sensores garantem que os dados coletados sejam precisos e úteis para o controle do voo e a realização de simulações e testes de validação.

Além disso, sistemas aviônicos redundantes são frequentemente empregados para garantir que, em caso de falha do sistema principal, outro sistema assuma o controle, minimizando o risco de falhas catastróficas.

1.3. Metodologia adotada

A metodologia de desenvolvimento deste projeto foi estruturada em três principais etapas: seleção de hardware, implementação de software e testes.

1. Seleção de Hardware: A escolha dos componentes de hardware foi baseada nas necessidades operacionais do foguete, buscando-se uma solução economicamente viável e tecnicamente eficiente. Entre os componentes essenciais, foram selecionados sensores barométricos, acelerômetros e giroscópios, com interfaces de comunicação como I2C e SPI para facilitar a integração. A robustez e a confiabilidade dos componentes foram critérios fundamentais para garantir o sucesso do sistema em condições de voo.

2. Desenvolvimento de Software: O software foi desenvolvido em linguagem Arduino, com foco na robustez e precisão. A programação foi orientada para garantir que o sistema aviônico pudesse processar dados de múltiplos sensores em tempo real, ativar eventos no momento correto (como o acionamento de cargas pirotécnicas) e armazenar informações do voo para análises posteriores.

3. Testes e Simulações: A validação do sistema foi realizada por meio de testes unitários dos componentes e simulações numéricas, que permitiram antecipar o comportamento do foguete em diferentes condições de voo. As simulações desempenharam um papel crucial na verificação dos algoritmos e na adequação das rotinas de controle de voo.

2. REQUISITOS DE HARDWARE

A aviônica aplicada ao foguetemodelismo envolve a integração de sistemas eletrônicos que monitoram, controlam e registram dados do voo. No contexto de foguetes de pequeno porte, como os utilizados no foguetemodelismo, a precisão dos dados e o controle sobre eventos críticos, como a recuperação por paraquedas, são essenciais para o sucesso da missão. Assim, sensores como barômetros, acelerômetros e giroscópios desempenham um papel crucial na coleta de informações sobre a trajetória, aceleração e altitude do foguete, fornecendo dados fundamentais para o sistema de controle durante todas as fases do voo.

O desenvolvimento do sistema aviônico foi baseado em uma metodologia que abrange desde a escolha de componentes até o desenvolvimento de software e os testes. Inicialmente, foram analisadas as necessidades do sistema para identificar os componentes mais adequados, levando em consideração desempenho, custo e facilidade de integração. Em seguida, o projeto foi estruturado em fases de implementação de hardware, escrita de software e validação por simulações e testes reais.

Os requisitos de hardware, detalhados a seguir, são cruciais para garantir que os sistemas aviônicos embarcados em foguetes de foguetemodelismo operem com eficiência, segurança e confiabilidade. Eles foram definidos com base nas funcionalidades necessárias para o voo, na coleta de dados, no controle robusto dos eventos de recuperação e no PTR da aviônica da EPTA de 2023.

2.1. Requisitos do Sistema

Fonte de Alimentação: O sistema deve funcionar em corrente contínua (DC). Uma das opções mais comuns e fáceis de encontrar são as baterias de 9V, que são práticas e amplamente disponíveis. Além disso, outra fonte aceitável são os 5V fornecidos ao se conectar o módulo a uma entrada USB de um computador. Utilizar a própria alimentação de 5V do conversor USB-TTL é uma alternativa eficiente para alimentar o sistema, aproveitando a conectividade USB.

Proteção contra Inversão de Polos: A inversão de polaridade pode causar danos severos aos componentes eletrônicos sensíveis do sistema. Para evitar isso, é essencial incluir uma proteção eficaz contra a inversão de polos. A utilização de um diodo Schottky é uma solução eficaz para este requisito, devido à sua baixa queda de tensão direta e resposta rápida,

protegendo assim o sistema de danos causados por polaridade invertida e evitando falhas por conexões equivocadas.

Regulação de Tensão: Para garantir que todos os componentes eletrônicos recebam a tensão correta, o sistema deve ser capaz de regular a tensão de alimentação. Isso é alcançado com o uso de reguladores de tensão que ajustam a tensão de entrada para os níveis apropriados para cada componente. Além disso, capacitores devem ser utilizados em conjunto com os reguladores para estabilizar a tensão e minimizar ruídos, assegurando um fornecimento de energia constante e sem flutuações.

Utilização de Componentes SMD: A miniaturização dos componentes é um aspecto crucial para otimizar o espaço e melhorar a montagem do sistema. A utilização de componentes SMD (Surface-Mount Device) é preferível sempre que possível, exceto para o display e demais componentes da interface homem-máquina, que podem requerer um formato diferente para facilitar a visualização e interação.

2.2. Instrumentação de Sensores

Sensor Barométrico: A capacidade de medir pressão, temperatura e altitude é essencial para a navegação e controle do foguete. O sistema deve ser capaz de instrumentalizar um sensor barométrico, como o ICP10100, que é capaz de aferir esses parâmetros com precisão e já possui proteção contra água. É importante pesquisar e desenvolver o circuito adequado para o funcionamento do sensor em componentes SMD e priorizar sensores que utilizem interfaces I2C ou SPI para comunicação eficiente. O sensor deve operar dentro de um range de pressão de 100 kPa a 57 kPa, cobrindo altitudes de até 5 km acima do nível do mar, no mínimo.

Sensor IMU (Unidade de Medição Inercial): Para monitorar a aceleração em torno dos eixos x, y e z, o sistema deve incluir um sensor IMU adequado, como o MPU6050. Este sensor lê a aceleração linear nos três eixos principais e a velocidade de rotação, fornecendo dados críticos sobre o movimento e a orientação do foguete. Novamente, a pesquisa e desenvolvimento do circuito adequado para o funcionamento do sensor em SMD são essenciais, assim como a priorização de sensores com interface I2C para comunicação eficiente.

2.3. Armazenamento de Dados

Armazenamento Local de Dados: O sistema deve possuir um armazenamento local de dados para garantir a integridade das informações coletadas e permitir uma análise detalhada pós-voo. Utilizar um SDcard, com capacidade mínima de 2 GB, é uma solução prática e eficiente para armazenar os dados de voo. Além disso, é necessário identificar e desenvolver o circuito necessário para o funcionamento correto e leitura de dados no SDcard.

2.4. Sinalização audiovisual

Indicação Visual e Sonora: A capacidade de sinalizar o funcionamento do sistema ao usuário é crucial para monitoramento constante e diagnóstico rápido de problemas. LEDs devem ser utilizados para indicar que o sistema está energizado e para mostrar o modo de funcionamento do sistema de maneira visual, como configurando ou aguardando lançamento. Adicionalmente. Adicionalmente, o uso de um buzzer para indicar o modo de funcionamento do sistema de maneira sonora também é essencial, proporcionando feedback auditivo em diferentes estados operacionais, possibilitando que o usuário identifique o estado do sistema mesmo sem estar olhando diretamente para ele.

2.5. Acionamento de Carga Pirotécnica

Interface para Carga Pirotécnica: O sistema deve ser capaz de acionar cargas pirotécnicas, um aspecto crítico para operações específicas como a abertura de paraquedas. Para isso, a interface deve incluir um conector para acoplar a carga pirotécnica e um circuito capaz de fornecer até 1 A de corrente para acionar a carga, considerando uma fonte de alimentação externa, como uma bateria. Componentes como bornes, transistores e resistores são necessários para desenvolver esse circuito.

2.6. Exportação de Dados

Interface de Exportação: Para análise detalhada e para garantir que as informações coletadas possam ser utilizadas em outros sistemas ou análises posteriores, o sistema deve possuir uma interface que possibilite a exportação de dados. O uso do sdcard para a exportação de dados é uma solução prática. Além disso, é importante projetar um mecanismo de travamento

mecânico para o cartão de memória, protegendo-o de vibrações e remoções acidentais durante o voo.

2.7. Proteção contra Água

Impermeabilização: Para garantir a durabilidade e o funcionamento do sistema em diferentes condições ambientais enfrentadas durante lançamentos, a placa de circuito deve possuir algum nível de proteção contra água. A utilização de um processo de impermeabilização por spray é recomendada para proteger a placa de ambientes úmidos ou molhados, assegurando a operação contínua do sistema mesmo em condições adversas.

2.8. Interface Homem-Máquina (IHM)

Display e Controle: Uma interface homem-máquina eficaz é essencial para a operação e configuração do sistema pelo usuário. Deve-se utilizar um display de pequeno porte, até uma polegada, para apresentar informações essenciais e realizar configurações simples sem a necessidade de mudança no algoritmo de controle do sistema. Utilizar um protocolo de comunicação simples, como I2C, facilita o design do circuito. Além disso, uma chave rotativa com botão pode ser utilizada para operar a IHM de forma intuitiva, permitindo uma interação fácil e eficiente com o sistema.

Modo de Suspensão: Para economizar energia, a IHM deve ser capaz de entrar em modo de suspensão após um certo período de inatividade. Dar preferência a displays que possam ser desligados para reduzir significativamente o consumo de energia é essencial, a utilização de um display OLED é uma solução prática para alcançar esse objetivo.

3. REQUISITOS DE SOFTWARE

Os requisitos de software são essenciais para garantir que o sistema funcione de forma eficiente, precisa e segura. Este capítulo aborda os principais requisitos que o software deve atender para operar o sistema aviônico embarcado, detalhando as funcionalidades e comportamentos necessários. Esses requisitos são fundamentais para assegurar que o sistema não só funcione conforme o esperado, mas também para garantir a segurança e a confiabilidade durante os lançamentos.

3.1. Saída do Modo de Suspensão

O sistema deve ser capaz de sair do modo de suspensão após alguns comandos do usuário. Para isso, a chave rotativa será utilizada para despertar o sistema. Esta funcionalidade é crucial para garantir que a IHM possa ser ativada de forma rápida e eficiente. A capacidade de sair do modo de suspensão rapidamente é importante para operações onde o tempo de resposta é crítico, como em preparações para lançamentos iminentes. Contudo, o sistema deve ser capaz de continuar executando rotinas para detecção de lançamento mesmo em suspensão para garantir o correto funcionamento e evitar resultados catastróficos.

3.2. Modo de Configuração

O sistema deve possuir um modo de configuração que permita definir e/ou exibir parâmetros críticos para a operação do foguete. A interface do display deve exibir um menu claro e intuitivo, proporcionando ao usuário a capacidade de ajustar configurações e visualizar informações importantes. Um modo de configuração bem projetado é essencial para facilitar o uso e a personalização do sistema pelo operador, permitindo ajustes rápidos e precisos antes do lançamento. Dessa maneira, o menu deve incluir as seguintes opções:

- Definir Condição de Acionamento: Permite ao usuário definir as condições para o acionamento de cargas pirotécnicas, podendo ser baseado em porcentagem, altitude ou modo automático (acionamento após o apogeu). Esta flexibilidade é importante para atender diferentes cenários de lançamento e requisitos de missão.
- Tempo para Entrar em Suspensão: Permite ajustar o tempo de inatividade após o qual a interface homem-máquina (IHM) entra em modo de suspensão para economizar

energia. Configurações precisas ajudam a balancear a conservação de energia com a prontidão operacional.

- Exibir Arquivos de Voo: Mostra os arquivos de voo armazenados na memória, permitindo ao usuário validar a criação de arquivos de dados rapidamente. Isso é importante para evitar erros operacionais e testes mal realizados.
- Exibir Dados dos Sensores: Apresenta as leituras dos dados de pressão, temperatura, altitude e aceleração. A disponibilidade desses dados em tempo real ajuda na verificação das condições pré-lançamento e durante o voo.
- Exibir Estatísticas do Lançamento Anterior: Fornece um resumo das estatísticas mais importantes do voo anterior, como altitude máxima, velocidade, entre outros. Esses dados são vitais para validações de desempenho e ajustes em futuras missões.
- Atalho para Modo de Detecção de Lançamento: Permite ao usuário entrar diretamente no modo de detecção de lançamento. Isso facilita a preparação rápida e a prontidão do sistema para lançamentos iminentes.
- Detecção Automática em suspensão: Configura o sistema para entrar automaticamente no modo de detecção de lançamento após a IHM entrar em suspensão. Essa funcionalidade assegura que o sistema esteja sempre pronto para detectar o lançamento, mesmo após períodos de inatividade.

3.3. Detecção de Lançamento

O sistema deve ser capaz de detectar o lançamento do foguete, analisando a variação de altitude dentro de margens de segurança definidas em software. Esta funcionalidade é crítica para a ativação de sequências específicas durante o voo, como a abertura de paraquedas ou outros mecanismos de segurança.

Utilizando dados de sensores barométricos e IMU, o software deve monitorar continuamente a altitude e identificar padrões característicos de um lançamento, ativando as rotinas necessárias conforme programado. O algoritmo de detecção deve ser robusto o suficiente para distinguir entre um lançamento verdadeiro e outras variações de altitude que possam ocorrer devido a condições ambientais ou manuseio do foguete. Adicionalmente, o sistema deve ser capaz de registrar esses eventos para análise posterior e validação do comportamento do software.

3.4. Filtragem de Dados

Para garantir a precisão das leituras e tomadas de decisão, o sistema deve implementar uma filtragem de dados eficaz em dados críticos, como nas leituras de altitude, evitando possíveis leituras ruidosas causadas por condições climáticas adversas ou interferências externas.

A aplicação de algoritmos de filtragem, como filtros Kalman, filtros passa-baixa ou filtro de média móvel devem ser pesquisados e incorporados ao software para suavizar os dados e fornecer leituras confiáveis e estáveis. A escolha do filtro adequado dependerá da natureza dos dados e das condições operacionais esperadas. A implementação desses filtros deve ser otimizada para funcionar em tempo real, garantindo que as leituras filtradas estejam sempre disponíveis para decisões críticas durante o voo.

3.5. Detecção de Desligamento durante Voo

Para garantir a integridade dos dados e a continuidade das operações, o sistema deve criar um arquivo de verificação no sdcard. Ao iniciar um novo voo, o parâmetro de verificação deve ser definido como "false". Após a conclusão do voo, este parâmetro muda para "true". Caso o sistema seja religado e encontre o parâmetro em "false", isso indicará que o lançamento anterior não foi concluído corretamente, ativando um modo de detecção de apogeu ou recuperação baseado nas leituras de altitude.

O software deve incluir rotinas que atualizem e verifiquem o status desse arquivo de verificação, assegurando que as ações apropriadas sejam tomadas em caso de reinicializações inesperadas. Essa funcionalidade é vital para a robustez do sistema, garantindo que qualquer interrupção durante o voo seja detectada e tratada de forma adequada. Além disso, a lógica de detecção deve ser capaz de diferenciar entre um desligamento planejado e um inesperado, adaptando as ações subsequentes conforme necessário.

3.6. Gerenciamento de Arquivos de Dados de Voo

Cada lançamento deve gerar um arquivo de dados específico. O sistema deve ser capaz de ler os arquivos existentes na memória e nomear um novo arquivo de dados corretamente, evitando sobrescrita acidental e mantendo um registro organizado dos voos.

O software deve incluir uma função de gerenciamento de arquivos que crie os arquivos de forma eficiente, assegurando que todos os dados de voo sejam armazenados de maneira clara e acessível. Essa função deve ser capaz de lidar com grandes volumes de dados e garantir que cada arquivo seja etiquetado de forma única, facilitando a recuperação e análise posterior. A estrutura de armazenamento deve ser projetada para permitir acesso rápido e eficiente aos dados, mesmo durante operações de voo críticas.

3.7. Dados de Parâmetros

O sistema deve manter um arquivo de configurações no sdcard, que deve ser importado ao iniciar o sistema. Qualquer alteração nos parâmetros por parte do usuário deve atualizar automaticamente este arquivo, garantindo que as configurações corretas sejam aplicadas em cada sessão.

As rotinas de leitura e escrita de parâmetros devem ser robustas, permitindo que o sistema carregue e salve configurações de forma confiável. O software deve ser capaz de detectar mudanças nos parâmetros e aplicar essas alterações de forma dinâmica, sem a necessidade de reinicializações prolongadas. A implementação deve garantir que qualquer ajuste feito pelo usuário seja imediatamente refletido nas operações do sistema, proporcionando um nível considerável de controle do sobre o sistema.

4. PROJETO DE HARDWARE

O projeto de hardware é uma etapa crucial no desenvolvimento do sistema eletrônico. Antes de iniciar a modelagem propriamente dita, foi necessário pesquisar as etapas adequadas para a produção de uma placa de circuito. Essas etapas incluem a possibilidade de adicionar mais camadas à placa, personalizar a placa, e modificar parâmetros específicos. Um dos principais desafios identificados foi a obtenção de modelos de todos os componentes para realizar o posicionamento correto no projeto.

Após essa fase inicial, realizou-se uma seleção entre os softwares de desenvolvimento de PCB (*Printed Circuit Board* ou Placa de Circuito Impresso). O software escolhido foi o Altium Designer, que oferece uma licença gratuita para estudantes e uma interface completa para o desenvolvimento de placas de circuito. No entanto, o Altium Designer não é de uso intuitivo, exigindo um período de adaptação e treinamento. A própria plataforma do Altium Designer oferece vídeos tutoriais detalhados sobre como utilizar cada recurso do software, tornando o processo de aprendizado eficiente.

Com o curso básico de Altium concluído, foram pesquisados os componentes que atenderiam aos requisitos de hardware. A maioria dos componentes foi escolhida em formato SMD, com alguns em formato PTH, dependendo de sua função. Foram estudados os protocolos de comunicação de cada componente e os circuitos necessários para seu funcionamento, incluindo a alimentação, resistores e capacitores necessários para seu correto desempenho.
Além disso, foi estudada a maneira de desenvolver um sistema de alimentação com um certo grau de segurança. Também foi pesquisado o circuito necessário para o acionamento da carga pirotécnica, o manuseio de um cartão de memória na placa e os componentes necessários para o funcionamento da interface homem-máquina (IHM).

Este capítulo detalha cada uma dessas etapas, desde a escolha dos componentes e ferramentas até a implementação dos circuitos específicos, abordando os desafios encontrados e as soluções adotadas para garantir a funcionalidade e a confiabilidade do sistema aviônico.

4.1. Escolha dos componentes

Para atender a todos os requisitos de hardware, foi realizada uma criteriosa seleção de componentes essenciais para o correto funcionamento da placa de circuito. Este processo teve como objetivo equilibrar economia e desempenho, garantindo que cada componente contribuísse para a eficiência e confiabilidade do sistema sem exceder o orçamento disponível.

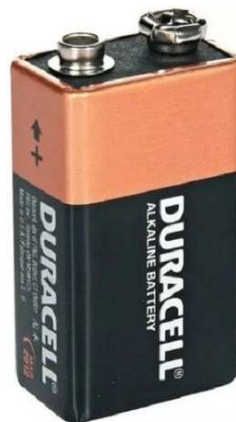
A seguir, são apresentados os principais componentes utilizados no desenvolvimento do circuito, bem como uma breve descrição de suas funções e razões para sua escolha.

- **Fonte de Alimentação**

O sistema deve funcionar em corrente DC. Para garantir uma alimentação estável, foram escolhidas duas fontes principais:

- **Bateria de 9V**: Uma fonte de energia fácil de encontrar e capaz de fornecer a tensão necessária para o sistema (Imagem 13).
- **Conversor USB-TTL**: Utiliza a alimentação de 5V disponível ao conectar o módulo a uma entrada USB de algum computador, garantindo uma alternativa prática e eficiente (Imagem 14).

Imagem 13 – Exemplo de bateria de 9V.



Fonte: Duracell bateria 9v [03 de agosto de2024]. Disponível em: www.amazon.com.

Imagem 14 – Conversor USB-TTL para carregar o código na placa.

Fonte: Módulo Conversor Usb/Serial TTL PL2303 [03 de agosto de 2024]. Disponível em: www.byteflop.com.br.

- **Proteção contra Inversão de Polos**

Para proteger o sistema contra a inversão de polos, foi utilizado um Diodo Schottky 1N5822 (Imagem 15). Este componente é eficiente para impedir que uma conexão incorreta cause danos ao circuito.

Imagem 15 – Diodo Schottky, 1N5822, 3A, 40V.

Fonte: Diodo Schottky 1N5822 40V 3A [03 de agosto de 2024]. Disponível em: www.baudaeletronica.com.br.

- **Regulação de Tensão**

A eletrônica do sistema deve ser capaz de regular a tensão de alimentação. Para isso, foram utilizados:

- **Reguladores de Tensão**: Adequam a tensão de entrada para níveis apropriados para os diferentes componentes do sistema, foram escolhidos reguladores da família LM1117 no formato SMD (Imagem 16).

Imagem 16 – Regulador de tensão da família LM1117 SMD.

Fonte: Regulador de tensão AMS1117 5V 1ª [03 de agosto de 2024]. Disponível em: www.eletrogate.com.

- **Capacitores**: Seguem os reguladores de tensão para ajudar a estabilizar a tensão e reduzir ruídos elétricos.

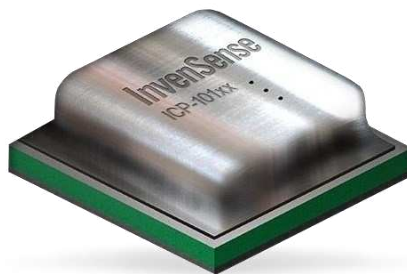
- **Componentes SMD**

Sempre que possível, foram utilizados componentes SMD (Surface-Mount Device) devido à sua compactação e facilidade de montagem automatizada. Exceções foram feitas para o display da interface homem-máquina (IHM), que requer um formato diferente devido à sua função específica.

- **Sensores Barométricos e Acelerômetros**

  - **Sensor Barométrico (ICP10100)**: Este sensor foi escolhido por sua capacidade de aferir pressão, temperatura e altitude com precisão. Ele opera dentro de uma faixa de pressão de 100 kPa a 57 kPa (até 5 km acima do nível do mar) e utiliza interface I2C ou SPI para comunicação, conforme o datasheet fornecido pela *TDK CORPORATION* (2019), facilitando a integração com o sistema (Imagem 17).

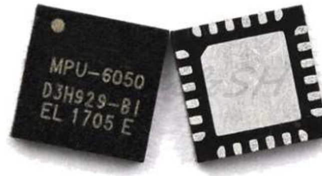
Imagem 17 – Ilustração de um sensor barométrico ICP10100 SMD.



Fonte: TDK InvenSense ICP-10100 [03 de agosto de 2024]. Disponível em: https://br.mouser.com.

  - **Sensor IMU (MPU6050)**: Selecionado para medir a aceleração linear e velocidade de rotação nos eixos x, y e z, com um fundo de escala de até 16g. Este sensor também utiliza interface I2C, conforme o datasheet fornecido pela *TDK CORPORATION* (2013), garantindo uma comunicação eficiente com o microcontrolador (Imagem 18).

Imagem 18 – Ilustração de um sensor IMU MPU6050 SMD.



Fonte: TDK InvenSense MPU-6050 [03 de agosto de 2024]. Disponível em: https://br.mouser.com.

- **Armazenamento Local de Dados**

Para o armazenamento de dados, foi escolhido um sdcard com capacidade mínima de 2GB (Imagem 19). Este componente é crucial para registrar informações durante o voo, garantindo que os dados coletados estejam seguros e sejam facilmente acessíveis para análise posterior. Também foi escolhido um modelo de socket com trava para dar mais segurança e proteger o sdcard contra remoções acidentais devido a vibração do foguete.

Imagem 19 – Modelo de Sdcard 2 GB.

Fonte: Sandisk microSD 2GB memory card [03 de agosto de 2024]. Disponível em: www.amazon.com.

- **Sinalização ao Usuário**

Para sinalizar o funcionamento do sistema ao usuário, foram utilizados:

- **LEDs**: Indicadores visuais (Imagem 20) para mostrar que o sistema está energizado, bem como para indicar diferentes modos de operação (configuração, aguardando lançamento, em voo).

- **Buzzer**: Proporciona sinalização sonora (Imagem 21) para diferentes estados do sistema, ajudando o usuário a identificar rapidamente o status atual.

Imagem 20 – Leds SMD.

Fonte: Wurth Elektronik 150120BS75000 [03 de agosto de 2024]. Disponível em: https://br.mouser.com/.

Imagem 21 - Buzzer

Fonte: CUI Devices CMT-1209-1290T [03 de agosto de 2024]. Disponível em: https://br.mouser.com/.

- **Acionamento de Carga Pirotécnica**

O acionamento seguro de cargas pirotécnicas é garantido por um circuito específico composto por:

- **Borne**: Para a conexão da carga pirotécnica (Imagem 22).

- **Transistor e Resistor**: Componentes que fornecem a corrente necessária (1A) para acionar a carga, considerando uma fonte de alimentação externa (imagem 23).

Imagem 22 – Borne modelo KF301-2P.

Fonte: Borne Para PCI KRE2 KF301 2 Pinos [03 de agosto de 2024]. Disponível em: www.eletropartscomponentes.com.br.

Imagem 23 – Transistor 2N2222.

Fonte: Transistor NPN 2N2222 [03 de agosto de 2024]. Disponível em: www.eletrogate.com.

- **Exportação de Dados**

Para a exportação de dados, é utilizado o sdcard, que facilita a transferência das informações coletadas. Além disso, foi escolhido um soquete para travamento mecânico do cartão de memória (Imagem 24), protegendo-o contra vibrações e remoções acidentais.

Imagem 24 – Socket para Sdcard modelo 47219-2001.

Fonte: Molex 47219-2001 [03 de agosto de 2024]. Disponível em: https://br.mouser.com/.

- **Proteção contra Água**

A placa de circuito deve ter algum tipo de proteção contra água e umidade, então é sugerido um processo de impermeabilização por spray, garantindo a durabilidade e a operação segura em ambientes adversos. É importante ressaltar que alguns componentes não podem ser completamente impermeabilizados, como o socket do sdcard, a saída de áudio do buzzer, pinos de conexão do conversor USB-TTL e as conexões dos bornes utilizados.

- **Interface Homem-Máquina (IHM)**

A IHM é composta por:

- **Display de Pequeno Porte**: Com até uma polegada, utilizado para exibir informações de forma clara e concisa. Foi escolhido o modelo de display oled de 0.96 polegadas com controlador SSD1306 (Imagem 25). Esse display conta com interface i2c, o que facilita a conexão.

- **Chave Rotativa com Botão**: Permite a navegação pelos menus e a configuração dos parâmetros do sistema. Foi escolhido um modelo genérico com posições discretas obtidas a partir de um encoder integrado, contando com um botão para comandos (Imagem 26).

Imagem 25 – Display Oled SSD1306.

Fonte: RoboCore [03 de agosto de 2024]. Disponível em: www.robocore.net.

Imagem 26 – Chave rotativa com botão e encoder modelo 01018.

Fonte: Aliexpress [03 de agosto de 2024]. Disponível em: https://pt.aliexpress.com/.

- **Modo de Suspensão da IHM**

Para economizar energia, a IHM entra em modo de suspensão após um certo tempo ocioso. São preferidos displays que possam ser desligados ou colocados em modo de baixo consumo, alimentados por algum pino do microcontrolador ou utilizando um display OLED.

- **Saída do Modo de Suspensão**

O sistema deve ser capaz de sair do modo de suspensão após comandos do usuário, utilizando a chave rotativa com botão da IHM.

- **Escolha do processador do sistema**

Para selecionar o processador adequado ao projeto, diversos requisitos técnicos foram considerados. O processador deveria possuir um número suficiente de GPIOs (*General-Purpose Input/Output*) para gerenciar múltiplos sensores e atuadores conectados ao sistema. Além disso, era essencial que o processador suportasse os protocolos de comunicação I2C e SPI, fundamentais para a interface com sensores barométricos e de aceleração, bem como para a integração de periféricos adicionais, como o cartão SD e o display da interface homem-máquina. A capacidade de memória também foi um critério crucial para comportar o sistema embarcado e todas as funcionalidades propostas.

Inicialmente, a escolha do processador recaiu sobre o microcontrolador da família Arduino, o ATmega328P. Este microcontrolador é amplamente utilizado em projetos de prototipagem e possui uma vasta comunidade de suporte, o que facilita o desenvolvimento e a resolução de problemas. No entanto, durante os testes iniciais, verificou-se que o ATmega328P não possuía memória RAM suficiente para executar todas as funcionalidades de software exigidas pelo projeto. Esta limitação impediu o desempenho adequado das tarefas, levando à necessidade de reconsiderar a escolha do processador.

Diante dessa limitação, a plataforma ESP32 (Imagem 27) foi selecionada como substituta. O ESP32 oferece uma capacidade significativamente maior de memória RAM, além de contar com um processador dual-core e conectividade Wi-Fi e Bluetooth integradas (*ESPRESSIF SYSTEMS*, 2024), conectividades essas que poderiam ser implementadas em recursos futuros. Essas características fazem do ESP32 uma solução ideal para aplicações que requerem processamento intensivo e comunicação sem fio. A escolha do ESP32 não só resolveu o problema de memória, como também agregou funcionalidades adicionais ao sistema, aumentando sua versatilidade e capacidade de integração.

Outro fator decisivo na escolha do ESP32 foi a compatibilidade com o IDE do Arduino, uma plataforma amplamente utilizada e suportada por uma vasta comunidade de desenvolvedores e entusiastas. A utilização do Arduino IDE facilita o desenvolvimento de software embarcado, oferecendo uma interface simples e acessível para programar o ESP32. Além disso, a extensa documentação e bibliotecas disponíveis para o Arduino proporcionam uma base sólida para implementação de funcionalidades específicas, além de acelerar o processo de depuração e testes. Esse suporte da comunidade garante que o desenvolvimento possa contar com recursos prontos, tutoriais e fóruns de discussão, tornando o ESP32 uma escolha versátil e prática para o projeto.

Imagem 27 – Processador da plataforma esp32.

Fonte: Mouser [03 de agosto de 2024]. Disponível em: https://br.mouser.com/.

Abaixo, é apresentado a Tabela 1, resumindo os requisitos de hardware citados anteriormente e atribuindo os componentes escolhidos para atendê-los:

Tabela 1 – Resumo dos requisitos do projeto.

Requisitos do projeto					
Tipo	**Índice**	**Requisito**	**Subíndice**	**Especificações**	**Componente**
Hardware	1	O sistema deve funcionar em corrente DC;	1.1	Uma fonte com certa facilidade de ser encontrada são baterias de 9 V;	Bateria 9V
			1.2	Outra fonte aceitável são os 5 V disponíveis ao se conectar o módulo a uma estrada USB de algum computador.	Utilizar a própria alimentação de 5V do conversor usb-ttl para alimentar o sistema.
	2	O sistema deve possuir uma proteção contra a inversão de polos;	2.1	Um diodo se mostra suficiente para cumprir esse requisito.	Diodo *shottky*
	3	A eletrônica do sistema deve ser capaz de regular a tensão de alimentação;	3.1	Deve ser utilizado reguladores de tensão para adequar a tensão aos diferentes componentes;	Regulador de tensão
			3.2	Cada regulador deve ser seguido de capacitores adequados para ajudar a estabilizar a tensão.	Capacitores
	4	Utilização de componentes em SMD, tanto quanto possível;	4.1	A maioria dos componentes deve ser SMD, exceto o display para interface homem máquina.	
	5	O sistema deve ser capaz de instrumentalizar um	5.1	Pesquisar um sensor barométrico capaz de aferir pressão, temperatura e altitude;	ICP10100

		sensor que afira pressão, temperatura e altitude;	5.2	Pesquisar o circuito adequado para o funcionamento do sensor em componentes SMD;	
			5.3	Priorizar um sensor que utilize interface i2c ou SPI.	
			5.4	A pressão deve variar num range de 100 kPa à 57 kPa (até 5 km acima do nível do mar)	
	6	O sistema deve ser capaz de instrumentalizar um sensor que leia aceleração em tornos dos eixos x, y e z;	6.1	Pesquisar um sensor IMU adequado;	MPU6050
			6.2	Pesquisar o circuito adequado para o funcionamento do sensor em componentes SMD;	
			6.3	Priorizar um sensor que utilize interface i2c.	
			6.4	Fundo de escala: 16g	
	7	Deve possuir um armazenamento local de dados;	7.1	Utilizar um sdcard para armazenamento de dados;	SDcard 2Gb (mínimo)
			7.2	Identificar o circuito necessário para o funcionamento correto e leitura de dados.	
	8	O sistema deve ser capaz de sinalizar seu funcionamento ao usuário;	8.1	Utilizar leds para indicar que o sistema está energizado;	Led
			8.2	Utilizar leds para indicar o modo de funcionamento do sistema de maneira visual (configurando, aguardando lançamento, em voo);	
			8.3	Inserir um led em paralelo com o sdcard para verificar se dados estão sendo lidos\gravados.	
			8.4	Utilizar um buzzer para indicar o modo de funcionamento do sistema de maneira sonora (configurando, aguardando lançamento, em voo).	*Buzzer*
	9	O sistema deve ser capaz de acionar uma carga pirotécnica;	9.1	A interface deve contemplar algum conector para se acoplar uma carga pirotécnica;	*Borne*
			9.2	Deve ser inserido um circuito capaz de fornecer 1 A de corrente para acionar a carga, levando em consideração uma fonte de alimentação externa.	Transistor e resistor
	10	O sistema eletrônico deve possuir alguma	10.1	Utilizar o sdcard para a exportação de dados.	

		interface que possibilite exportação de dados;	10.2	Pesquisar e projetar um mecanismo para travamento mecânico do sdcard. Protegendo-o assim de vibrações e remoções acidentais.	
	11	A placa de circuito deve possuir alguma proteção contra água	11.1	Utilizar um processo de impermeabilização por spray para proteger a placa de ambientes	
	12	O sistema deve possuir uma interface homem máquina para ser operada pelo usuário;	12.1	Deve ser utilizado um display de pequeno porte (até uma polegada);	
			12.2	Utilizar um protocolo de comunicação que facilite o circuito, como a comunicação i2c;	
			12.3	Utilizar alguma chave rotativa com botão para operar a IHM.	
	13	A IHM deve entrar em modo de suspensão após um certo tempo ocioso;	13.1	Dar preferência à displays que possam ser desligados ou algo próximo a isso, para economizar energia;	
			13.2	Alimentar o display a partir de algum pino do microcontrolador ou utilizar um display OLED.	
	14	O sistema deve contar com um processador capaz de suportar todas as funcionalidades do sistema.	14.1	O processador deve possuir uma quantidade suficiente de GPIO's para conexões;	ESP32
			14.2	Deve suportar os protocolos de comunicação I2C e SPI;	
			14.3	Pesquisar o circuito fundamental para o seu funcionamento	

Fonte: O Autor.

4.2. Comunicação I2C

4.2.1. História e Funcionamento do Protocolo I2C

O protocolo I2C (*Inter-Integrated Circuit*) foi desenvolvido pela *Philips Semiconductor* (agora *NXP Semiconductors*) no início dos anos 1980 para facilitar a comunicação entre diversos componentes eletrônicos dentro de um sistema. Inicialmente, o objetivo era criar um meio simples e eficiente de comunicação entre microcontroladores e periféricos como sensores, displays e outros dispositivos, utilizando apenas dois fios de conexão. Desde sua criação, o I2C

tornou-se um dos protocolos de comunicação mais amplamente utilizados em eletrônica devido à sua simplicidade, flexibilidade e eficiência (*NPX Semiconductors*, 2021).

O I2C utiliza dois fios para comunicação (*NPX Semiconductors*, 2021): o SDA (*Serial Data Line*) e o SCL (*Serial Clock Line*). O protocolo é baseado em um sistema mestre-escravo, onde um dispositivo mestre controla o relógio e inicia a comunicação com os dispositivos escravos (*Texas Instruments*, 2015). Cada dispositivo conectado ao barramento I2C possui um endereço único, permitindo ao mestre identificar e comunicar-se com cada escravo individualmente. Essa estrutura simplificada reduz a quantidade de cabos necessários, facilitando a interconexão de múltiplos componentes em um mesmo sistema.

4.2.2. Funcionamento Básico

O funcionamento do I2C (Imagem 28) pode ser dividido em algumas etapas principais, que são fundamentais para a compreensão do protocolo (*NPX Semiconductors*, 2021):

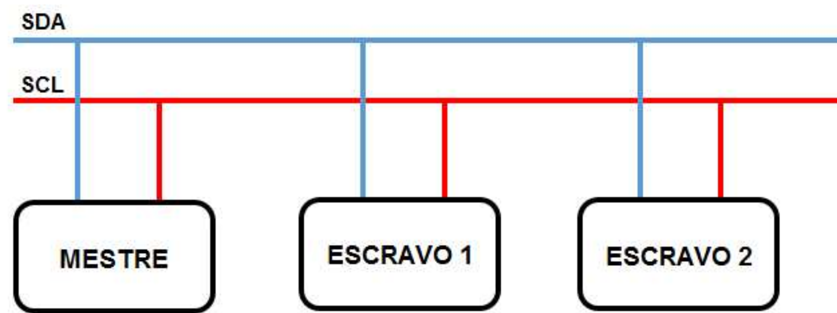
Início da Comunicação: O dispositivo mestre gera um sinal de início (*start condition*) puxando a linha SDA de alta para baixa enquanto a linha SCL permanece alta. Este sinal indica aos dispositivos escravos que uma transmissão de dados está prestes a começar. Este é um comando crucial que sincroniza todos os dispositivos no barramento (Imagem 29).

Endereçamento: O mestre envia um byte de endereço seguido de um bit de leitura/escrita. Cada escravo verifica se o endereço corresponde ao seu próprio. O escravo correspondente responde com um sinal de reconhecimento (ACK). Este mecanismo permite ao mestre identificar com qual escravo ele está se comunicando, evitando colisões de dados (Imagem 30).

Transmissão de Dados: Os dados são transmitidos byte a byte. Após cada byte, o receptor envia um ACK para confirmar a recepção bem-sucedida. Isso garante a integridade da comunicação, permitindo que o mestre saiba que os dados foram recebidos corretamente.

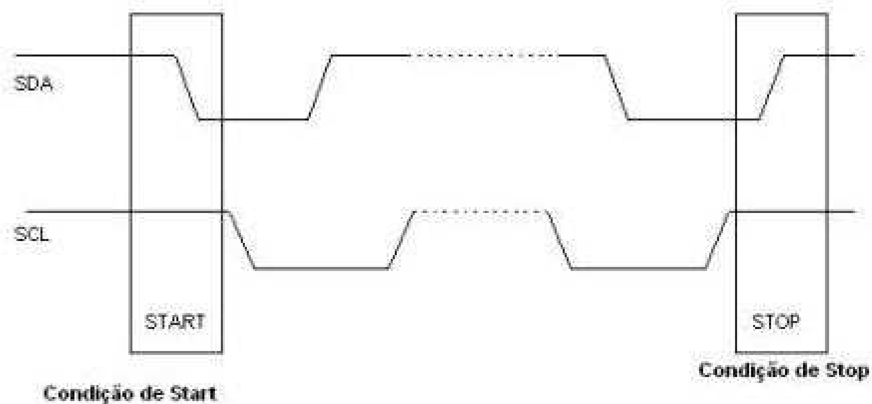
Fim da Comunicação: O mestre gera um sinal de parada (*stop condition*) puxando a linha SDA de baixa para alta enquanto a linha SCL está alta, indicando o fim da transmissão. Este comando finaliza a sessão de comunicação, permitindo que os dispositivos saibam que a transmissão está completa.

Imagem 28 – Conexão básica para a comunicação no protocolo I2C.



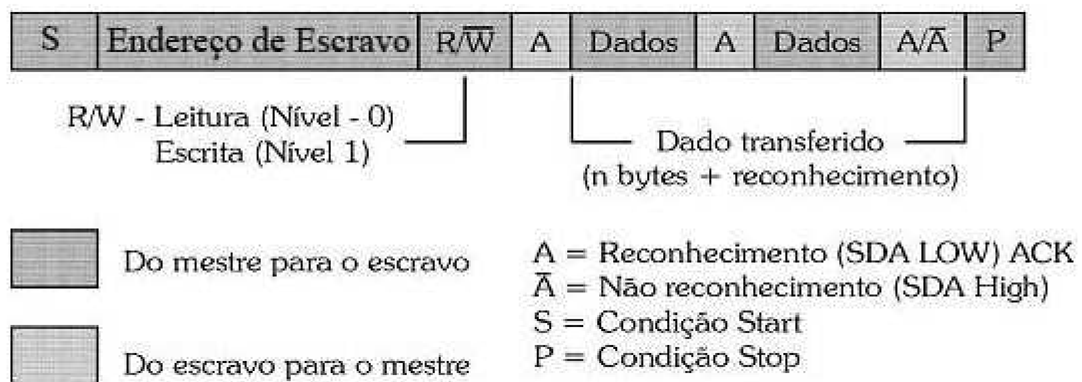
Fonte: Protocolo I2C Comunicação entre Arduinos [0e de agosto de 2024]. Disponível em: https://portal.vidadesilicio.com.br/.

Imagem 29 – Condições de início e parada da comunicação.



Fonte: Microcontrolandos [03 de agosto de 2024]. Disponível em: https://microcontrolandos.blogspot.com/.

Imagem 30 – Formato da transmissão I2C.



Fonte: Microcontrolandos [03 de agosto de 2024]. Disponível em: https://microcontrolandos.blogspot.com/.

4.2.3. Vantagens e Desvantagens

Vantagens:

Segundo a *NPX Semiconductors*, as vantagens desse protocolo de comunicação são, em resumo:

Simplicidade: Apenas dois fios são necessários para conectar múltiplos dispositivos, o que reduz a complexidade do hardware.

Flexibilidade: Permite comunicação entre dispositivos de diferentes fabricantes, tornando o protocolo versátil e amplamente aplicável.

Escalabilidade: Suporta múltiplos mestres e até 128 dispositivos em um único barramento (teoricamente mais, dependendo do endereçamento). Isso torna o I2C adequado para sistemas complexos com muitos periféricos.

Desvantagens:

Velocidade: A taxa de transferência é relativamente baixa comparada a outros protocolos como SPI. Isso pode ser uma limitação em aplicações que requerem alta velocidade de comunicação.

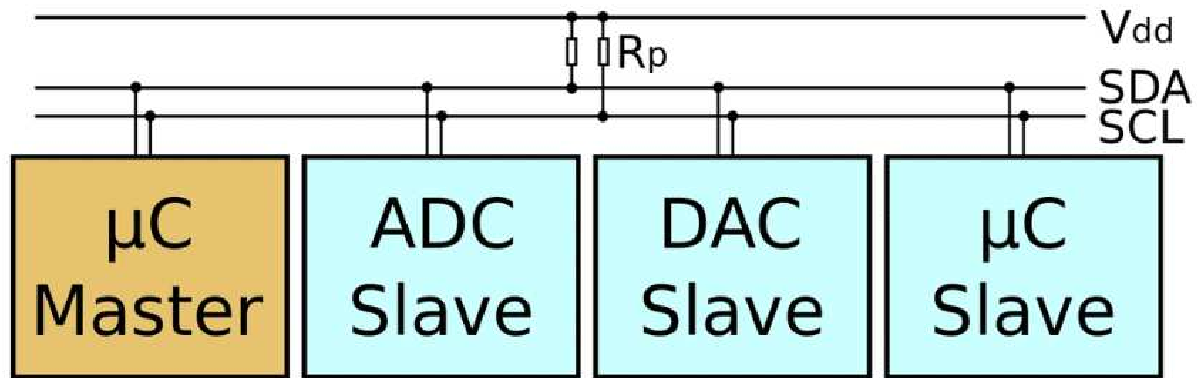
Distância: O comprimento dos cabos deve ser limitado devido à capacitância das linhas que afeta a velocidade e integridade do sinal. Em sistemas onde os dispositivos estão fisicamente distantes, o I2C pode não ser a melhor escolha.

4.2.4. Conexão *Pull-Up*

No barramento I2C, as linhas SDA e SCL são mantidas em um estado de alta impedância quando não estão sendo ativamente acionadas pelos dispositivos. Para garantir que as linhas retornem ao estado alto corretamente, são utilizados resistores *pull-up* (Imagem 31). Estes resistores conectam as linhas SDA e SCL à fonte de alimentação, garantindo que, na ausência de um sinal ativo, as linhas mantenham um nível lógico alto (EMBARCADOS, 2023).

O valor típico dos resistores *pull-up* pode variar entre 1 kΩ e 10 kΩ, dependendo da capacitância total do barramento e da velocidade de operação desejada. Resistor *pull-up* inadequados podem levar a problemas de comunicação, como sinais de relógio distorcidos ou falhas na detecção de *start* e *stop conditions*. A escolha adequada dos resistores *pull-up* é crucial para a operação estável e confiável do barramento I2C (EMBARCADOS, 2023).

Imagem 31 – Conexão I2C em *Pull-Up*.



Fonte: Embarcados [03 de agosto de 2024]. Disponível em: Comunicação I2C - Embarcados - Sua fonte de informações sobre Sistemas Embarcados.

4.2.5. Componentes e Aplicações

Os sensores barométricos, acelerômetros e outros periféricos utilizados em projetos de hardware geralmente possuem interfaces I2C, permitindo facilmente integração e comunicação eficiente. A escolha de componentes com interface I2C facilita a interconexão e o gerenciamento de múltiplos dispositivos, reduzindo a complexidade do circuito. No projeto em questão, sensores como o ICP10100 para medir pressão, temperatura e altitude, e o MPU6050 para medir aceleração em torno dos eixos x, y e z e o display OLED utilizam o protocolo I2C para comunicação com o microcontrolador.

4.3. Barramento de comunicação SPI

4.3.1. História e funcionamento do protocolo SPI

O protocolo SPI (*Serial Peripheral Interface*) foi desenvolvido pela Motorola nos anos 1980, destinado a proporcionar uma comunicação síncrona de alta velocidade entre microcontroladores e periféricos. Desde sua criação, o SPI tem sido amplamente adotado devido à sua simplicidade, flexibilidade e desempenho superior em termos de velocidade. O protocolo é especialmente útil em aplicações onde a alta taxa de transferência de dados é crucial, como em comunicação com memórias flash, sensores, e displays de alta resolução.

A principal motivação para o desenvolvimento do SPI foi a necessidade de um protocolo que permitisse a rápida troca de dados entre um microcontrolador e seus periféricos, sem a complexidade associada a outras soluções de comunicação da época. A sua aceitação e uso

contínuos ao longo das décadas demonstram sua eficácia em atender a essa necessidade, destacando-se em várias aplicações industriais e de consumo (WIKIPEDIA, 2023).

4.3.2. Funcionamento Básico

O SPI utiliza quatro linhas principais para comunicação (*TEXAS INSTRUMENTS*, 2012):

MISO (*Master In Slave Out*): Linha pela qual o escravo envia dados ao mestre.

MOSI (*Master Out Slave In*): Linha pela qual o mestre envia dados ao escravo.

SCK (*Serial Clock*): Linha de relógio gerada pelo mestre para sincronizar a transferência de dados.

SS (*Slave Select*): Linha utilizada pelo mestre para selecionar qual escravo está ativo na comunicação.

A comunicação SPI é *full-duplex*, o que significa que os dados podem ser enviados e recebidos simultaneamente. Isso é realizado por meio de um sistema mestre-escravo, onde o mestre controla o relógio e as linhas de dados, enquanto os escravos respondem aos comandos do mestre. Cada escravo é selecionado individualmente pelo mestre utilizando a linha SS, permitindo a comunicação com múltiplos dispositivos no mesmo barramento (*TEXAS INSTRUMENTS*, 2012).

A arquitetura de SPI é relativamente simples em comparação com outros protocolos. Não requer endereçamento complexo dos dispositivos escravos, como ocorre no I2C, o que facilita o design de sistemas de comunicação rápidos e eficientes. No entanto, a simplicidade do SPI também traz algumas limitações, como a necessidade de linhas de controle adicionais para cada dispositivo escravo, o que pode ser um desafio em sistemas com muitos periféricos (*TEXAS INSTRUMENTS*, 2012).

As informações a seguir foram inferidas a partir do datasheet fornecido pela *TEXAS INSTRUMENTS* (2012).

4.3.3. Etapas da comunicação SPI

1. Início da Comunicação: O mestre seleciona o escravo apropriado ativando a linha SS (normalmente puxando-a para baixo). Isso habilita o escravo selecionado a participar da comunicação. Este passo é crucial, pois garante que apenas o dispositivo desejado responderá aos comandos do mestre.
2. Transmissão de Dados: O mestre gera o sinal de relógio na linha SCK e envia dados pela linha MOSI. Simultaneamente, o escravo envia dados pela linha MISO. Cada bit é transferido em cada ciclo do relógio, permitindo uma troca rápida de informações. A velocidade do relógio é configurada pelo mestre, que pode ajustá-la conforme as capacidades dos escravos para assegurar uma comunicação estável.

3. Fim da Comunicação: Após a transferência dos dados, o mestre desativa a linha SS (normalmente puxando-a para cima), sinalizando o fim da comunicação com o escravo. O escravo então desabilita sua linha de dados, aguardando a próxima comunicação. Este passo é importante para garantir que a linha de comunicação fique disponível para outros dispositivos, prevenindo interferências.

4.3.4. Vantagens e Desvantagens

Vantagens:

- Alta Velocidade: A taxa de transferência de dados é significativamente mais alta em comparação com protocolos como I2C, tornando-o ideal para aplicações que exigem rápida comunicação. O SPI pode operar em frequências de relógio muito elevadas, dependendo das características dos dispositivos conectados.
- Simplicidade de Implementação: O SPI é relativamente fácil de implementar em hardware e software, com uma estrutura simples que facilita a depuração. Esta simplicidade se traduz em menor overhead de protocolo e maior eficiência de comunicação.
- Flexibilidade: Suporta múltiplos dispositivos no mesmo barramento com a capacidade de comunicação full-duplex. A capacidade de conectar diversos dispositivos permite a criação de sistemas complexos com múltiplos sensores e atuadores.

Desvantagens:

- Número de Pinos: Requer mais pinos de I/O em comparação com o I2C, o que pode ser uma limitação em sistemas com recursos limitados. Em microcontroladores com um número reduzido de pinos, essa necessidade adicional pode ser um fator restritivo.
- Complexidade em Múltiplos Escravos: Embora suporte múltiplos escravos, a necessidade de uma linha SS separada para cada escravo pode complicar o design em sistemas com muitos dispositivos. Cada dispositivo adicional aumenta a complexidade do circuito de controle.

4.3.5. Aplicações e Componentes

O SPI é amplamente utilizado em diversas aplicações, desde comunicação com memórias flash e cartões SD até sensores de alta velocidade e displays. No projeto em questão, o SPI pode ser utilizado para a comunicação com o cartão de memória, permitindo uma rápida leitura e gravação de dados de voo. Além disso, sensores e outros periféricos que requerem alta taxa de transferência podem se beneficiar do uso do SPI.

4.3.6. Configuração Pull-Up no protocolo SPI

No contexto do SPI, a configuração pull-up não é tão comum quanto no I2C, mas pode ser usada em linhas SS para garantir que elas fiquem em um estado conhecido quando não estão sendo ativamente controladas pelo mestre. Adicionar resistores pull-up às linhas SS pode prevenir estados flutuantes, que poderiam levar a comportamentos imprevisíveis nos dispositivos escravos.

4.4. Modelagem da placa de circuitos através do Altium Designer

A modelagem da placa de circuitos é uma etapa crítica no desenvolvimento de qualquer sistema eletrônico. A escolha da ferramenta adequada para este processo pode influenciar significativamente a eficiência do design, a facilidade de manufatura e a qualidade final do produto. O Altium Designer é uma das plataformas mais respeitadas e amplamente utilizadas para o design de placas de circuito impresso (PCB), oferecendo uma gama de funcionalidades que facilitam cada etapa do processo de desenvolvimento. Além disso, toda a instrução necessária sobre a utilização desse software está disponível nas plataformas online destinadas

a ensino do mesmo (disponível em: https://www.altium.com/education). Esta seção discutirá as vantagens de utilizar o Altium Designer para a modelagem da placa de circuitos no projeto.

Uma das principais vantagens do Altium Designer é sua interface integrada e intuitiva. A plataforma permite que os engenheiros realizem todas as etapas do design em um único ambiente, desde a captura esquemática até o layout da PCB, simulação e verificação. Esta integração melhora a eficiência do fluxo de trabalho e reduz a possibilidade de erros que podem ocorrer ao alternar entre diferentes ferramentas.

O Altium Designer também se destaca pela sua ampla e personalizável biblioteca de componentes. Esta biblioteca inclui modelos detalhados de uma vasta gama de componentes eletrônicos, além de permitir que os usuários criem e personalizem seus próprios componentes. Essa flexibilidade é crucial para projetos que envolvem componentes específicos ou customizados, garantindo que todos os elementos necessários para o projeto estejam disponíveis.

Outro ponto forte do Altium Designer são suas ferramentas avançadas de roteamento. A plataforma oferece funcionalidades como roteamento automático, controle de impedância e ajuste de comprimento de trilhas, que ajudam os engenheiros a otimizar o layout das trilhas na PCB. Essas ferramentas são essenciais para criar designs que atendam aos requisitos de integridade de sinal e desempenho elétrico, especialmente em projetos complexos.

Antes de proceder com a fabricação da PCB, é essencial verificar que o design funcionará conforme esperado. O Altium Designer oferece ferramentas de simulação e verificação integradas, como análise de integridade de sinal e verificação de regras de design (DRC). Essas ferramentas permitem identificar e corrigir potenciais problemas no estágio de design, economizando tempo e recursos que seriam gastos em retrabalhos após a fabricação.

O suporte a designs multicamadas é outra vantagem significativa do Altium Designer. A plataforma permite a criação de PCBs multicamadas, que são essenciais para projetos avançados que necessitam de alta densidade de componentes e trilhas. Esta capacidade permite a criação de circuitos mais compactos e complexos, otimizando o uso do espaço disponível e melhorando o desempenho elétrico.

Além disso, o Altium Designer facilita a colaboração entre equipes de design. A plataforma permite que múltiplos engenheiros trabalhem simultaneamente no mesmo projeto, utilizando ferramentas de gerenciamento de versões e controle de acesso que garantem que as alterações sejam rastreadas e gerenciadas de maneira eficiente. Isso minimiza conflitos e melhora a coordenação entre os membros da equipe.

A documentação precisa e detalhada gerada pelo Altium Designer é fundamental para a fabricação e montagem da PCB. A plataforma gera automaticamente toda a documentação necessária, incluindo diagramas esquemáticos, listas de materiais (BOM), arquivos Gerber e instruções de montagem. Esta documentação facilita a comunicação com os fabricantes e garante que a PCB seja produzida corretamente.

No contexto do projeto em questão, as vantagens do Altium Designer serão aproveitadas para modelar a placa de circuitos que integra diversos componentes críticos, como sensores, módulos de comunicação e sistemas de alimentação. A interface intuitiva e as ferramentas de roteamento avançadas serão particularmente úteis para garantir que o design da PCB atenda aos requisitos de desempenho e confiabilidade.

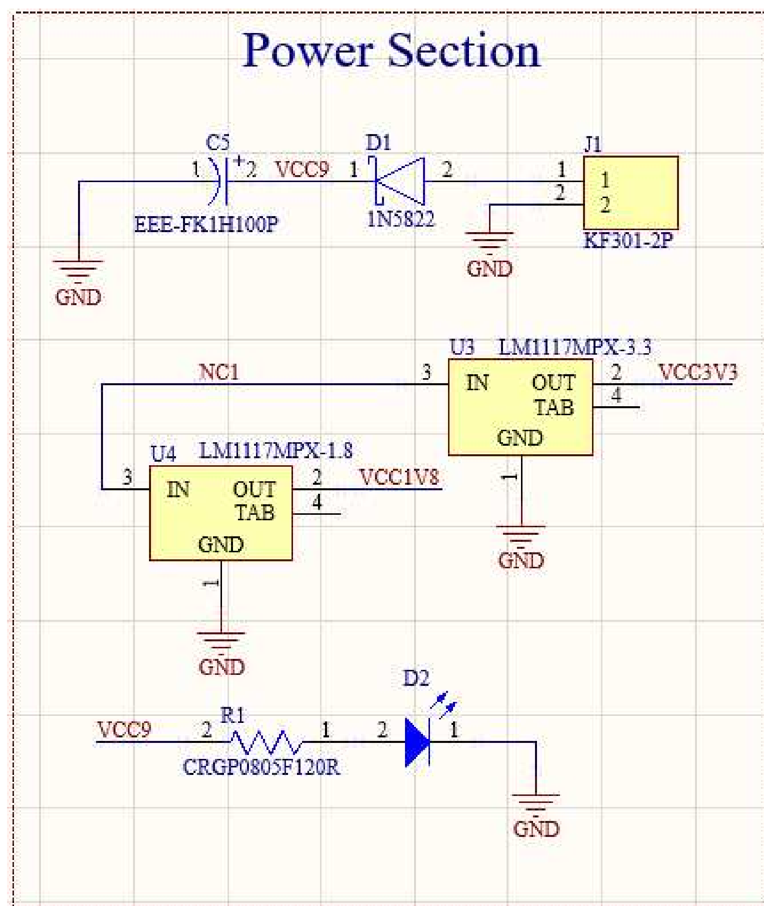
4.4.1. Circuito de alimentação do sistema

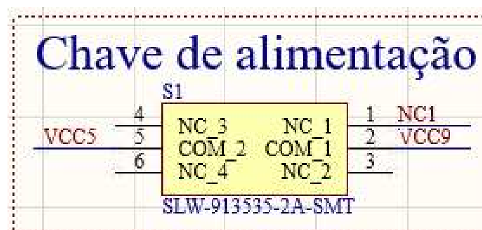
O circuito de alimentação (Imagem 32) possui duas partes principais. A primeira é um pequeno interruptor que permite selecionar a fonte de energia, que pode ser proveniente do conversor USB-TTL ou das baterias. É importante não utilizar ambas as fontes simultaneamente e evitar usar a energia do conversor para testes de acionamento, pois isso pode danificar a porta USB do computador conectado. Vale ressaltar que o conversor será utilizado apenas para carregar o software na placa e para possíveis atualizações de firmware. Após o carregamento do código, a placa deve operar exclusivamente com a bateria.

Quando a alimentação é proveniente da bateria, esta deve ser conectada aos bornes, e o interruptor deve estar comutado para a posição da bateria. A alimentação passa então por um diodo Schottky para evitar a inversão de polos e está em paralelo com um capacitor eletrolítico, proporcionando uma tensão estável de 9 volts, adequada para aplicações na placa.

Por outro lado, a alimentação proveniente do conversor USB-TTL, por já ser regulada pelo computador, não possui todos esses componentes auxiliares. O usuário deve garantir a correta ligação nos pinos destinados ao conversor. O interruptor conecta ambas as fontes a um barramento que está ligado a dois reguladores de tensão, um de 3,3 volts e outro de 1,8 volts. A maioria dos componentes opera com a primeira tensão, enquanto o barômetro opera com a segunda. Além disso, há um pequeno LED SMD para indicar que o sistema está energizado.

Imagem 32 – Circuito responsável pela alimentação do sistema.





Fonte: O Autor.

4.4.2. Circuito do processador

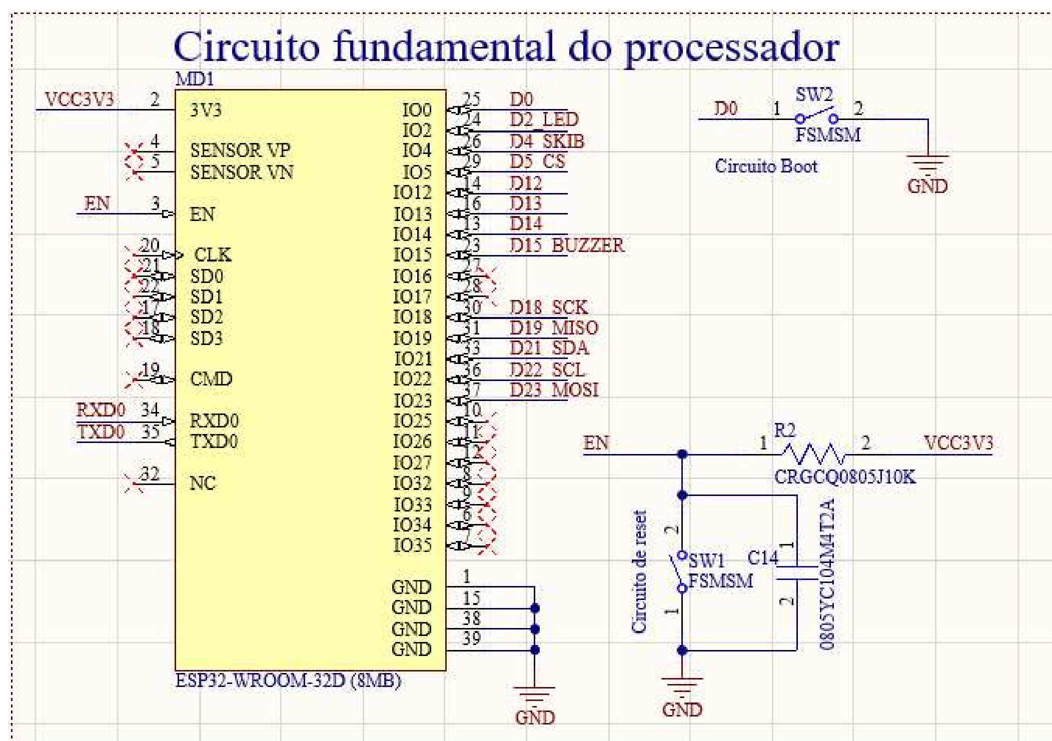
O circuito fundamental do módulo do processador ESP32 (Imagem 33) foi projetado com base nos diagramas fornecidos pelo fabricante. Este projeto assegura que todas as conexões e funcionalidades essenciais sejam corretamente implementadas para o funcionamento do sistema.

Os pinos RXD0 e TXD0 do ESP32 são utilizados para a interface com o conversor USB-TTL, facilitando a programação e atualizações de firmware. Os pinos D2 e D15 são designados para o controle do LED e do *buzzer*, respectivamente, indicando o status operacional do sistema. O pino D4 é responsável pelo acionamento da carga pirotécnica, uma função crítica para o sistema.

Para a interface homem-máquina (IHM) os pinos D12, D13 e D14 são configurados para ler a chave rotativa da IHM, permitindo uma navegação eficiente pelos menus e opções.

No que diz respeito à comunicação SPI, o pino o pino D5 (CS), D18 (SCK), D19 (MISO), e D23 (MOSI) são utilizados para a comunicação com o sdcard. Já a comunicação I2C utiliza os pinos D21 (SDA) e D22 (SCL), que são essenciais para a comunicação com sensores como o acelerômetro, barômetro e o display da IHM.

Imagem 33 – Circuito fundamental do processador



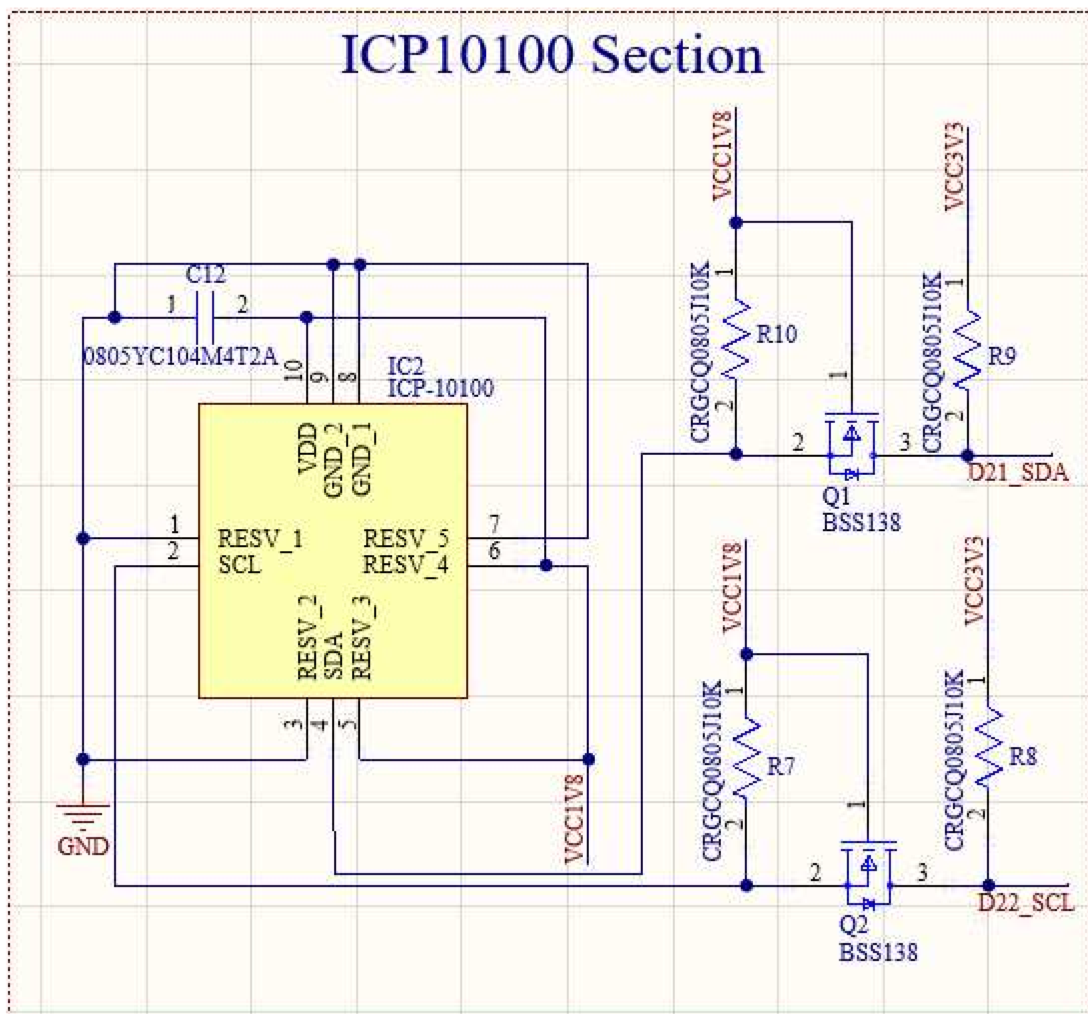
Fonte: O Autor.

4.4.3. Circuito do barômetro

O barômetro digital ICP10100 (Imagem 34), que opera a 1,8 V, requer um circuito específico para garantir seu correto funcionamento. Para isso, foram implementados dois *level shifters,* permitindo a comunicação adequada com o processador ESP32, que opera a uma tensão diferente. A implementação dos *level shifters* é crucial para evitar problemas ocasionados por sobretensão, protegendo assim tanto o barômetro quanto o processador.

O circuito para o funcionamento do ICP10100 foi projetado com base nos diagramas fornecidos pelo fabricante. Assegurando a confiabilidade das leituras barométricas, que são essenciais para o monitoramento das condições de voo e o desempenho geral do sistema. A utilização dos *level shifters* facilita a tradução de níveis de tensão entre os dispositivos, mantendo a integridade dos sinais de comunicação e evitando danos ao hardware.

Imagem 34 – Circuito para o funcionamento do barômetro ICP10100.



Fonte: O Autor.

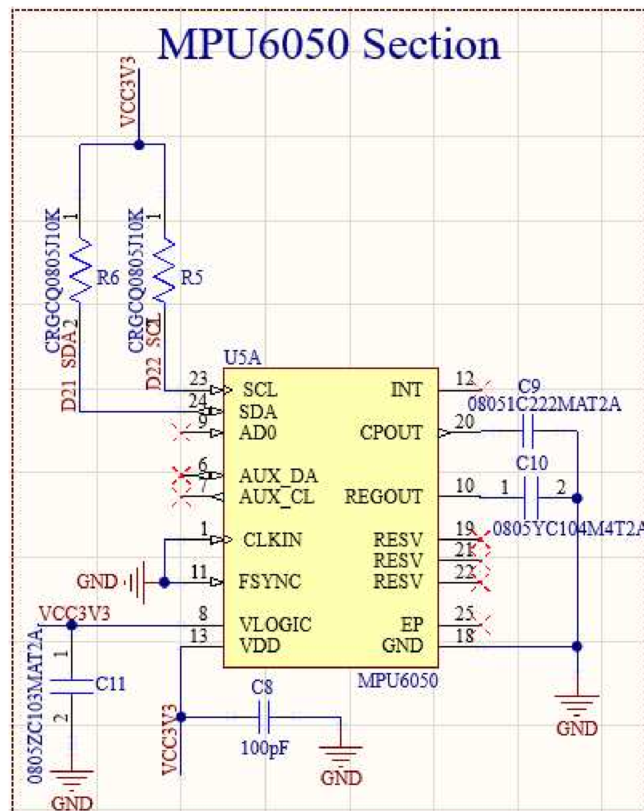
4.4.4. Circuito acelerômetro

Assim como no barômetro, o circuito do acelerômetro (Imagem 35) foi projetado com base nos diagramas fornecidos pelo fabricante. Esse procedimento garante que todas as especificações técnicas e operacionais sejam atendidas, proporcionando um desempenho confiável e preciso.

Seguindo os diagramas, foram adicionados alguns capacitores de desacoplamento no circuito do acelerômetro. Esses capacitores são essenciais para filtrar ruídos e estabilizar a tensão de alimentação, assegurando uma operação estável e minimizando interferências que poderiam afetar a precisão das medições.

Além dos capacitores, foram implementados resistores de *pull-up* nos barramentos I2C. Esses resistores são cruciais para garantir que os sinais de comunicação entre o acelerômetro e o processador sejam interpretados corretamente, evitando falhas na transmissão de dados e garantindo a integridade das leituras. A adição dos resistores de *pull-up* mantém os níveis lógicos apropriados nos barramentos I2C, assegurando uma comunicação eficiente e sem erros.

Imagem 35 – Circuito para o funcionamento do acelerômetro MPU6050.



Fonte: O Autor.

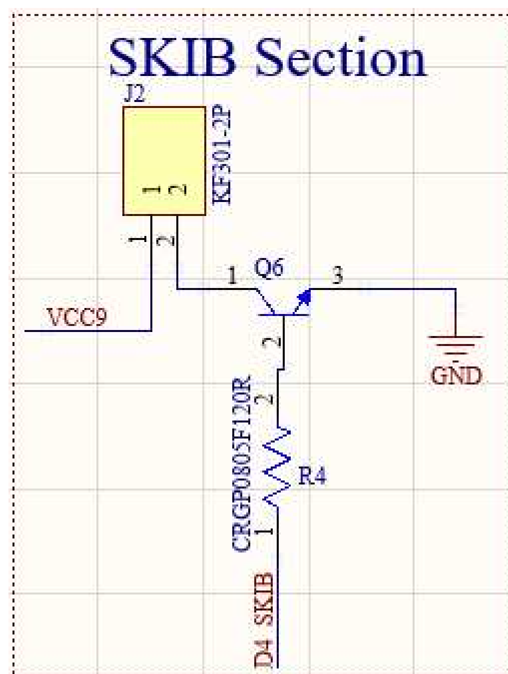
4.4.5. Circuito de acionamento da carga pirotécnica

O circuito para o acionamento da carga pirotécnica (Imagem 36) foi projetado de forma simples e eficiente. O pino do controlador responsável pelo acionamento envia um sinal para o transistor, que atua como uma chave, permitindo a passagem da corrente necessária para ativar a carga pirotécnica.

Esse design utiliza a capacidade do transistor de controlar a corrente através de um sinal de baixa potência do controlador. Quando o sinal é enviado pelo controlador, o transistor conduz, permitindo que a corrente flua através da carga pirotécnica e a acione.

A carga pirotécnica deve estar corretamente conectada ao borne designado para garantir um acionamento seguro e eficaz. Esse método de acionamento é confiável e minimiza o risco de falhas, proporcionando uma resposta rápida, dependendo da carga da bateria, e precisa na ativação da carga pirotécnica.

Imagem 36 - Circuito para a ativação da carga pirotécnica (Seção do SKIB).



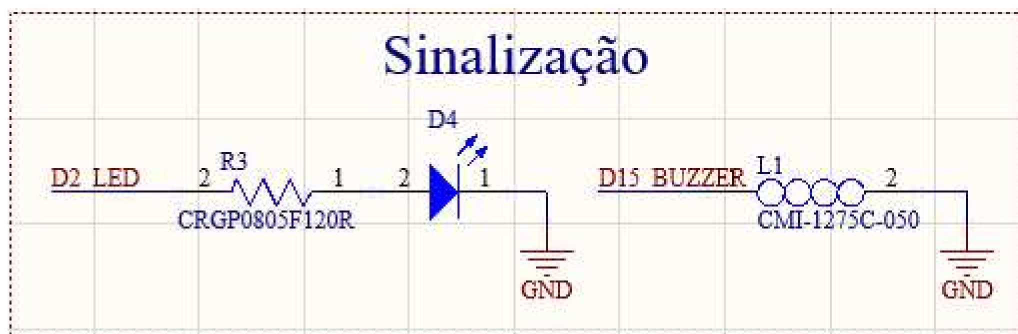
Fonte: O Autor.

4.4.6. Circuito das sinalizações sonora e luminosa

O circuito de sinalização (Imagem 37) de funcionamento é composto por dois componentes principais: um LED para sinalização luminosa e um buzzer para sinalização sonora. A presença desses componentes é crucial devido à imprevisibilidade das condições de lançamento, que podem acarretar diversos erros inesperados.

A combinação desses dois componentes permite ao usuário identificar de forma rápida e eficiente o estado do sistema, mesmo em condições adversas de lançamento. A integração do LED e do buzzer no circuito de sinalização melhora a usabilidade e a segurança do sistema, garantindo que o usuário esteja sempre ciente do status operacional do sistema e possa tomar ações corretivas se necessário.

Imagem 37 – Circuitos para sinalização luminosa e sonora.



Fonte: O Autor.

4.4.7. Circuito para a utilização do cartão de memória

O circuito do SD card (Imagem 38) foi desenvolvido seguindo os padrões e diagramas fornecidos pelo fabricante. Este circuito é essencial para o armazenamento local de dados, permitindo que todas as informações coletadas durante o voo sejam salvas de forma segura e acessível.

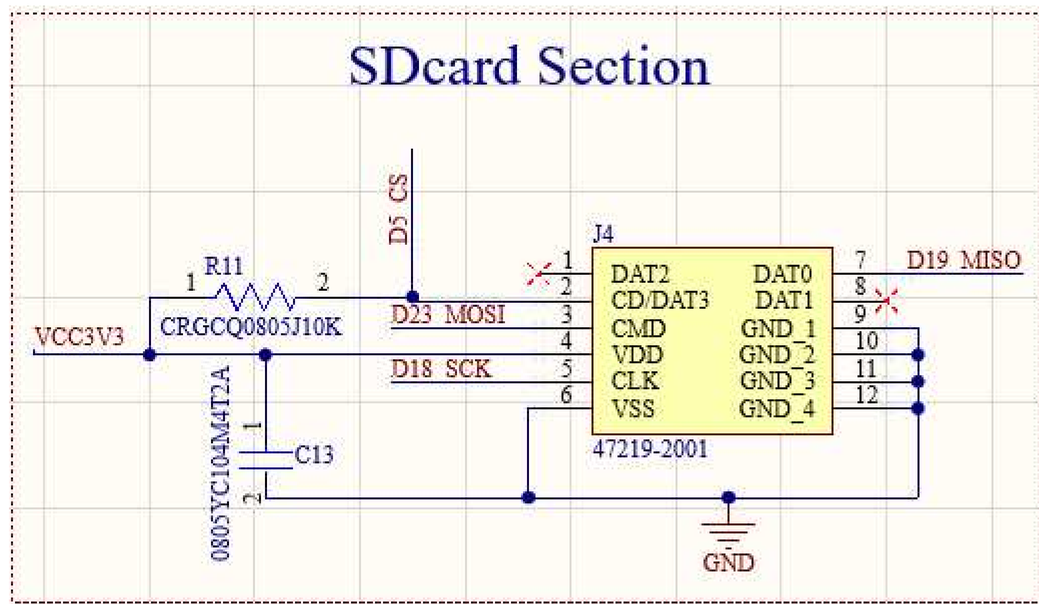
O SD card é conectado ao processador através do barramento SPI, utilizando os pinos dedicados para MISO (Master In Slave Out), MOSI (Master Out Slave In), SCK (Serial Clock) e CS (Chip Select). Estes pinos garantem a comunicação eficiente e rápida entre o processador e o cartão de memória, permitindo a leitura e gravação de dados conforme necessário.

Além da conexão SPI, o circuito inclui capacitores de desacoplamento para estabilizar a alimentação elétrica e garantir o funcionamento correto do SD card. A estabilização é

fundamental para evitar falhas de leitura/escrita que poderiam comprometer a integridade dos dados armazenados.

O circuito também possui um mecanismo de travamento mecânico para o SD card, protegendo-o contra vibrações e remoções acidentais durante o voo. Este detalhe é crucial para garantir que o SD card permaneça firmemente encaixado e funcional ao longo de toda a missão.

Imagem 38 – Circuito para a utilização do cartão de memória (Seção do cartão SD).



Fonte: O Autor.

4.4.8. Circuito dos componentes da IHM

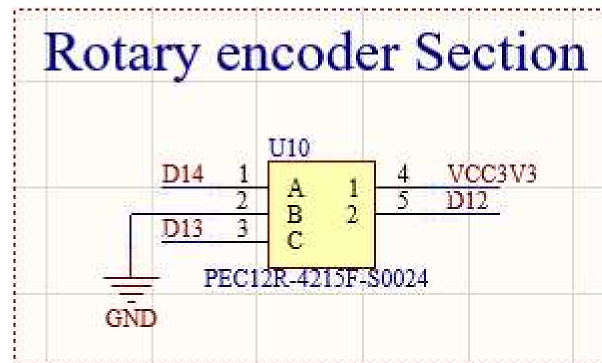
A interface homem-máquina (IHM) do sistema é composta por uma chave rotativa (Imagem 39) e um display OLED (Imagem 40). Esses componentes são essenciais para a interação do usuário com o sistema, permitindo a navegação pelos menus e o controle de suas funcionalidades.

A chave rotativa é conectada aos pinos D12, D13 e D14 do controlador, utilizando resistores *pull-up* internos da arquitetura do ESP32 para simplificar as conexões e garantir leituras estáveis dos sinais. Essa configuração permite que a chave rotativa funcione de forma eficaz, registrando os movimentos de rotação e pressionamento, que são utilizados para selecionar e confirmar as opções do menu.

O display OLED, por sua vez, é controlado através do barramento I2C, utilizando os pinos SDA (D21) e SCL (D22). Esta interface de comunicação é escolhida devido à sua

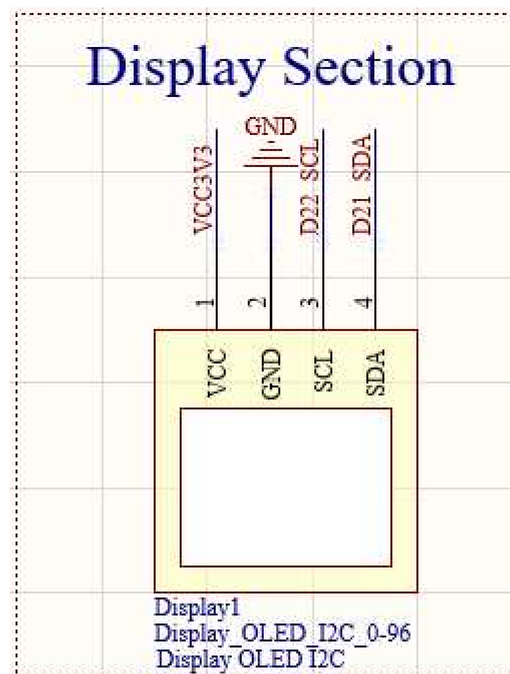
simplicidade e eficiência, permitindo uma fácil integração do display com o processador e uma comunicação rápida e confiável para a atualização das informações exibidas.

Imagem 39 – Conexão para leitura da chave rotativa da IHM (Seção do decodificador rotativo).



Fonte: O Autor.

Imagem 40 – Conexão para controle do display da IHM.

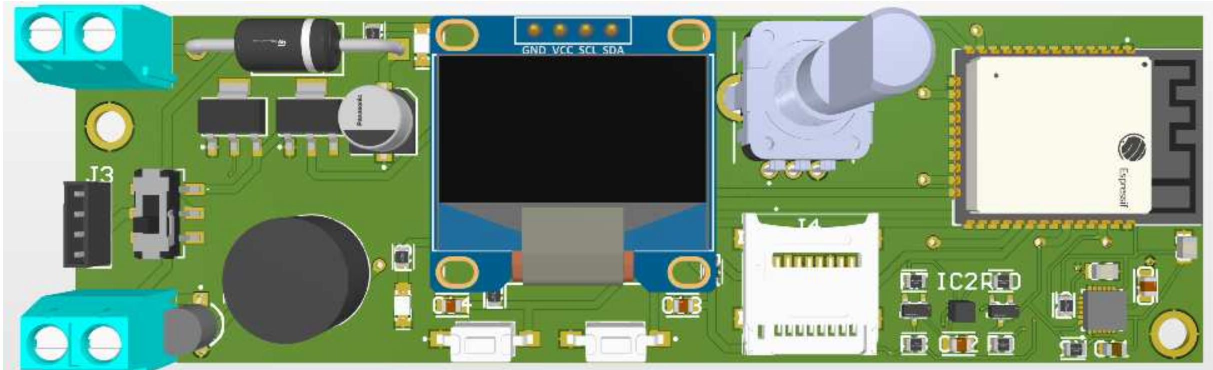


Fonte: O Autor.

4.4.9. Modelagem 3D completa da placa de circuito

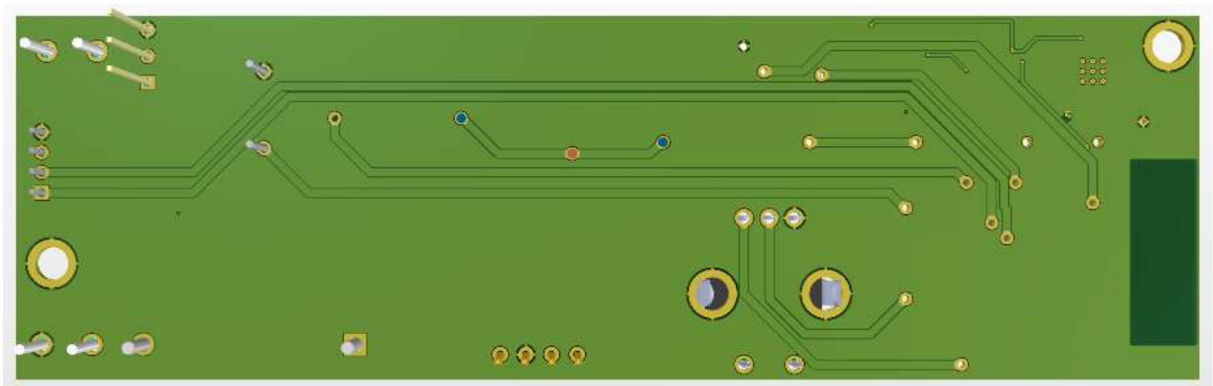
As imagens 41 e 42 apresentam como seria a versão inicial da PCB do sistema conforme os componentes descritos anteriormente.

Imagem 41 – Visão da Top Layer da PCB.



Fonte: O Autor.

Imagem 42 – Visão da Bottom Layer da PCB.



Fonte: O Autor.

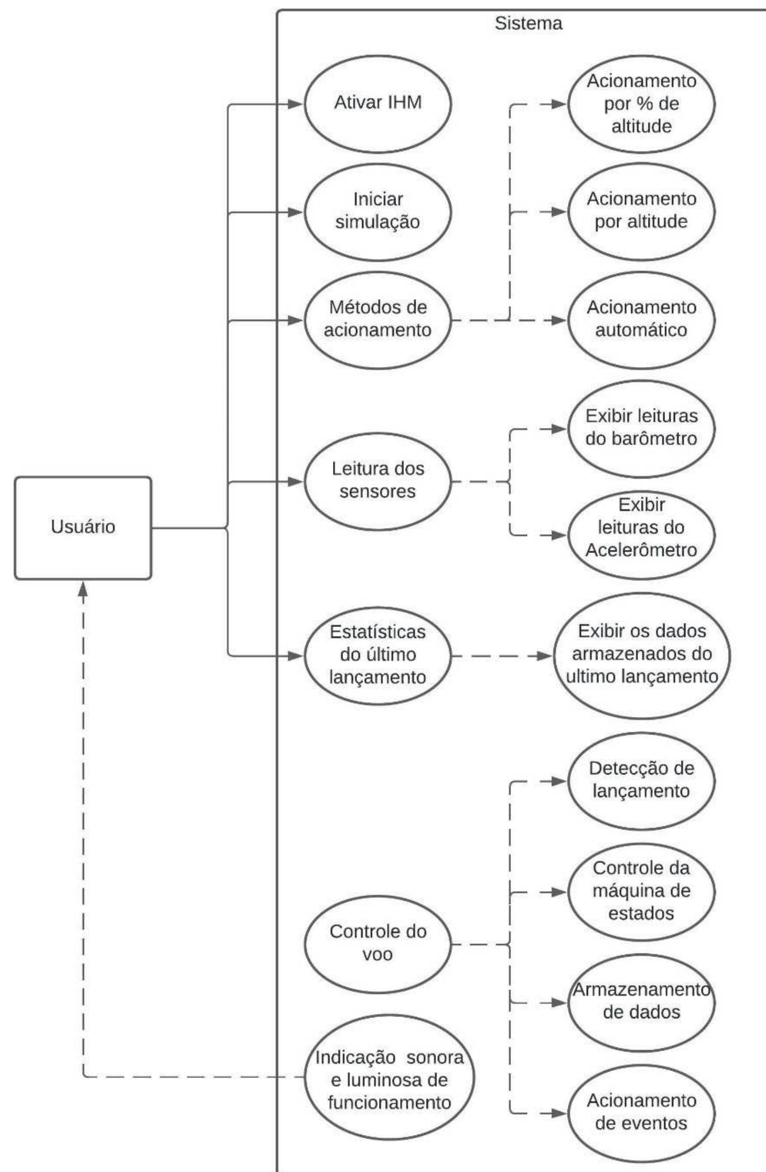
5. PROJETO DE SOFTWARE

Além das especificações de hardware, é imprescindível que o sistema possua um algoritmo lógico robusto para controlar todas as suas ações. Dessa forma, foi desenvolvido um código em linguagem Arduino baseado em C para gerenciar o funcionamento do computador de voo. O software desenvolvido é responsável por coordenar a leitura de sensores, o armazenamento de dados, a sinalização de eventos e o acionamento de cargas pirotécnicas, entre outras funções críticas. A seguir, serão detalhadas as diversas partes do código, abordando desde a linguagem e lógica utilizadas e, dependendo do componente, até os algoritmos específicos para o funcionamento do sistema.

Para proporcionar uma visão clara e estruturada das interações entre o usuário e o sistema de Interface Homem-Máquina (IHM) desenvolvido para o controle de voo do foguete, foi elaborado um diagrama de casos de uso (Imagem 43). Este diagrama é uma ferramenta essencial para entender as funcionalidades do software e como elas são acessadas e utilizadas pelo usuário.

A elaboração deste diagrama não só facilita a compreensão do fluxo de operações e a relação entre as diferentes funções do sistema, mas também serve como um guia para a implementação e melhorias futuras do software. Ele é particularmente útil na fase de desenvolvimento, onde cada caso de uso pode ser detalhado e testado individualmente para assegurar que o sistema atende às expectativas e requisitos do projeto. Nele, é possível observar funções que o usuário poderia acionar diretamente, como ativar a IHM, iniciar simulações, selecionar o método de acionamento, ver as leituras dos sensores e ver estatísticas do último lançamento. O sistema retorna feedbacks de funcionamento através de sinais luminosos e sonoros para o usuário além de realizar rotinas internas de controle de voo, como a detecção de lançamento, controle das rotinas de memória e acionamento de eventos.

Imagem 43 – Diagrama de casos de uso do sistema pelo usuário



Fonte: O Autor.

5.1. Linguagem de programação utilizada no projeto

A linguagem de programação utilizada para o desenvolvimento do software deste projeto é a linguagem Arduino, que se baseia no C/C++. A escolha dessa linguagem foi motivada por diversas razões, incluindo sua simplicidade, robustez e ampla comunidade de suporte.

Segundo o Professor Flávio Guimarães, do canal BrincandoComIdeias, a plataforma Arduino foi criada em 2005 por Massimo Banzi, David Cuartielles, Tom Igoe, Gianluca

Martino e David Mellis, no Interaction Design Institute Ivrea, na Itália. O objetivo inicial era fornecer uma ferramenta de fácil acesso para estudantes e entusiastas de eletrônica, possibilitando o rápido desenvolvimento de projetos de prototipagem eletrônica. Desde então, a linguagem Arduino e seu ambiente de desenvolvimento integrado (IDE) se tornaram extremamente populares, especialmente em projetos de microcontroladores e sistemas embarcados.

A linguagem Arduino é uma adaptação simplificada do C/C++, com bibliotecas específicas que facilitam a interação com hardware, como sensores, atuadores e outros componentes eletrônicos. Uma das principais vantagens da linguagem Arduino é a facilidade de uso. A sintaxe simplificada e a vasta quantidade de exemplos e bibliotecas disponíveis tornam o desenvolvimento de software mais acessível, mesmo para aqueles com pouca experiência em programação.

Para sistemas microcontrolados, como o computador de voo deste projeto, a linguagem Arduino oferece várias vantagens:

1. Facilidade de Programação: A linguagem é intuitiva e possui uma curva de aprendizado suave, permitindo que desenvolvedores se concentrem mais na lógica do projeto do que nos detalhes técnicos da programação.
2. Ampla Documentação e Suporte: A extensa comunidade de usuários e desenvolvedores de Arduino proporciona uma vasta quantidade de recursos, como tutoriais, fóruns de discussão e bibliotecas, que podem ser aproveitados para solucionar problemas e acelerar o desenvolvimento.
3. Compatibilidade com Diversos Componentes: A linguagem Arduino e suas bibliotecas são compatíveis com uma ampla gama de sensores, módulos de comunicação e outros componentes, facilitando a integração dos diversos elementos do sistema.
4. Eficiência e Desempenho: Embora seja simples de usar, a linguagem Arduino, sendo baseada em C/C++, permite a criação de código eficiente, adequado para as limitações de memória e processamento típicas de microcontroladores.
5. Desenvolvimento Rápido e Prototipagem: A simplicidade da linguagem e a disponibilidade de bibliotecas prontas para uso permitem um desenvolvimento rápido de protótipos, essencial para iterar e testar funcionalidades rapidamente.

Especificamente para um computador de voo, a linguagem Arduino é uma excelente escolha devido à necessidade de integração com diversos sensores e atuadores, além de requerer

uma alta confiabilidade e desempenho. A robustez da linguagem, combinada com a facilidade de implementação de algoritmos complexos e a possibilidade de realizar leituras e escritas rápidas de dados, faz com que seja ideal para este tipo de aplicação.

5.2. Utilização do acelerômetro

5.2.1. Inicialização do Acelerômetro

A primeira etapa para utilizar o acelerômetro é inicializá-lo. Isso envolve configurar os pinos do microcontrolador que se conectarão ao acelerômetro, bem como a biblioteca específica que facilitará a comunicação entre o microcontrolador e o sensor. A biblioteca utilizada geralmente depende do modelo do acelerômetro, mas para o MPU6050, uma das bibliotecas mais comuns é a Wire.h para comunicação I2C, junto com a biblioteca específica do sensor. A biblioteca escolhida foi a “MPU6050_tockn.h”, por questão de facilidade pois já conta filtros de leituras e realiza conversões adequadas para a correta interpretação das unidades dos dados.

5.2.2. Protocolo de Comunicação

O protocolo de comunicação utilizado pelo acelerômetro MPU6050 é o I2C. Esse protocolo permite a comunicação entre o microcontrolador e o acelerômetro usando apenas dois pinos: SDA (dados) e SCL (*clock*). O endereço I2C do MPU6050 deve ser conhecido para que o microcontrolador possa se comunicar com ele corretamente.

1. Configuração dos Pinos: Configurar os pinos SDA e SCL no microcontrolador (por exemplo, pinos D21 e D22 no ESP32).
2. Inicialização da Biblioteca I2C: Utilizar a biblioteca Wire.h para iniciar a comunicação I2C no setup do código.
3. Verificação de Conexão: Verificar se o acelerômetro está corretamente conectado e respondendo no endereço I2C esperado.

5.2.3. Leitura dos Dados

Uma vez que o acelerômetro esteja inicializado e a comunicação I2C configurada, é possível ler os dados do sensor. O processo de leitura geralmente envolve a solicitação de dados

dos registradores específicos do acelerômetro e a conversão desses dados em valores compreensíveis de aceleração.

1. Solicitação de Dados: Enviar um comando para o acelerômetro, solicitando a leitura dos dados de aceleração.
2. Leitura dos Registradores: Ler os valores dos registradores de dados de aceleração do eixo X, Y e Z.
3. Conversão dos Dados: Converter os dados brutos lidos dos registradores em valores de aceleração em g (gravidade).

5.3. Utilização do barômetro

5.3.1. Inicialização e Configuração

A inicialização do barômetro começa com a configuração dos pinos de comunicação I2C (SDA e SCL) no ESP32, que são conectados aos pinos 21 (SDA) e 22 (SCL) respectivamente. É necessário garantir que o barramento I2C esteja devidamente configurado com resistores *pull-up* para manter a linha de dados em estado alto quando o barramento não estiver em uso.

Após a configuração dos pinos, o próximo passo é inicializar a comunicação I2C e verificar se o sensor está corretamente conectado e respondendo aos comandos. Isso pode ser feito enviando um comando de inicialização e aguardando uma resposta adequada do sensor.

5.3.2. Leitura dos Dados

Uma vez que a comunicação com o ICP10100 está estabelecida, é possível configurar o sensor para realizar medições periódicas. O código deve incluir funções para solicitar leituras de pressão, temperatura e calcular a altitude com base nos dados de pressão.

5.3.3. Procedimento de Leitura

1. Configuração do Sensor: O sensor deve ser configurado para operar no modo desejado (por exemplo, modo contínuo ou modo de baixa potência). Isso envolve escrever valores específicos nos registradores de configuração do sensor via I2C.

2. Solicitação de Leitura: Para solicitar uma leitura, o código envia um comando específico para o sensor indicando que uma nova leitura deve ser realizada.
3. Leitura dos Dados: Após um tempo de espera adequado para a conclusão da medição, o código lê os valores dos registradores do sensor. Estes valores são então convertidos em unidades compreensíveis (por exemplo, Pa para pressão, °C para temperatura).
4. Cálculo da Altitude: Com base nos dados de pressão lidos, a altitude é calculada utilizando a fórmula padrão de conversão de pressão para altitude, levando em consideração a pressão ao nível do mar.

5.4. Controle das indicações sonora e luminosa

As indicações sonora e luminosa são componentes essenciais para o monitoramento do funcionamento do sistema em tempo real. Elas fornecem feedback ao usuário sobre o estado do sistema e notificam sobre possíveis erros ou eventos importantes durante a operação. No projeto em questão, foram utilizados um *buzzer* para sinalização sonora e um LED para sinalização luminosa.

5.4.1. Inicialização e Configuração dos Componentes

A configuração inicial dos componentes de sinalização sonora e luminosa envolve a definição dos pinos do microcontrolador aos quais eles estão conectados. No caso do ESP32, os pinos utilizados foram configurados no início do código, garantindo que os sinais corretos sejam enviados para o *buzzer* e os LEDs.

Para a sinalização luminosa, foi utilizado um LED que indicam diferentes estados do sistema. O LED principal é conectado ao pino D2 do ESP32 e é utilizado para mostrar que o sistema está energizado e funcionando corretamente. A sinalização sonora é realizada utilizando um *buzzer*, conectado ao pino D15 do ESP32. O *buzzer* é utilizado para emitir diferentes padrões de som que indicam o estado do sistema. As duas sinalizações funcionando juntas dando um indicativo visual e sonoro, elas atuam de maneira intermitente enquanto o sistema está em modo de pré-lançamento, caso o sistema são realize essa sinalização, significa que existe algum problema e o usuário deve verificar.

5.5. Utilização dos elementos da IHM

A Interface Homem-Máquina (IHM) do sistema é composta por dois elementos principais: um display I2C e uma chave rotativa integrada por um botão e um encoder. Esses componentes são utilizados para exibir e modificar as configurações do sistema, proporcionando uma interação intuitiva e eficiente para o usuário.

5.5.1. Display I2C

O display I2C é responsável por apresentar as informações do sistema e permitir a navegação pelas diversas telas de configuração. As telas são compostas por itens de menu, cabeçalhos, e possuem um modo de suspensão baseado em um timer de inatividade. Esse timer desliga o display após um período sem interação, economizando energia e aumentando a vida útil do sistema.

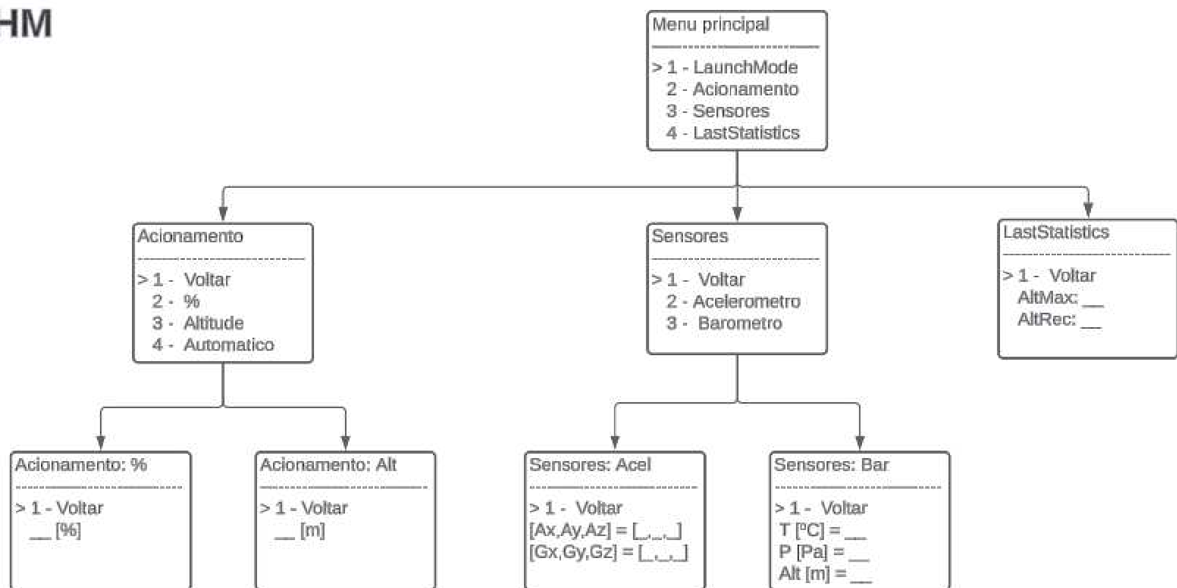
5.5.2. Chave Rotativa com Encoder Integrado

A chave rotativa com encoder integrado é o dispositivo de entrada principal da IHM. O encoder permite a navegação entre as telas e opções de menu, enquanto o botão integrado é utilizado para selecionar itens e confirmar ações. O funcionamento da chave rotativa é baseado na leitura de pulsos dos pinos do encoder, diferenciando os comandos de rotação para subir e descer nas opções, e os cliques do botão para seleção.

5.5.3. Interface

A interface do sistema é composta por telas de menus, que incluem cabeçalhos e opções organizadas de forma intuitiva para facilitar a compreensão do usuário. A seguir, a imagem 44 apresenta uma ilustração do diagrama das telas:

Imagem 44 – Diagrama da distribuição de telas da IHM.



Fonte: O Autor.

5.5.4. Funcionamento

1. Navegação pelo Menu: Ao girar a chave rotativa, o encoder gera pulsos que são interpretados pelo sistema para navegar entre as opções de menu. A rotação no sentido horário move a seleção para baixo, enquanto a rotação no sentido anti-horário move a seleção para cima.
2. Seleção de Opções: Ao pressionar o botão integrado na chave rotativa, o sistema reconhece o comando de seleção. Esse comando permite ao usuário entrar em submenus ou modificar valores dos parâmetros selecionados.
3. Modificação de Parâmetros: Quando uma opção que permite modificação é selecionada, o usuário pode ajustar o valor girando a chave rotativa e alterando seu lavor dentro de uma faixa. Após a alteração, o novo valor é salvo no cartão SD, garantindo que as configurações sejam mantidas entre reinicializações.
4. Modo de Suspensão: O sistema monitora a atividade do usuário e, após um período definido de inatividade, entra em modo de suspensão, desligando o display para economizar energia. Qualquer interação com a chave rotativa ou o botão reativa o display, permitindo ao usuário continuar a operação normalmente.

É importante destacar que o sistema só permite a utilização do da IHM durante o estado de pré-lançamento. Quando um lançamento é detectado, essa funcionalidade é desabilitada e o sistema ativa as rotinas necessárias para o controle e monitoramento do voo.

5.5.5. Lógica para leitura da chave rotativa

A chave rotativa possui 3 saídas de sinal, duas referentes aos pulsos do encoder e uma referente ao botão. O sistema analisa esses pulsos para determinar a ação a ser realizada.

Exemplificando, denota-se os pinos da chave rotativa como as saídas A e B e C, nas quais saem os pulsos do lado esquerdo e direito do encoder e o estado do botão, respectivamente. Os sinais são da forma de ondas quadradas de maneira que, em cada caso, o microcontrolador identifica as ações conforme a Tabela 2:

Tabela 2 – Identificação dos comentos da chave rotativa

Ação	**Sinal A**	**Sinal B**	**Sinal C**
Girar o encoder para a esquerda	0	1	X
Girar o encoder para a direita	1	0	X
Pressionar o botão	X	X	0
Nenhuma ação	1	1	1

Fonte: O Autor

5.5.6. Algoritmo para utilização do display

A geração das telas da IHM é fundamentada na utilização de uma matriz que contém os elementos essenciais do display. A partir dessa matriz, o microcontrolador coordena a construção das telas, posicionando os diversos elementos de acordo com um algoritmo específico de organização. Este algoritmo permite ao sistema controlar de maneira eficaz a disposição dos cabeçalhos e das opções associadas a cada menu, garantindo uma apresentação clara e organizada das informações para o usuário. As telas são construídas linha por linha e são utilizadas variáveis de marcação para identificar cada tela e realizar funções específicas. As matrizes estão ilustradas na Imagem 45:

Imagem 45 – Matriz de construção das telas

```
// Define os nomes das opções do menu
const char menutitulos[][20] = { "Menu principl", "Acionamento", "Sensores" };
const char menuOptions[][4][20] = { { "1 - LaunchMode", "2 - Acionamento", "3 - Sensores", "4 - LastStatistics" },
                                    { "1 - Voltar", "2 - Altitude", "3 - %", "4 - Automatico" },
                                    { "1 - Voltar", "2 - Acelerometro", "3 - Barometro", "" },
                                    { "1 - Voltar", "Alt[m]:", "", "" },
                                    { "1 - Voltar", "Alt[%]:", "", "" },
                                    { "1 - Voltar", "A:", "G:", "*:" },
                                    { "1 - Voltar", "Temp[C]: ", "Pres[Pa]: ", "Alt[m]: " } };
```

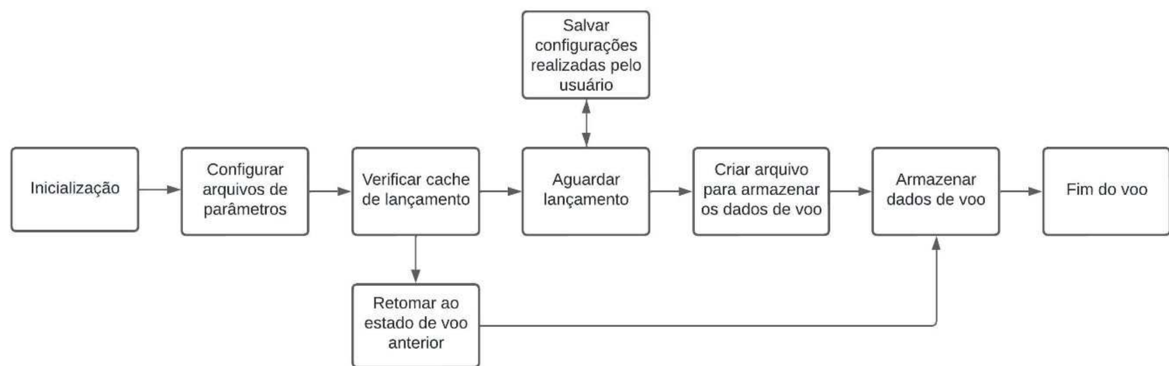
Fonte: O Autor.

5.6. Utilização do cartão de memória

O cartão de memória é utilizado para armazenar os dados de voo, salvar configurações e realizar verificações de segurança. Inicialmente, o módulo de memória é inicializado por meio de bibliotecas adequadas. Em seguida, as configurações são verificadas. Caso não existam configurações salvas no cartão de memória, o sistema grava os parâmetros padrão definidos em software. Em inicializações subsequentes, o sistema dará preferência aos dados presentes no cartão de memória e verificará o estado de voo, determinando se o sistema foi desligado durante um lançamento ou não, e tomando as ações apropriadas para cada situação.

Posteriormente, o cartão de memória será utilizado para salvar as configurações modificadas através da IHM até o momento em que um lançamento seja detectado e o sistema calcule um nome adequado para os arquivos de dados de voo. Para a verificação de lançamento, um arquivo de "cache" é armazenado na memória do cartão SD, contendo um parâmetro booleano para identificar se foram detectados um lançamento e um parâmetro numérico para identificar o estado atual do voo. Dessa forma, ao inicializar o sistema, esse cache é verificado em relação ao voo anterior. Caso o sistema não esteja nas condições iniciais definidas durante a primeira inicialização ou na conclusão do voo, ele identifica o estado anterior e retoma o seu funcionamento a partir desse ponto. A seguir, a Imagem 46 apresenta o diagrama das rotinas do cartão de memória:

Imagem 46 – Diagrama de rotinas do cartão de memória.



Fonte: O Autor.

5.7. Métodos para acionamento da recuperação

O momento mais crítico após o lançamento é o acionamento do mecanismo de recuperação, que assegura o pouso seguro do foguete. Geralmente, utiliza-se um paraquedas para esse propósito. Este trabalho abordará o acionamento através de uma carga pirotécnica, que pode ser utilizada para ativar a recuperação, como a liberação de um paraquedas, por exemplo.

Dentre os métodos para detectar o momento de ativação, foi utilizado o método barométrico. Nele, a recuperação será acionada quando o foguete atingir uma altitude determinada através da IHM. Os métodos para acionamento consistem em três tipos de verificação:

1. **Valor percentual do apogeu detectado**: O sistema considera um valor percentual baseado no apogeu, a altitude máxima atingida pelo foguete.
2. **Altitude específica definida pelo usuário**: O usuário define uma altitude específica para a ativação da recuperação.
3. **Método automático**: Este método aciona a recuperação em 50% do apogeu para qualquer lançamento, valor padrão em competições de modelismo de foguetes.

Esses métodos garantem uma flexibilidade no acionamento, permitindo ajustes conforme as necessidades específicas de cada lançamento e competição.

5.8. Acionamento barométrico

O acionamento da recuperação utilizando o método barométrico é uma técnica essencial em sistemas de recuperação de foguetes. Esse método baseia-se na medição precisa da altitude do foguete, calculada a partir das leituras de pressão atmosférica fornecidas por um barômetro.

A pressão atmosférica diminui com o aumento da altitude, um princípio bem estabelecido em física atmosférica. Barômetros digitais, como o ICP10100, são capazes de medir essas variações de pressão com velocidade e boa resolução, permitindo calcular a altitude do foguete em tempo real.

Para implementar o método barométrico em um sistema de recuperação, o barômetro digital é inicialmente calibrado e configurado para transmitir dados de pressão ao microcontrolador principal do foguete. A lógica de controle do microcontrolador utiliza esses dados para calcular a altitude atual do foguete em intervalos regulares.

O método barométrico é amplamente utilizado devido à sua precisão e confiabilidade. Barômetros digitais modernos oferecem leituras estáveis e rápidas, essenciais para a detecção precisa do apogeu e o acionamento oportuno da recuperação. Além disso, o método barométrico é menos suscetível a interferências mecânicas ou erros associados a outros métodos, como sensores de velocidade ou giroscópios.

5.9. Máquina de estados do sistema

A máquina de estados do sistema é essencial para controlar a sequência de eventos e ações do computador de voo durante a missão do foguete. Cada estado representa uma fase específica do voo e define as ações que o sistema deve realizar. A transição entre os estados ocorre com base em condições predefinidas, permitindo um controle preciso e organizado do processo. Abaixo, são descritos os principais estados e suas funções:

Estado de Pré-lançamento: No estado de pré-lançamento, o sistema analisa continuamente os dados de altitude aguardando o início do lançamento. Durante este estado, a Interface Homem-Máquina (IHM) está ativa, permitindo ao usuário configurar os parâmetros de voo, verificar o status do sistema e fazer ajustes necessários. Este estado é crítico para garantir que todas as condições estejam adequadas para o lançamento.

Estado Subindo: Assim que o lançamento é detectado, o sistema transita para o estado "Subindo". Neste estado, um novo arquivo é criado no cartão de memória para armazenar os dados do voo. O sistema começa a salvar informações vitais, como altitude, aceleração e outros parâmetros relevantes. O estado "Subindo" continua até que o foguete atinja o apogeu, o ponto mais alto de seu voo.

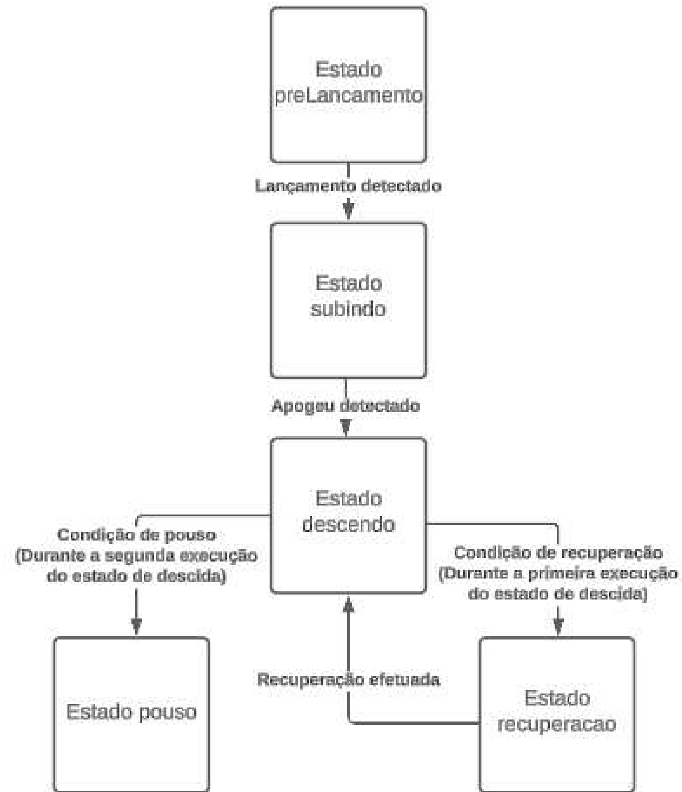
Estado Descendo: Após alcançar o apogeu, o sistema entra no estado "Descendo". Durante a descida, o sistema continua a salvar dados no cartão de memória. Este estado é monitorado de perto para identificar o momento apropriado para acionar a recuperação, com base nas condições predefinidas, como altitude ou desaceleração.

Estado Recuperação: Neste estado, a carga pirotécnica é acionada para liberar o paraquedas ou outro mecanismo de recuperação. Após a ativação, o sistema retorna ao estado "Descendo" para continuar monitorando a descida até o pouso seguro do foguete.

Estado Pouso: O estado "Pouso" é ativado quando o sistema detecta que o foguete parou de se movimentar. Neste momento, os arquivos de dados abertos são fechados e o sistema entra em estado de hibernação para conservar energia. Este estado finaliza o ciclo de voo, garantindo que todos os dados relevantes foram salvos e que o sistema está pronto para ser recuperado e reutilizado em futuras missões.

A Imagem 47 apresenta um diagrama resumindo o funcionamento da máquina de estados:

Imagem 47 – Diagrama de funcionamento da máquina de estados.



Fonte: O Autor.

6. INSTRUMENTAÇÃO DOS SENSORES

A instrumentação de sensores é um campo fundamental na engenharia, que envolve a utilização de dispositivos para medir variáveis e convertê-las em sinais que podem ser lidos e processados por sistemas eletrônicos. Esses sensores são essenciais em uma ampla variedade de aplicações, incluindo automação industrial, monitoramento ambiental, sistemas de segurança, e, particularmente, em sistemas aviônicos e computadores de voo. No contexto dos sistemas aviônicos, a instrumentação precisa e confiável dos sensores é crucial para garantir a segurança e eficiência dos lançamentos.

A instrumentação de sensores desempenha um papel vital em sistemas de controle e monitoramento. Em computadores de voo, sensores como acelerômetros e barômetros fornecem dados críticos sobre o estado e o desempenho do veículo durante todas as fases do voo. Esses dados são utilizados para monitorar a trajetória, detectar condições anômalas e ativar mecanismos de segurança, como a recuperação do foguete. A precisão e a confiabilidade desses sensores são fundamentais para garantir que as decisões tomadas pelo sistema de controle sejam corretas e oportunas.

A calibração de sensores é o processo de ajustar e verificar a precisão de um sensor, garantindo que suas medições estejam dentro de especificações aceitáveis. A calibração é essencial porque sensores podem apresentar desvios, variações na sensibilidade e outros erros devido a fatores como fabricação, envelhecimento e condições ambientais. Um sensor mal calibrado pode fornecer dados imprecisos, levando a decisões erradas e, potencialmente, a falhas catastróficas.

Para otimizar o desempenho dos sensores, é essencial compreender os tipos de erros que podem afetar suas leituras. Entre esses erros estão o *bias* (ou *offset*), que representa o erro em zero, e o ganho (ou escala), que define a proporcionalidade entre o valor real e o valor medido. Além desses, existem fatores como a assimetria, a não-linearidade, a zona morta e a quantização, cada um impactando a precisão das medições de maneira específica (DOEBELIN, 1990).

A correção desses fatores, através da função de transferência do sensor, é fundamental para converter as leituras obtidas em valores precisos de grandeza real (EMBARCADOS, 2018). Por exemplo, sensores não lineares, como os utilizados para medir a velocidade do ar, exigem correções específicas baseadas em equações matemáticas complexas, como as derivadas do teorema de Bernoulli, garantindo assim que as medições sejam precisas e confiáveis.

Nos computadores de voo e sistemas aviônicos, a calibração dos sensores é de extrema importância devido às condições rigorosas e exigências de alta precisão típicas dessas aplicações. A calibração garante que os sensores forneçam dados precisos e confiáveis em todas as condições operacionais. Por exemplo, um acelerômetro mal calibrado pode resultar em medições incorretas de aceleração, afetando a detecção de eventos críticos como o lançamento e o apogeu. Similarmente, um barômetro não calibrado pode fornecer leituras erradas de altitude, comprometendo a ativação dos mecanismos de recuperação. Portanto, a calibração regular e precisa dos sensores é uma prática indispensável para manter a integridade e a segurança dos sistemas de controle de voo.

6.1. Calibração do barométrico

A calibração do barômetro é um passo crucial para garantir a precisão e a confiabilidade dos dados de altitude em um computador de voo. O barômetro é responsável por medir a pressão atmosférica, que é então convertida em dados de altitude. Esses dados são fundamentais para monitorar a trajetória do voo, detectar o apogeu e ativar mecanismos de recuperação, como o paraquedas. A precisão das medições do barômetro é, portanto, essencial para a segurança e a eficácia do sistema aviônico.

6.1.1. Princípios de funcionamento de barômetros digitais

Os barômetros digitais funcionam medindo a pressão atmosférica e convertendo essas medições em dados eletrônicos que podem ser processados por um microcontrolador. Eles utilizam sensores piezo-resistivos ou capacitivos que respondem às mudanças na pressão do ar. Quando a pressão atmosférica varia, essas mudanças são detectadas pelo sensor, que gera um sinal elétrico proporcional à pressão medida. Este sinal é então digitalizado e interpretado pelo sistema de controle do voo (NORTHROP, 2014).

Os barômetros digitais são amplamente utilizados em aplicações aviônicas devido à sua alta precisão, estabilidade e capacidade de fornecer dados em tempo real. Eles são integrados com circuitos eletrônicos que permitem uma calibração precisa e ajustes automáticos para compensar variações ambientais.

6.1.2. Métodos de calibração de barômetros digitais

A importância da calibração do barômetro em um computador de voo não pode ser subestimada. A pressão atmosférica varia com a altitude e também pode ser influenciada por condições meteorológicas. Um barômetro não calibrado pode fornecer leituras imprecisas, o que pode levar a decisões erradas pelo sistema de controle do voo. Por exemplo, se o barômetro subestimar a altitude, o paraquedas pode ser acionado prematuramente, resultando em um pouso inadequado. Por outro lado, se o barômetro superestimar a altitude, o paraquedas pode ser acionado tarde demais, aumentando o risco de danos ao foguete.

A calibração do barômetro envolve a comparação das leituras do sensor com um padrão conhecido e ajustando o sensor para corrigir quaisquer desvios. Este processo pode ser realizado em várias etapas do ciclo de vida do sensor: durante a fabricação, antes do lançamento, e, em alguns casos, durante o voo, utilizando sistemas de autocalibrarão.

Uma técnica eficaz para calibrar um barômetro digital envolve o uso de um ambiente de pressão controlada, utilizando bombas de ar capazes de aumentar ou diminuir a pressão dentro de um recipiente. Esse método emprega um instrumento de medição de pressão de referência, como um manômetro analógico acoplado ao recipiente, para medir múltiplos pontos de pressão. Dessa forma, é possível calibrar o barômetro digital por meio de compensação direta ou pela obtenção de uma curva de interpolação, garantindo a precisão das leituras de pressão.

6.2. Calibração do acelerômetro

A calibração de acelerômetros é uma etapa crucial no desenvolvimento e manutenção de sistemas de medição precisos, especialmente em aplicações como modelagem de foguetes, onde a precisão é fundamental. A calibração garante que os dados obtidos dos acelerômetros sejam precisos e confiáveis.

No foguetemodelismo, especialmente em projetos avançados que envolvem sistemas de navegação inercial, como os computadores de voo, os acelerômetros desempenham um papel crucial. Eles são utilizados para medir a aceleração do foguete ao longo do tempo, fornecendo dados essenciais para calcular a velocidade e a posição do foguete durante o voo. A calibração precisa dos acelerômetros assegura que esses dados sejam confiáveis, mesmo em condições extremas de aceleração.

6.2.1. Princípios de funcionamento de acelerômetros digitais

Os acelerômetros digitais são sensores que medem a aceleração de um objeto em uma ou mais direções. Eles funcionam baseando-se no princípio da capacitância variável ou no efeito piezoelétrico, dependendo do tipo de acelerômetro (NORTHROP, 2014).

Capacitância Variável: A maioria dos acelerômetros digitais utiliza o princípio da capacitância variável, onde uma massa interna em movimento altera a distância entre placas capacitivas conforme o sensor é acelerado. Essa mudança na capacitância é convertida em um sinal elétrico proporcional à aceleração.

Efeito Piezoelétrico: Outros acelerômetros usam materiais piezoelétricos que geram uma carga elétrica quando submetidos a uma força mecânica. Essa carga é proporcional à força aplicada, permitindo a medição da aceleração.

Os acelerômetros digitais convertem os sinais analógicos gerados por esses mecanismos em sinais digitais através de um conversor A/D (analógico para digital). Esses sinais digitais podem então ser processados por microcontroladores ou outros sistemas eletrônicos.

6.2.1. Métodos de calibração de acelerômetros e giroscópios digitais

Os acelerômetros e giroscópios são componentes fundamentais em sistemas de medição inercial, utilizados em diversas aplicações que demandam precisão na captura de movimentos e orientação. O acelerômetro, embora nomeado de forma um tanto equivocada, na verdade mensura a força resultante aplicada sobre ele, que inclui tanto acelerações lineares quanto a aceleração gravitacional. Cada eixo do acelerômetro é composto por um sistema massa-mola que se deforma de acordo com a aceleração aplicada (NORTHROP, 2014), permitindo calcular a aceleração resultante com base na Lei de Newton ($F = ma$).

Para calibrar um acelerômetro, utiliza-se a gravidade como referência. Este processo envolve posicionar o sensor em seis orientações distintas (+x, -x, +y, -y, +z, -z), alinhando cada eixo com a aceleração gravitacional. Os valores obtidos nessas orientações são então utilizados para determinar os offsets (valores de leitura sem aceleração) e os ganhos (fatores de escala para converter valores numéricos em unidades de aceleração real) para os eixos X, Y e Z do sensor (EMBARCADOS, 2018).

Por outro lado, os giroscópios, ou velocímetros angulares, medem a taxa de rotação angular em torno de um eixo específico. Estes sensores baseiam-se em uma massa que vibra quando o sensor é rotacionado, gerando um deslocamento linear mensurável. Diferentemente dos acelerômetros, os giroscópios são sensíveis ao offset, especialmente devido a variações de temperatura e outras condições ambientais. Portanto, a calibração de um giroscópio requer o cálculo cuidadoso de valores de bias (offset inicial) e escala (fator de conversão para unidades angulares) para cada um dos eixos de medição (EMBARCADOS, 2018).

Para realizar a calibração de um giroscópio, são utilizados métodos como a integração numérica para calcular ângulos a partir das leituras de velocidade angular. Isso pode envolver rotacionar o sensor em movimentos conhecidos, como 90 graus, e comparar os valores calculados com os valores reais obtidos. Ajusta-se então o *bias* inicial e o fator de escala para garantir a precisão das medições angulares ao longo do tempo de operação do sensor (EMBARCADOS, 2018).

7. TESTES FUNCIONAIS DO SISTEMA

Para avaliar o desempenho do sistema e comprovar sua funcionalidade, foi montado um protótipo em protoboard utilizando jumpers e módulos dos sensores e componentes apropriados. Os testes foram realizados tanto de maneira unitária, em cada componente isoladamente, quanto com os componentes integrados. Esses ensaios foram essenciais para determinar parâmetros operacionais e definir estratégias no desenvolvimento do software como um todo.

Durante os testes, cada componente foi analisado separadamente para entender seu funcionamento e sua integração com outros elementos. O módulo do acelerômetro MPU6050 foi testado, e a variação de altitude foi simulada com base em um conjunto de equações representativas. Utilizou-se um ESP32 Dev Kit V1 como microcontrolador, juntamente com seu LED integrado. Para sinalização sonora, foi empregado um módulo de *buzzer*. Um display OLED I2C do mesmo modelo que seria utilizado na placa final foi utilizado como interface homem-máquina (IHM), juntamente com um módulo de chave rotativa. Adicionalmente, um módulo de cartão de memória com um sdcard de 64 GB foi utilizado para testar o armazenamento de dados.

Com essas abordagens, foi possível obter uma compreensão aprofundada do funcionamento individual de cada componente e de sua interação no sistema integrado, assegurando um desenvolvimento de software eficiente e funcional.

7.1. Teste do barômetro

Devido à necessidade de importar o barômetro ICP10100, juntamente com os procedimentos necessários e custos adicionais para a adaptação de um módulo, não foi possível testar o sensor propriamente dito. Em vez disso, para realizar os testes do sistema, evolução da máquina de estados e demais recursos, foi simulado um lançamento manipulando as variáveis de pressão e altitude com base em equações barométricas no *Matlab*. Essas equações e suas implementações serão discutidas em detalhes posteriormente neste trabalho.

Para simular um cenário de lançamento, foi considerado uma variação de altitude de mil metros acima do nível do mar, mantendo uma temperatura constante durante a simulação.

Esses valores simulados permitiram verificar o comportamento do sistema sob condições variáveis de pressão e altitude, garantindo que os algoritmos desenvolvidos para a evolução da máquina de estados e controle do foguete fossem validados de maneira eficaz.

Os testes simulados indicaram que, mesmo sem a presença física do barômetro, foi possível avaliar a resposta do sistema às mudanças de pressão e altitude. Essa abordagem permitiu a identificação de possíveis falhas e a necessidade de ajustes nos algoritmos de controle, além de fornecer uma base sólida para a futura integração do barômetro real. A simulação mostrou-se uma ferramenta valiosa para prever o comportamento do sistema em condições reais de voo, destacando a importância da modelagem matemática na fase de desenvolvimento e teste de sistemas embarcados.

Ao seguir esta metodologia, é garantido que o sistema está preparado para a integração do barômetro ICP10100 assim que este estiver disponível, minimizando os riscos e o tempo de ajuste durante a fase de testes finais.

Para os parâmetros de variação de altitude simulados, inicialmente foram estabelecidos pontos como altitude máxima e alcance horizontal máximo, que o lançamento começa na origem de maneira a determinar uma equação de parábola:

$$x_1 = posição\ inical$$
$$x_2 = posição\ final$$
$$x_3 = \frac{x_2}{2} (posição\ onde\ ocorre\ ymax)$$
$$ymax = altitude\ máxima$$

A equação que descreve um movimento parabólico é da forma:

$$y = ax^2 + bx + c \text{ (1)}$$

Para determinar os coeficientes a, b e c, são utilizadas as seguintes condições:

1 – No ponto de lançamento (x1 = 0), a altitude inicial é zero, portando (1) fica:

$$0 = ax_1^2 + bx_1 + c, mas\ x_1 = 0, então: c = 0$$

$$y = ax^2 + bx \text{ (2)}$$

2 – No ponto de altitude máxima, aplicando x3 em (2), a altitude é ymax:

$$y_{max} = ax_3^2 + bx_3 \text{ (3)}$$

3 - No ponto final (x2), a altitude é zero:

$$0 = ax_2^2 + bx_2 \rightarrow ax_2 + b \rightarrow b = -ax_2 \text{ (4)}$$

Substituindo (4) em (3), encontramos os coeficientes a e b:

$$a = \frac{y_{\max}}{\mathrm{x}_3 * (\mathrm{x}_3 - \mathrm{x}_2)} \text{ (5)}$$

$$b = \frac{-y_{\max}}{\mathrm{x}_3 * (\mathrm{x}_3 - \mathrm{x}_2)} \text{ (6)}$$

Portanto, aplicando (5) e (6) em (2), a equação da altitude para a simulação é modelada por:

$$y = \frac{y_{\max}}{\mathrm{x}_3 * (\mathrm{x}_3 - \mathrm{x}_2)} * x^2 - \frac{\mathrm{ymax}}{\left(\mathrm{x}_3 * (\mathrm{x}_3 - \mathrm{x}_2)\right)} * x$$

Considerando que x3 < x2, é possível ajustar a equação para (7):

$$y = -\frac{y_{\max}}{\mathrm{x}_3 * (\mathrm{x}_3 - \mathrm{x}_2)} * x^2 + \frac{\mathrm{ymax}}{\left(\mathrm{x}_3 * (\mathrm{x}_3 - \mathrm{x}_2)\right)} * x \text{ (7)}$$

Para obter valores aproximados de pressão e altitude, foi utilizado a equação hipsométrica (8), que é fundamental na meteorologia e na engenharia aeroespacial para relacionar a variação de altitude com a pressão atmosférica. Esta equação é expressa como:

$$\mathrm{h} = \frac{\mathrm{T}_0}{L} * \left(\left(\frac{P_0}{P}\right)^{\frac{RL}{g}} - 1\right) \text{ (8)}$$

Onde:

h – Altitude

T0 – Temperatura ao nível do mar

P0 – Pressão ao nível do mar

P – Pressão de uma determinada altitude

R – Constante dos gases perfeitos

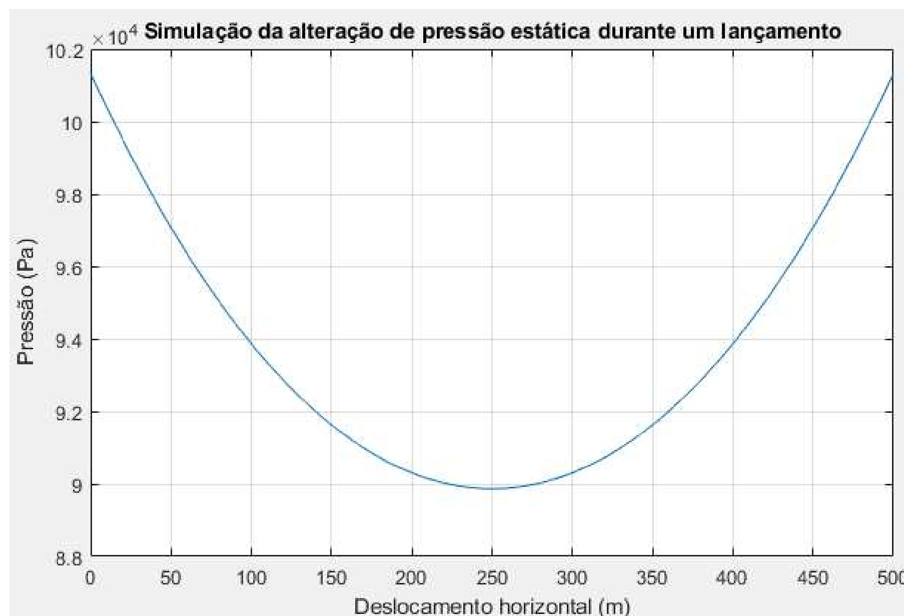
L – Taxa de variação da temperatura com a altitude

g – Gravidade

A equação (8) é válida sob condições atmosféricas padrão, assumindo uma temperatura constante e uma distribuição de pressão conhecida. A importância dessa equação reside na sua capacidade de fornecer uma relação precisa entre pressão e altitude, essencial para calibrar instrumentos de medição e realizar simulações de voo.

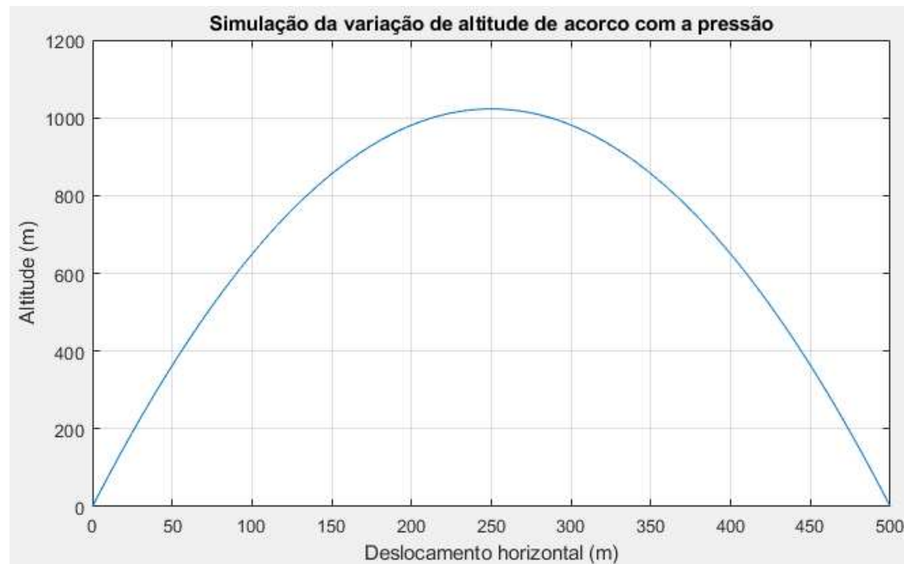
Para a simulação (Imagens 48 e 49), considerando uma variação de altitude de mil metros acima do nível do mar e uma temperatura constante, os valores aproximados de pressão e altitude foram calculados, servindo de base para os testes do sistema e a evolução da máquina de estados. Essa abordagem foi necessária devido à impossibilidade de testar diretamente o sensor barométrico ICP10100.

Imagem 48 – Simulação da alteração de pressão estática para uma altitude de mil metros.



Fonte: O Autor.

Imagem 49 – Simulação da variação de altitude com base nas pressões estimadas.



Fonte: O Autor.

7.2. Teste do acelerômetro

Inicialmente, procurou-se uma biblioteca para facilitar a utilização do acelerômetro (Imagem 50). Com isso, optou-se pela biblioteca do "Tockn", que simplifica significativamente o uso do sensor, realizando a calibração do giroscópio, convertendo os valores de aceleração em função da gravidade e os dados de rotação para graus por segundo. Além disso, possibilita obter os valores de posição angular por três métodos distintos: filtro de Kalman, integração da velocidade angular do giroscópio e decomposição da aceleração da gravidade nos três eixos.

A ideia inicial era utilizar os dados de atitude do acelerômetro como uma redundância para controlar o foguete, em caso de falha do barômetro, baseando-se em dados de lançamentos e variações angulares de foguetes reais. Dada a rotação nos eixos longitudinal e transversal do foguete, analisaram-se os três métodos para obtenção da atitude angular. Cada método apresentou vantagens e desvantagens.

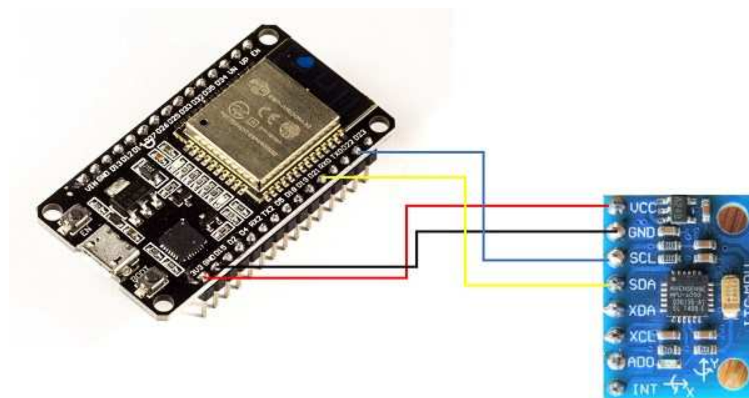
Primeiramente, testou-se a obtenção dos ângulos através da decomposição da aceleração da gravidade nos três eixos. Essa decomposição permite calcular as posições angulares do foguete por meio de cálculos trigonométricos. No entanto, surgiram dois problemas: um eixo sempre seria perpendicular à gravidade, impossibilitando a leitura completa da angulação; e a precisão da decomposição só é garantida para valores de aceleração próximos à gravidade, o que pode causar problemas no início do voo devido às altas acelerações do empuxo do foguete.

O segundo método testado foi a integração dos valores de velocidade angular do giroscópio. Este método consegue obter os dados de atitude nos três eixos, independentemente da sua direção no espaço. Inicialmente, pareceu ser uma boa alternativa, sendo possível compensar a rotação longitudinal do foguete combinando os dados angulares dos três eixos. Contudo, esse método mostrou-se preciso apenas em baixas variações angulares. Em testes com múltiplas rotações, a integração apresentou erros devido ao ruído crescente, o que poderia resultar em leituras errôneas e afetar a evolução correta da máquina de estados.

O terceiro método foi o filtro de Kalman (KALMAN, 1960), que utiliza a fusão de sensores para obter os dados de atitude. Este método combina a decomposição da aceleração da gravidade nos três eixos com a integração do giroscópio. Embora tenha o potencial de compensar as limitações dos dois métodos anteriores, tornou-se inviável devido ao tempo de resposta necessário para corrigir os erros acumulados na integração e os ruídos nas leituras.

Portanto, o controle baseado em dados cinemáticos, embora inicialmente pareça simples, apresenta desafios devido ao ruído nas leituras e às rotações longitudinais, adicionando complexidade à obtenção precisa dos ângulos de atitude. Para o modelo de acelerômetro proposto e nos cenários considerados, esses métodos não se mostraram confiáveis para controlar eventos, sendo recomendada a abordagem barométrica para o controle do foguete.

Imagem 50 – Conexão do MPU6050 no esp32.



Fonte: Interfacing ESP32 with MPU6050 [03 de agosto de 2024]. Disponível em: www.tutorialspoint.com.

7.3. Testes de tratamento de erros

Durante o desenvolvimento do projeto, foram identificados diferentes tipos de erros e métodos para tratá-los. Esses erros foram classificados em duas categorias: internos do sistema

e de natureza eletrônica. Erros internos incluem a falta de arquivos de parâmetros e a duplicação de arquivos de dados na memória. Já os erros eletrônicos abrangem componentes desconectados ou sem comunicação.

Na fase de prototipagem para testes, verificou-se que não seria possível tratar todos os erros de maneira uniforme devido à complexidade dos componentes, às bibliotecas utilizadas e ao tipo de controle implementado. Para componentes de conexão simples, como os de sinalização sonora e luminosa (*buzzer* e LED) e a chave seletora da IHM, que são controlados apenas por portas digitais, a verificação de conexão não é viável. Já para componentes que utilizam o protocolo I2C, como o display, o acelerômetro e o barômetro, a comunicação com eles é verificada através dos seus endereços I2C para garantir a conexão adequada. A conexão e a presença do cartão SD também são checadas por meio de testes de comunicação entre o microcontrolador e o cartão de memória.

Os erros de software estão relacionados à mudança de parâmetros e à existência de arquivos. Esses parâmetros possuem limites de funcionamento internos e são reescritos com base em valores padrão do software, caso estejam ausentes ou incorretos. Em caso de erro, o sistema emite sinais sonoros e luminosos contínuos para alertar o usuário. Se o sistema inicializar corretamente, ele emitirá sinais intermitentes de som e luz.

Para facilitar a identificação de erros, ao conectar o computador de voo ao software Arduino IDE, um log detalha as ações realizadas pelo sistema, permitindo ao usuário identificar rapidamente onde ocorreu o erro.

7.4. Teste da IHM

Para testar a Interface Homem-Máquina (IHM), inicialmente foi analisado o funcionamento da chave rotativa para compreender seu mecanismo, dado que este componente seria responsável por todas as interações com o display, incluindo a movimentação do cursor para seleção de opções, realização de cliques e modificação de valores. Após estudar os sinais da chave rotativa e configurar todos os comandos necessários, aplicou-se esse conhecimento no display.

Em seguida, a lógica de funcionamento do display foi estudada, testando-se os comandos disponibilizados pela biblioteca utilizada para seu controle. Inicialmente, comandos simples foram testados, como limpar a tela, escrever textos e gerar formas básicas. Logo depois, foi idealizado um modelo de telas de menus que se encaixasse no espaço disponível de maneira clara e compreensível.

Seguiu-se então com a implementação dos menus, através da criação de algoritmos de controle baseados na tela em que o usuário se encontrava. Dessa forma, foi possível construir uma IHM funcional que englobasse todas as funções propostas, garantindo uma interface intuitiva e eficaz para o usuário. As Imagens 51 a 57 apresentam telas da IHM, contudo algumas imagens ficaram de difícil visualização devido problemas de foco da câmera utilizada.

Imagem 51 – Tela inicial.

Fonte: O Autor.

Imagem 52 – Tela de seleção do método de acionamento.

Fonte: O Autor.

Imagem 53 – Tela de definição da altitude de recuperação.

Fonte: O Autor.

Imagem 54 – Tela de definição do percentual de altitude de recuperação.

Fonte: O Autor.

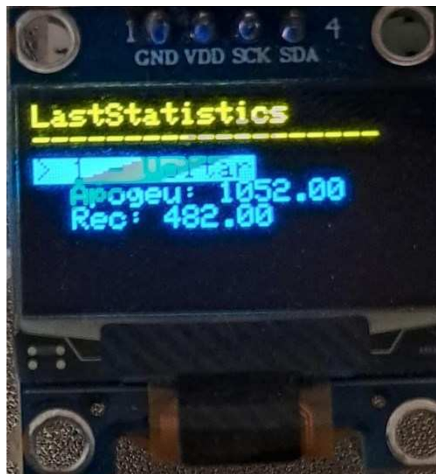
Imagem 55 – Tela de leitura dos sensores.

Fonte: O Autor.

Imagem 56 – Tela de leitura do acelerômetro

Fonte: O Autor.

Imagem 57 – Estatísticas do último lançamento na memória.



Fonte: O Autor.

7.5. Teste da máquina de estados

Foram estabelecidos cinco estados para o funcionamento do sistema conforme a Tabela 3:

Tabela 3 – Estados do sistema.

Estado	Descrição	Condição de saída
"estadoPreLancamento"	Estado inicial do sistema, nesse estado é possível acessar a IHM e verificar as sinalizações	Variação positiva de altitude maior que 50 metros.

	sonoras\luminosas do sistema. Os sensores são verificados continuamente	
“estadoSubindo”	Durante esse estado os dados são armazenados no sdcard até alcançar o apogeu e inverter o sentido de movimento.	A diferença entre a altitude relativa e a média móvel das altitudes torna-se negativa.
“estadoDescendo”	Os dados continuam sendo salvos até as condições de recuperação serem alcançadas, após isso é executado novamente até as condições de pouso.	1º - Alcançar a condição de acionamento definida na IHM, mudando para o estado de recuperação. 2º - Alcançar as condições de pousou (altitude relativa menor do que 20 metros), mudando para o estado de pouso.
“estadoRecuperacao”	A carga pirotécnica é acionada e retorna para o estado anterior.	-
“estadoPouso”	Finaliza a gravação de dados, confirma a conclusão do voo no arquivo “launch.txt” e entra em hibernação.	-

Fonte: O Autor.

Para testar a máquina de estados, foi implementado a simulação de variação de altitude empregada para substituir o barômetro, já mencionada anteriormente. Contudo, foi implementado um certo ruído, que será comentado mais à frente, considerando os ruídos e pequenas variações lidas por um barômetro digital real, a fim de aproximar os testes de condições mais condizentes com a realidade.

Foram realizados diversos testes na máquina de estados para verificar se a lógica adotada estava funcionando corretamente. Após ajustes finos nas condições de mudança de estados, a o sistema se mostrou completamente funcional e confiável, até mesmo em casos em que eram incluídas leituras extremamente ruidosas.

7.6. Testes das condições de acionamento da carga pirotécnica

A definição de condições robustas para o acionamento da carga pirotécnica durante o lançamento de um foguete é essencial para garantir uma recuperação segura e eficiente. A precisão no momento de acionamento é crucial para evitar falhas que possam resultar em danos ao equipamento ou riscos à segurança. Assim, a determinação do instante exato para o acionamento deve ser baseada em eventos bem definidos do perfil de voo, assegurando que o sistema de recuperação seja ativado apenas nas condições apropriadas.

Para testar a robustez do sistema de acionamento, foram realizadas simulações utilizando o software MATLAB. O objetivo foi modelar a trajetória parabólica do foguete e incluir variações de altitude com um ruído de ±30 metros, uma faixa maior do que a normalmente captada em testes da EPTA (±5 metros). Essas simulações permitiram avaliar como o sistema reage a variações significativas de altitude, testando a resiliência do algoritmo de acionamento em condições tanto próximas à realidade quanto extremas.

Os dados foram simulados utilizando o mesmo processo empregado para simular o barômetro e a variação de altitude, mas com a inclusão de ruídos aleatórios dentro de uma faixa ampla de valores. Essa abordagem foi adotada devido às diversas situações que podem ocorrer em um cenário real, como variações bruscas de temperatura e pressão dentro da fuselagem do foguete antes do lançamento e durante a verificação dos elementos internos, o que pode acarretar leituras errôneas e acidentes no acionamento de eventos.

Para uma análise mais realista, foram realizadas simulações no software *OpenRocket* pela EPTA para simular um perfil de lançamento completo, incluindo a abertura do paraquedas no apogeu. O *OpenRocket* permite a modelagem detalhada do comportamento do foguete, considerando fatores como empuxo, arrasto e mudanças nas condições atmosféricas ao longo do voo.

Essas simulações testaram as condições de acionamento da recuperação, analisando como o sistema responde a um perfil de voo mais complexo e realista. O *OpenRocket* gera um conjunto de dados que inclui a trajetória completa do foguete, desde o lançamento até o apogeu e a descida, permitindo validar o algoritmo de acionamento em condições que se aproximam da realidade operacional. Os dados do *OpenRocket* foram então importados para o *MATLAB*, onde foi acrescentado um ruído para testar a robustez da condição de acionamento.

A condição de acionamento foi definida pela comparação da altitude relativa com a média móvel de um certo número de leituras de altitude anteriores. Esse método funciona como uma derivada, onde o sinal pode indicar o crescimento ou decrescimento do perfil médio da

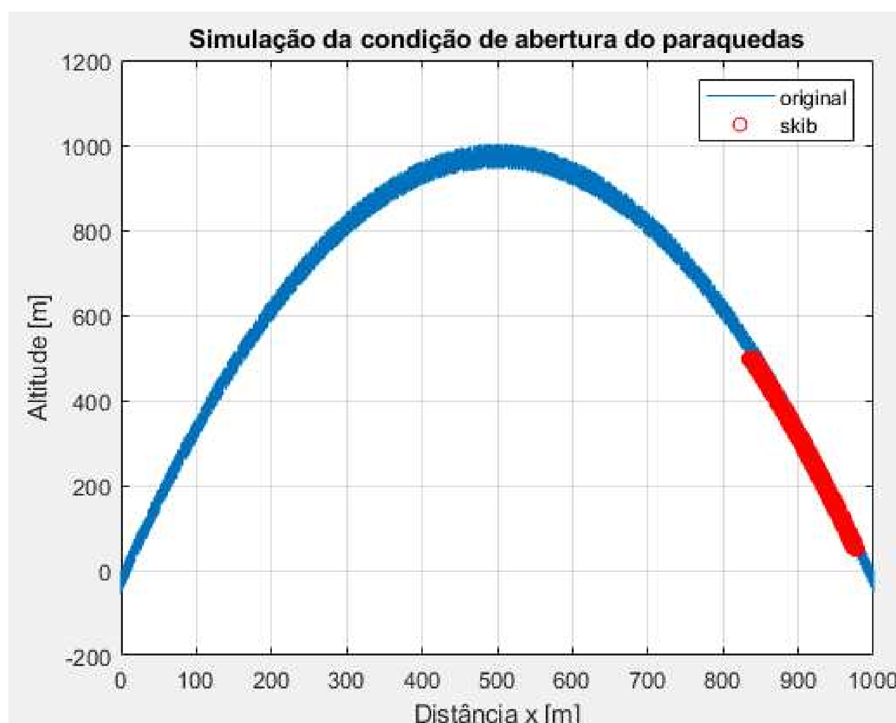
variação de altitude. Durante a subida, o sinal é positivo, pois a altitude atual é maior do que a média das altitudes anteriores. Na descida, o sinal é negativo, já que a altitude atual é menor do que a média das altitudes anteriores. Assim, combinando a condição de um sinal negativo com a altitude dentro de uma faixa escolhida, o sistema identifica o momento adequado para acionar a recuperação, resistindo a leituras ruidosas e a condições mais severas de temperatura e pressão.

Os resultados das simulações indicaram que o sistema de recuperação é capaz de lidar eficazmente com as variações de altitude simuladas, acionando a carga pirotécnica de forma precisa nos momentos adequados. A adição do ruído à trajetória parabólica demonstrou que o sistema mantém sua robustez, enquanto a simulação com o perfil de lançamento no *OpenRocket* confirmou que o algoritmo de acionamento é confiável em cenários mais complexos.

Dessa forma, a combinação das simulações em *MATLAB* e *OpenRocket* forneceu uma base sólida para validar as condições de acionamento da carga pirotécnica, garantindo a segurança e eficiência da recuperação do foguete em diferentes situações de voo.

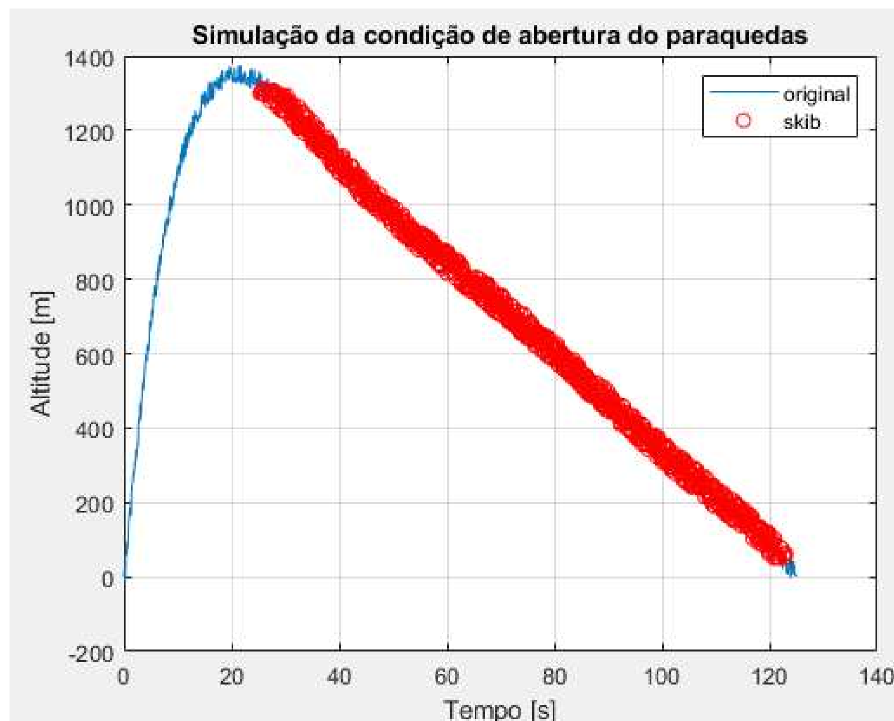
As imagens 58 e 59 ilustram as simulações realizadas, lembrando que o tempo de acionamento é prolongado devido à leve demora para realizar a queima da carga pirotécnica.

Imagem 58 – Simulação de variação de altitude com ruído e acionamento da recuperação em 500 m (*MATLAB*).



Fonte: O Autor.

Imagem 59 – Simulação de lançamento com dados fornecidos pelo *OpenRocket* com acionamento da recuperação após o apogeu.



Fonte: O Autor.

7.7. Simulação do funcionamento do sistema completo

Durante a inicialização do sistema, o primeiro passo é verificar a conexão com os sensores e componentes, como o SD card (Imagem 60).

Imagem 60 – Verificação dos componentes.

```
15:53:10.773 -> 9�2��L�1��□(�□z t�mDisplay iniciado...
15:53:11.097 -> Verificando MPU6050 no endereco: 0x68
15:53:11.097 -> MPU6050 conectado.
15:53:11.097 -> Verificando OLED Display no endereco: 0x3C
15:53:11.143 -> OLED Display conectado.
15:53:11.234 -> MPU6050 inicializado...
15:53:11.234 -> ICP10100 inicializado...
15:53:11.234 -> Inicializando cartão SD...
15:53:11.280 -> Inicialização do Sdcard feita.
```

Fonte: O Autor.

Após essa verificação inicial, o sistema confere a presença de arquivos de parâmetros na memória (Imagem 61). Se esses arquivos forem encontrados, o sistema importa as definições contidas neles; caso contrário, ele utiliza os dados predefinidos no software e cria novos

arquivos de parâmetro. Neste estágio inicial, um ruído de ±2 metros de altitude é gerado para indicar que o sistema foi ligado e está em repouso.

Imagem 61 – Verificação dos arquivos de parâmetros.

```
10:27:39.670 -> Verificando parametros...
10:27:39.670 -> Arquivo não encontrado, criando e salvando valores...
10:27:39.757 -> Valores salvos com sucesso.
10:27:39.757 -> Verificando launch...
10:27:39.835 -> Arquivo não encontrado, criando e salvando valores...
10:27:39.870 -> Valores salvos com sucesso.
```

Fonte: O Autor.

Uma vez que a IHM recebe um comando específico para iniciar a simulação de dados de altitude, incluindo ruídos, o sistema passa a calcular um perfil de altitude parabólico (Imagem 62). Esse perfil simula a variação de altitude no sistema, e após cinco segundos, os dados de altitude são gerados, permitindo a evolução da máquina de estados.

Imagem 62 – Equação para simulação da variação de altitude.

```
15:48:35.042 -> 0.00 0
15:48:35.080 ->
15:48:35.080 -> y = (-0.40)*x^2 + (40.00)x
15:48:40.145 -> 16.00 0
```

Fonte: O Autor.

No estado inicial, denominado "estadoPreLancamento", o sistema detecta uma variação significativa de 50 metros na altitude. Nesse momento, é criado um arquivo de armazenamento de dados denominado "launch.txt", onde os dados de estado são salvos. Essa ação garante que, em caso de desligamento acidental durante o voo, o sistema possa retomar a execução a partir do ponto onde parou. Simultaneamente, o log do sistema sinaliza que o foguete está subindo, e o sistema continua a armazenar dados (Imagem 63).

Imagem 63 – Criação do arquivo de armazenamento de dados de voo e detectando subida.

```
15:48:40.145 -> 45.84 0
15:48:40.184 -> 57.81 0
15:48:40.184 -> Dados em: /dados(0).txt
15:48:40.184 -> Valores salvos com sucesso.
15:48:40.267 -> SUBINDO...
15:48:40.267 -> 46.71 1
15:48:40.267 -> 19.61 1
```

Fonte: O Autor.

Ao alcançar o apogeu, o sistema detecta um sinal negativo na comparação da altitude relativa com a média móvel das altitudes anteriores, indicando que o foguete começou a descer (Imagem 64). Nesse ponto, o sistema muda para o estado "estadoDescendo", salva o estado no arquivo "launch.txt" e sinaliza o início do primeiro percurso da descida, continuando a salvar dados até que as condições de recuperação sejam atendidas.

Imagem 64 – Detectando a primeira parte da descida.

```
15:48:46.749 -> 1042.35 1
15:48:46.749 -> 984.45 1
15:48:46.879 -> Valores salvos com sucesso.
15:48:46.879 -> DESCENDO1...
15:48:46.879 -> 1008.12 2
15:48:46.879 -> 1025.52 2
```

Fonte: O Autor.

Quando a condição de acionamento da recuperação é detectada, o sistema entra no estado "estadoRecuperacao" (Imagem 65). Nesse estado, ele salva as informações no arquivo "launch.txt" e aciona a carga pirotécnica, voltando em seguida ao estado de descida. Durante essa segunda parte da descida, o sistema continua a salvar dados até que as condições de pouso sejam atendidas.

Imagem 65 – Acionando a recuperação e identificando a segunda parte da descida.

```
15:48:50.622 -> 548.26 2
15:48:50.668 -> 545.44 2
15:48:50.715 -> 514.96 2
15:48:50.715 -> RECUPERACAO ATIVADA!
15:48:50.807 -> Valores salvos com sucesso.
15:48:50.807 -> DESCENDO2...
15:48:50.807 -> 537.72 2
15:48:50.807 -> 524.63 2
15:48:50.807 -> 496.21 2
```

Fonte: O Autor.

Finalmente, quando o foguete atinge as condições de pouso, o sistema muda para o estado "estadoPouso" (Imagem 66). Ele insere a conclusão do lançamento no arquivo "launch.txt" e entra em modo de hibernação para poupar energia até que o foguete seja recuperado.

Imagem 66 – Condições de pouso detectadas.

```
15:48:52.634 -> 22.94 2
15:48:52.634 -> 8.97 2
15:48:52.719 -> Valores salvos com sucesso.
15:48:52.719 -> POUSO...
```

Fonte: O Autor.

Após a realização dos testes iniciais, se os arquivos de parâmetros já estiverem presentes na memória, o sistema importará automaticamente esses dados durante os próximos lançamentos (Imagem 67). Esses parâmetros importados podem ser ajustados conforme necessário através da IHM, garantindo flexibilidade e precisão nas configurações do sistema para cada novo lançamento.

Imagem 67 – Resgatando os parâmetros da memória.

```
15:53:12.214 -> Verificando parametros...
15:53:12.214 -> Arquivo encontrado, lendo valores...
15:53:12.296 -> Valores lidos: metodo = 3, altitudeRef = 0.600000, altitudeRec = 500.000000
15:53:12.389 -> Verificando launch...
15:53:12.389 -> Arquivo encontrado, lendo valores...
15:53:12.421 -> Valores lidos: launch = 0, estado = 0
```

Fonte: O Autor.

Além disso, o sistema adota precauções para não sobrescrever arquivos de dados de lançamentos anteriores, renomeando adequadamente os novos arquivos de lançamentos, conforme a imagem 68.

Imagem 68 – Criando arquivo de dados no segundo teste.

```
15:53:30.460 -> 47.94 0
15:53:30.460 -> 2.85 0
15:53:30.460 -> 51.85 0
15:53:30.548 -> Dados em: /dados(1).txt
15:53:30.548 -> Valores salvos com sucesso.
15:53:30.548 -> SUBINDO...
15:53:30.593 -> 23.75 1
15:53:30.593 -> 31.69 1
15:53:30.593 -> 40.63 1
```

Fonte: O Autor.

8. DISCUSSÃO

Durante o desenvolvimento deste projeto, foram enfrentados diversos desafios, especialmente relacionados ao uso do acelerômetro MPU6050 para controle de voo. Embora amplamente utilizado devido ao seu custo relativamente baixo e boa sensibilidade, o MPU6050 apresentou limitações significativas em termos de precisão e estabilidade dos cálculos para aplicações de controle de voo de foguetes. O erro acumulado crescia exponencialmente, e as correções através de métodos como o filtro de Kalman mostraram-se demasiadamente lentas para esta aplicação específica. A principal dificuldade encontrada foi a alta sensibilidade do sensor a ruídos e vibrações, comprometendo a confiabilidade dos dados durante o lançamento. Além disso, a necessidade de filtros complexos para eliminar esses ruídos, aliada à limitada capacidade de processamento do microcontrolador utilizado, impôs desafios adicionais ao projeto. Devido a essas dificuldades, optou-se por utilizar apenas o barômetro ICP10100 para controle dos eventos de lançamento, pois este sensor se mostrou mais adequado para fornecer leituras precisas de altitude, essenciais para a ativação dos mecanismos de recuperação do foguete no momento correto.

Durante o desenvolvimento do projeto, houve um aumento significativo nos custos devido às novas taxas de importação no Brasil e à alta do dólar. Segundo dados da Receita Federal e da Secretaria da Fazenda, o processo de importação inclui uma série de impostos além do Imposto de Importação, que foi aumentado em 2024 para produtos eletrônicos, passando de 20% para 25%. Além disso, outros tributos também incidem sobre os produtos importados, como:

- **ICMS (Imposto sobre Circulação de Mercadorias e Serviços)**: varia de estado para estado, com alíquotas que vão de 17% a 19%. Esse imposto é calculado sobre o valor total da mercadoria, incluindo o preço do produto, o valor do frete e outros impostos.
- **PIS/COFINS (Programa de Integração Social/Contribuição para o Financiamento da Seguridade Social)**: juntos, somam 9,25% sobre o valor da mercadoria.
- **Imposto sobre Produtos Industrializados (IPI)**: que pode variar dependendo do tipo de produto, com uma alíquota que normalmente fica em torno de 15%.

Além disso, a alta do dólar, que variou entre R$ 4,80 e R$ 5,40 durante o período de desenvolvimento, aumentou o custo dos componentes, impactando diretamente a viabilidade econômica do projeto. Esses fatores elevaram o custo final de produção do sistema para aproximadamente 120 dólares, um valor comparável ao de sistemas comerciais prontos para

uso (COTS), que variam entre 50 e 120 dólares (conforme os preços de fabricantes como a PerfectFlite e MissileWorks), aproximadamente. Esse aumento de custo limitou a competitividade do sistema em relação às soluções comerciais disponíveis no mercado.

Para aprimorar o sistema e torná-lo mais eficiente, algumas mudanças naturais poderiam ser implementadas em futuros desenvolvimentos. Uma das principais melhorias seria a realização de testes com lançamentos reais de baixa altitude (≤50 m), permitindo uma validação mais precisa e prática das funcionalidades do sistema. Outra mudança importante seria a elaboração de um circuito que permitisse a remoção do display, utilizando-o apenas quando necessário para realizar alterações nas configurações. Esse circuito poderia incluir um jumper para informar ao sistema se a presença do display deveria ser verificada ou não, contribuindo para a redução do tamanho e massa da PCB, fatores cruciais para a estabilidade do foguete.

Adicionalmente, componentes poderiam ser posicionados em ambos os lados da PCB para otimizar o espaço disponível e reduzir ainda mais o tamanho e a massa do sistema. Essas modificações são essenciais para melhorar a estabilidade do foguete durante o voo. Além disso, poderiam ser adicionadas mais opções à IHM, incluindo configurações para acionamento de dois estágios de recuperação, acrescentando uma saída física adicional para conexão de um segundo circuito pirotécnico. Outra funcionalidade interessante a ser incluída na versão final poderia ser a manutenção da opção de simulação de lançamento dentro do sistema, com a possibilidade de modificar os parâmetros da simulação, um recurso valioso para testes e ajustes finos sem riscos de perdas catastróficas de material.

Essas discussões e propostas de melhorias estão diretamente relacionadas aos resultados e análises apresentados ao longo deste trabalho. A implementação de testes reais e a otimização do hardware são passos cruciais para a evolução do sistema de controle de voo, buscando sempre o equilíbrio entre custo, eficiência e segurança. O desenvolvimento contínuo e a adaptação às condições do mercado são fundamentais para garantir a viabilidade e a competitividade do sistema em futuras aplicações.

9. CONCLUSÃO

O desenvolvimento deste projeto de um computador de voo para foguetemodelismo revelou-se desafiador, mas repleto de aprendizados e conquistas. Ao longo do trabalho, foram enfrentadas diversas dificuldades técnicas e econômicas, cada uma contribuindo para o amadurecimento e refinamento das soluções propostas.

Um dos principais desafios técnicos foi o uso do acelerômetro MPU6050 para controle de voo. Apesar de seu baixo custo e boa sensibilidade, o sensor mostrou-se inadequado devido à sua alta sensibilidade a ruídos e vibrações, comprometendo a precisão e estabilidade dos dados. A utilização do filtro de Kalman, embora eficaz em outras aplicações, demonstrou ser demasiadamente lenta para os requisitos específicos deste projeto. Em resposta a esses desafios, optou-se pelo barômetro ICP10100, que se provou mais confiável para fornecer leituras precisas de altitude, essenciais para o controle dos eventos de lançamento.

Economicamente, o projeto enfrentou um aumento significativo nos custos de produção devido às novas taxas de importação no Brasil e à alta do dólar. O custo final de produção do sistema foi aumentado, aproximando-se do valor de alguns sistemas comerciais prontos para uso. Este fator ressaltou a importância de considerar variáveis econômicas e políticas na gestão de projetos tecnológicos, reforçando a necessidade de flexibilidade e adaptação às condições do mercado.

A utilização de algoritmos para simulação mostrou-se imprescindível para todos os testes e definição de parâmetros durante o desenvolvimento do projeto, evitando testes em campo potencialmente destrutivos. Além disso, a reestruturação do circuito, visando à remoção do display para reduzir o tamanho e a massa da PCB, e a implementação de componentes em ambos os lados da placa são mudanças que melhorariam a estabilidade do foguete durante o voo.

A adição de novas funcionalidades à Interface Homem-Máquina (IHM), como configurações para acionamento de dois estágios de recuperação e a opção de modificar parâmetros da simulação de lançamento, são melhorias que aumentariam a versatilidade e a eficácia do sistema, permitindo um controle mais detalhado das etapas de voo.

Este trabalho forneceu uma base considerável para o desenvolvimento de sistemas aviônicos para foguetemodelismo, com avanços significativos tanto no projeto de hardware quanto de software. O hardware foi projetado para ser simples e eficiente, enquanto o software foi desenvolvido com foco na robustez e precisão, utilizando algoritmos de filtragem e controle avançados. A experiência adquirida e as soluções implementadas ao longo deste projeto

mostram que os objetivos inicialmente estabelecidos foram plenamente alcançados, fornecendo um guia claro e didático para o desenvolvimento de sistemas aviônicos de baixo custo e funcionalidade elevada para foguetemodelismo.

A experiência adquirida e as soluções implementadas ao longo deste projeto são um importante ponto de partida para futuras pesquisas e desenvolvimentos de cunho acadêmico. A contínua evolução e adaptação do sistema de controle de voo, alinhada às necessidades do mercado e às condições tecnológicas, será fundamental para assegurar a competitividade e a eficácia desta tecnologia em aplicações futuras. Este projeto não só contribuiu para o avanço técnico no campo do foguetemodelismo em nível universitário, mas também destacou a importância de uma abordagem integrada e adaptativa em projetos de engenharia.

10. REFERÊNCIAS

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 10520: informação e documentação: citações em documentos: apresentação. Rio de Janeiro: ABNT, 2002.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 14724: informação e documentação: trabalhos acadêmicos: apresentação. Rio de Janeiro: ABNT, 2011.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 6024: informação e documentação: numeração progressiva das seções de um documento: apresentação. Rio de Janeiro: ABNT, 2012a.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 6027: informação e documentação: sumário: apresentação. Rio de Janeiro: ABNT, 2012b.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. NBR 6028: informação e documentação: resumo: apresentação. Rio de Janeiro: ABNT, 2003.

DOEBELIN, E. O. Measurement Systems: Application and Design. 4th ed. McGraw-Hill, 1990.

EMBARCADOS. Como fazer calibração de Sensores na prática. Disponível em: https://embarcados.com.br/calibracao-de-sensores-na-pratica/. Acesso em: 02 jul. 2024.

EMBARCADOS. Comunicação I2C. Disponível em: https://embarcados.com.br/comunicacao-i2c/. Acesso em: 18 out. 2024.

EMBARCADOS. Introdução à Calibração de Sensores. Disponível em: https://embarcados.com.br/introducao-a-calibracao-de-sensores/. Acesso em: 02 jul. 2024.

ESPRESSIF SYSTEMS. Esp32_datasheet_en.pdf. Disponível em: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf. Acesso em: 02 jul. 2024.

GUIMARÃES, Flávio. Conheça a História do Arduino #QuartaMaker. YouTube, 15 set. 2021. Disponível em: https://www.youtube.com/watch?v=VXgb9Am3GAo. Acesso em: 18 out. 2024.

HELFRICK, Albert. Principles of Avionics. 10th ed. Bloomington: Avionics Communications Inc., 2012.

IBGE. Normas de apresentação tabular. 3rd ed. Rio de Janeiro: IBGE, 1994.

KALMAN, R. E. A New Approach to Linear Filtering and Prediction Problems. Transactions of the ASME - Journal of Basic Engineering, Series D, v. 82, p. 35-45, 1960.

NORTHROP, Robert B. Introduction to Instrumentation and Measurements. 3rd ed. Boca Raton: CRC Press, 2014.

NXP SEMICONDUCTORS. The I2C Bus Specification. Disponível em: https://www.nxp.com/docs/en/user-guide/UM10204.pdf. Acesso em: 02 jul. 2024.

TDK CORPORATION. Ds-000186-icp-101xx.pdf. Disponível em: https://product.tdk.com/system/files/dam/doc/product/sensor/pressure/capacitive-pressure/data_sheet/ds-000186-icp-101xx.pdf. Acesso em: 02 jul. 2024.

TDK CORPORATION. Mpu-6000-datasheet1.pdf. Disponível em: https://product.tdk.com/system/files/dam/doc/product/sensor/mortion-inertial/imu/data_sheet/mpu-6000-datasheet1.pdf. Acesso em: 02 jul. 2024.

TEXAS INSTRUMENTS. I2C Bus Technical Overview and Frequently Asked Questions. Disponível em: https://www.ti.com/lit/an/slva704/slva704.pdf. Acesso em: 02 jul. 2024.

TEXAS INSTRUMENTS. SPI User Guide. Disponível em: https://www.ti.com/lit/ug/sprugp2a/sprugp2a.pdf. Acesso em: 18 out. 2024.

WIKIPEDIA. Serial Peripheral Interface. Disponível em: https://en.wikipedia.org/wiki/Serial_Peripheral_Interface. Acesso em: 18 out. 2024.

APÊNDICE

A seguir, estão os códigos desenvolvidos durante a realização desse trabalho. Inicialmente, serão apresentados os arquivos necessários para o funcionamento pleno e completo do sistema (apêndices A ao E). Em seguida, os arquivos que devem ser substituídos para a realizar a simulação de lançamento sem a presença de um barômetro (apêndices F e G). Vale ressaltar que os arquivos devem estar em uma mesma pasta para o correto funcionamento do sistema.

APÊNDICE A – CodigoTCC.ino (Arquivo para a execução do sistema)

```
//DECLARACAO DE VARIAVEIS
//Variaveis de parametros
float altitudeMin = 50.0;  //valor padrao (menor altitude e incremento da mudanca de altitude de acionamento)
float altitudeRef = 0.6;   //valor de referencia para acionamento %
float altitudeRec = 500.0; //valor de altitude para abertura da recuperacao
float altitudeMax = 0.0;   //valor maximo de altitude relativa
float lastApogeu = 0.0;    //guarda o valor do apogeu do ultimo lançamento
float lastRec = 0.0;       //guarda o valor da altitude de recuperacao do ultimo lancamento
byte estado = 0;           //variavel para controle do estado
bool launch = false;       //verifica se o lancamento anterior foi concluido

//definir constantes para as IO's
const int buzzer = 15;  //porta para controle do buzzer
const int led = 2;      //porta para controle do led
const int skib = 4;     //porta para controle do acionamento skib

//Importando os estados do voo
extern void estadoPreLancamento();
extern void estadoSubindo();
extern void estadodescendo();
extern void estadoRecuperacao();
extern void estadoPouso();

//Importando as funcoes de controle da IHM
extern void iniciadisplay();
extern void displayIHM();

//Importando as funcoes de controle do SDcard
extern void iniciaSdcard();
extern void verificaCache();
extern void verificaParametros();

//Importando as funcoes para inicializacao dos sensores
extern void iniciaSensores();
```

```
void setup() {
  Serial.begin(9600);  //iniciando a comunicacao serial
  // Inicialização do sistema
  pinMode(led, OUTPUT);   //inicia led
  pinMode(skib, OUTPUT);  //inicia porta de controle do skib
  pinMode(buzzer, OUTPUT); //inicia buzzer

  ledcSetup(0, 1000, 10);    //Atribuimos ao canal 0 a frequencia de 1000Hz com resolucao de
10bits.
  ledcAttachPin(buzzer, 0);  //Atribuimos o pino do buzzer ao canal 0.

  noTone(buzzer);  //Inicia o buzzer mudo

  iniciadisplay();
  iniciaSensores();
  iniciaSdcard();
  //Define uma altitude de referencia para calculo de altitude relativa para os calculos
  calculaAltitudeReferencia();
  // Verificação dos arquivos de parâmetros
  verificaParametros();
}

void loop() {
    switch (estado) {
      case 0:
        estadoPreLancamento();
        //Controla a IHM
        displayIHM();
        break;

      case 1:
        estadoSubindo();
        break;

      case 2:
        estadodescendo();
        break;

      case 3:
        estadoRecuperacao();
        break;
      case 4:
        estadoPouso();
        break;

      default:
        break;
    }
}
```

APÊNDICE B – IHM.ino (Controle do display)

```
#include <Adafruit_GFX.h>      //biblioteca para controle do display
#include <Adafruit_SSD1306.h>  //bibioteca para controle do display

extern byte estado; //estados do sistema

//Importando variaveis dos sensores
//Acelerometro
extern float acX;
extern float acY;
extern float acZ;
extern float velGX;
extern float velGY;
extern float velGZ;
extern float angX;
extern float angY;
extern float angZ;
//Barometro
extern float altitudeRec;
extern float altitudeMin;
extern float altitudeRef;
extern float lastApogeu;
extern float lastRec;
extern float altitude;
extern float temperature_C;
extern float pressure_P;
//Sinalizacoes
extern const int led;
extern const int buzzer;

//Prototipo das funcoes
void displayIHM();
void iniciadisplay();
extern void leituraDados();
```

```
// Define o pino do botão subir, confirmar e descer
const int encAPin = 12,  // pino do Encoder A
  encBPin = 13,          // pino do Encoder B
  bttPin = 14;           // pino do botao

// Matrizes construturas do menu
const char menutitulos[][20] = { "Menu princiapl", "Acionamento", "Sensores",
"LastStatistics" };
const char menuOptions[][4][20] = { { "1 - LaunchMode", "2 - Acionamento", "3 - Sensores",
"4 - LastStatistics" },  // opc == 0
                                    { "1 - Voltar", "2 - Altitude", "3 - %", "4 - Automatico" },    // opc == 1
                                    { "1 - Voltar", "2 - Acelerometro", "3 - Barometro", "" },      // opc == 2
                                    { "1 - Voltar", "Alt[m]:", "", "" },                            // opc == 3
                                    { "1 - Voltar", "Alt[%]:", "", "" },                            // opc == 4
                                    { "1 - Voltar", "A:", "G:", "*:" },                             // opc == 5
                                    { "1 - Voltar", "Temp[C]:  ", "Pres[Pa]: ", "Alt[m]:  " },      // opc == 6
                                    { "1 - Voltar", NULL, NULL, NULL } };                           // opc == 7

//Variaveis para controle da IHM
int numOptions = 4;            //controla o espaço de movimento do cursor
int menu = 0;                  //indica o menu exibido na parte superior
int opc = 0;                   //enumera as telas
int selectedIndex = 0;         //controla o item selecionado na tela
bool flag = false;             //verifica se houve algum comando
bool displayON = false;        //liga e desliga a tela
int tempoSTB = 30 * 1E3;       //tempo para a IHM entrar em stand-by [s]
bool editandoValores = false;  //controla o momento de controlar o menu e alterar valores

//Variaveis para controle do metodo de acionamento
int metodo = 3;  // 1 -  altitude, 2 - por %, 3 - automatico

//Parametros do display
#define SCREEN_WIDTH 128
```

```
#define SCREEN_HEIGHT 64
#define SCREEN_ADDRESS 0x3C

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -
1);  //instaciando objeto para controle do display

void iniciadisplay() {
  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {  //verifica se o
display está conectado
    while (1)
      ;
  }
  Serial.println("Display  iniciado...");

  display.clearDisplay();              //limpa  a tela
  display.setTextColor(SSD1306_WHITE);  //define a cor da escrita
  display.setTextSize(1);              //define o tamanho da fonte
  display.display();                  //atualiza a tela

  pinMode(encAPin, INPUT_PULLUP);
  pinMode(encBPin, INPUT_PULLUP);
  pinMode(bttPin, INPUT_PULLUP);
  //Serial.println("Display  iniciado");
}

void displayIHM() {
  //desliga a tela caso saia do estado inicial do sistema
  if (estado != 0) {
    display.clearDisplay();
    display.display();
    displayON = false;
    return;
  }
  //Atualiza as leituras dos botoes
```

```
  bool encA = !digitalRead(encAPin);         //le o valor do encoder A
  bool encB = !digitalRead(encBPin);         //le o valor do encoder B
  bool btt = !digitalRead(bttPin);           //le o valor do botao
  static volatile bool LastA, LastB, LastBtt;  //guarda os estados anteriores dos encoders
  static volatile unsigned long int timer4;    //timer para atualizar a tela
  static volatile unsigned long int timer3;    //timer para stand-by display

  //entra em LaunchMode
  if ((btt == LOW && LastBtt == HIGH) && (menu == 0 && selectedIndex == 0 && opc ==
0)) {
    flag = false;
    displayON = false;
    display.clearDisplay();
    display.display();
  }
  //Controle da tela de acionamento
  //opcao voltar [Menu principal <- Acionamento]
  if ((btt == LOW && LastBtt == HIGH) && (menu == 1 && selectedIndex == 0) && opc
== 1) {
    menu = 0;
    selectedIndex = 0;
    opc = 0;
    flag = true;
  }
  //Metodo de acionamento
  //opcao voltar [Acionamento <- AltAcionamento]
  if ((btt == LOW && LastBtt == HIGH) && (menu == 1 && selectedIndex == 0) && opc
== 3) {
    selectedIndex = 0;
    opc = 1;
    flag = true;
    numOptions = 4;
  }
  //define metodo de acionamento como alt
```

```
  if ((btt == LOW && LastBtt == HIGH) && (menu == 1 && selectedIndex == 1 && opc ==
1)) {
    menu = 1;
    selectedIndex = 0;
    opc = 3;
    flag = true;
    numOptions = 2;
    //metodo = 1;
  }

  //habilita a edicao da altitude de acionamento
  if ((btt == LOW && LastBtt == HIGH) && (menu == 1 && selectedIndex == 1 && opc ==
3)) {
    editandoValores = !editandoValores;
    if (editandoValores == false) {
      metodo = 1;
      salvarArquivo("/parametros.txt");
    }
  }

  //opcao voltar [Acionamento <- %]
  if ((btt == LOW && LastBtt == HIGH) && (menu == 1 && selectedIndex == 0) && opc
== 4) {
    selectedIndex = 0;
    opc = 1;
    flag = true;
    numOptions = 4;
  }
  //define metodo de acionamento como %
  if ((btt == LOW && LastBtt == HIGH) && (menu == 1 && selectedIndex == 2) && opc
== 1) {
    menu = 1;
    selectedIndex = 0;
    opc = 4;
```

```
    flag = true;
    numOptions = 2;
    //metodo = 2;
  }

  //habilita  a edicao da % de acionamento
  if ((btt == LOW && LastBtt == HIGH) && (menu == 1 && selectedIndex == 1 && opc ==
4)) {
    editandoValores = !editandoValores;
    if (editandoValores == false) {
      metodo = 2;
      salvarArquivo("/parametros.txt");
    }
  }

  //definir metodo de acionamento como automatico
  if ((btt == LOW && LastBtt == HIGH) && (menu == 1 && selectedIndex == 3) && opc
== 1) {
    flag = true;
    metodo = 3;
    altitudeRef = 0.5;
    salvarArquivo("/parametros.txt");
  }
  //Controle da tela de sensores
  //opcao volta [Menu principal <- sensores]
  if ((btt == LOW && LastBtt == HIGH) && (menu == 2 && selectedIndex == 0) && opc
== 2) {
    menu = 0;
    selectedIndex = 0;
    opc = 0;
    flag = true;
    numOptions = 4;
  }
```

```
//opcao voltar [sensores <- acelerometro]
if ((btt == LOW && LastBtt == HIGH) && (menu == 2 && selectedIndex == 0) && opc
== 5) {
  selectedIndex = 0;
  opc = 2;
  flag = true;
  numOptions = 3;
}

//opcao acelerometro [sensores -> acelerometro]
if ((btt == LOW && LastBtt == HIGH) && (menu == 2 && selectedIndex == 1) && opc
== 2) {
  selectedIndex = 0;
  opc = 5;
  flag = true;
  numOptions = 1;
}

//opcao voltar [sensores <- barometro]
if ((btt == LOW && LastBtt == HIGH) && (menu == 2 && selectedIndex == 0) && opc
== 6) {
  selectedIndex = 0;
  opc = 2;
  flag = true;
  numOptions = 3;
}

//opcao acelerometro [sensores -> barometro]
if ((btt == LOW && LastBtt == HIGH) && (menu == 2 && selectedIndex == 2) && opc
== 2) {
  selectedIndex = 0;
  opc = 6;
  flag = true;
  numOptions = 1;
```

```
}

//entra na tela acionamento
if ((btt == LOW && LastBtt == HIGH) && (menu == 0 && selectedIndex == 1)) {
  menu = 1;
  selectedIndex = 0;
  opc = 1;
  flag = true;
}

//entra na tela sensores
if ((btt == LOW && LastBtt == HIGH) && (menu == 0 && selectedIndex == 2)) {
  menu = 2;
  selectedIndex = 0;
  opc = 2;
  flag = true;
  numOptions = 3;
}

//volta  tela inicial <- LastStatistics
if ((btt == LOW && LastBtt == HIGH) && (menu == 3 && selectedIndex == 0 && opc ==
7)) {
  menu = 0;
  selectedIndex = 0;
  opc = 0;
  flag = true;
  numOptions = 4;
}

//entra na tela LastStatistics
if ((btt == LOW && LastBtt == HIGH) && (menu == 0 && selectedIndex == 3 && opc ==
0)) {
  menu = 3;
  selectedIndex = 0;
```

```
    opc = 7;
    flag = true;
    numOptions = 1;

    resgataLastStatistics(); //atualiza os valores do ultimo lancamento com base nos dados em
memoria
  }

  //controle da interacao da chave seletora com a IHM
  if ((millis()  - timer4 >= 30) && (LastA != encA || LastB != encB)) {  //verifica o seletor
    if (editandoValores == false) {
      if (encA == false && encB == true) {  // incrementa o seletor
        selectedIndex = (selectedIndex + 1) % numOptions;
        flag = true;
      } else if (encA == true && encB == false) {  // decrementa o seletor
        selectedIndex--;
        if (selectedIndex < 0) selectedIndex = numOptions - 1;
        flag = true;
      }
    } else {
      if (opc == 3) {                                   //mudando a altitude de acionamento da rec
        if (encA == false && encB == true) {                      // incrementa o seletor
          altitudeRec += altitudeRec < 3000 ? altitudeMin : 0.0;  //altitude maxima de
acionamento de 3km
          flag = true;
        } else if (encA == true && encB == false) {  // decrementa o seletor
          altitudeRec -= altitudeRec > altitudeMin ? altitudeMin : 0.0;
          if (altitudeRec < 50.0) altitudeRec = 50.0;
          flag = true;
        }
      }

      if (opc == 4) {                                //mudando o % de acionamento da rec
        if (encA == false && encB == true) {               // incrementa o seletor
```

```
        altitudeRef += altitudeRef < 0.95 ? 0.05 : 0.0;  //altitude maxima de acionamento de
3km
        flag = true;
      } else if (encA == true && encB == false) {  // decrementa o seletor
        altitudeRef -= altitudeRef > 0.5 ? 0.05 : 0.0;
        if (altitudeRef < 0.5) altitudeRef = 0.5;
        flag = true;
      }
    }
  }
}

//ativa o stand by
if (millis() - timer3 >= tempoSTB && displayON && opc != 5) {
  display.clearDisplay();
  display.display();
  //Serial.println("STB ON");
  displayON = false;
}

//constroi a tela
if (flag && (millis() - timer4 >= 80)) {                    //constroi apenas se alguma
mudanca for feita e apos um certo tempo entre a ultima atualizacao
  display.clearDisplay();                                  //limpa a tela
  display.setTextColor(SSD1306_WHITE);                           //define texto na cor
normal
  display.setCursor(0, 0);                                 //posiciona o cursor no inicio do
tela
  display.println(menutitulos[menu]);                           //escreve o nome do menu no
cabecalho da tela
  display.println("-------------------");                       //divide o cabecalho das opcoes
  for (int i = 0; i < (numOptions == 1 ? numOptions + 3 : numOptions); i++) {  // controla a
animacao do seletor
```

```
    if (i == selectedIndex) {                                  // faz o item selecionado inverter
as cores
      display.setTextColor(SSD1306_BLACK, SSD1306_WHITE);
      display.print("> ");
    } else {
      display.setTextColor(SSD1306_WHITE);
      display.print("  ");
    }

    if (opc == 1 && (metodo == i)) {  //escreve as opcoes destacando o metodo de
acionamento da recuperacao
      display.print(menuOptions[opc][i]);
      display.print(" (*)");
    } else {
      display.print(menuOptions[opc][i]);
    }

    if (opc == 5) {  //exibe os dados do acelerometro
      if (i == 1) display.print(String(acX) + ";" + String(acY) + ";" + String(acZ));
      if (i == 2) display.print(String(velGX) + ";" + String(velGY) + ";" + String(velGZ));
      if (i == 3) display.print(String(int(angX)) + ";" + String(int(angY)) + ";" +
String(int(angZ)));
    }

    if (opc == 6) {  //exibe os dados do barometro
      if (i == 1) display.print(pressure_P);
      if (i == 2) display.print(temperature_C);
      if (i == 3) display.print(altitude);
    }

    if (opc == 3) {  //exibe a altitudeRec na respectiva tela de acionamento
      if (i == 1) display.print(altitudeRec);
    }
```

```
      if (opc == 4) {  //exibe a % na respectiva tela de acionamento
        if (i == 1) display.print(altitudeRef);
      }

      if (opc == 7) { //exibe os dados do ultimo lancamento
        if (i == 1 && (lastApogeu == 0.0 || lastRec == 0.0)) {
          display.print("Sem lancamentos \n anteriores...");
        } else {
          if (i == 1) display.println("Apogeu: " + String(lastApogeu));
          if (i == 1) display.print("Rec: " + String(lastRec));
        }
      }

      display.println();
    }

    display.display();  //atualiza a tela

    if (opc != 5 && opc != 6) {
      flag = false;  //comeca a contar o timer para desligar a tela
    } else {
      flag = true;  //para poder atualizar os valores dos sensores no display
    }
    timer3 = millis();  //atualiza o timer do stand-by
    timer4 = millis();  //atualiza o timer do seletor
    displayON = true;
  }
  //guarda os valores anteriores do seletor
  LastA = encA;
  LastB = encB;
  LastBtt = btt;
}
```

APÊNDICE C – SDCard.ino (Controle da leitura e escrita na memória)

```
// BIBLIOTECAS
#include <SPI.h>     //biblioteca para controle da comunicao SPI
#include <SD.h>      //biblioteca para controle do sdcard
#include <String.h>  //biblioteca para manipular strings

void iniciaSdcard();
void verificaParametros();
void verificaCache();
void criaArqDados();

unsigned int timer2 = 0;    //timer para referencia de gravacao para os dados

File myFile;  //instaciando objeto para controle do sdcard

extern int metodo;
extern float altitudeRef;
extern float altitudeRec;
extern const int skib;
extern float altitude;
extern float pressure_P;
extern float lastApogeu;
extern float lastRec;
extern float altRef;
extern float acX;
extern float acY;
extern float acZ;
extern float velGX;
extern float velGY;
extern float velGZ;
extern float angX;
extern float angY;
extern float angZ;
extern byte estado;
extern bool launch;
extern bool flagEstadoDescendo;
//definir nome padrao dos arquivos (dados, cache, parametros)
String nomeArq;  //strng que guarda o nome do arquivo de dados

void iniciaSdcard() {  //estudar a possibilidade para trocar a saida de texto para o display
  Serial.println("Inicializando cartão SD...");
  if (!SD.begin(5)) {
    Serial.println("Falha na inicialização do sdcard!");
    while (1) {
      tone(buzzer, 1000, 200);
      digitalWrite(led, HIGH);
    }
  }
  Serial.println("Inicialização do Sdcard feita.");
}
```

```
void criaArqDados() {
  nomeArq = "/dados(0).txt";
  int contador = 0;
  while (SD.exists(nomeArq)) {
    // Se o arquivo já existe, incrementa o contador e tenta novamente
    contador++;
    nomeArq = "/dados(" + String(contador) + ").txt";
  }
  Serial.println("Dados em: " + nomeArq);
  // Abre o arquivo no modo de escrita
  myFile = SD.open(nomeArq, FILE_WRITE);
  myFile.println("Tempo[s]  Altitude[m] Pressao[Pa] Temperatur[C]  Ax Ay Az GyX GyY
GyZ AngX AngY AngZ Rec");
  myFile.close();
}

void verificaParametros() {
  Serial.println("Verificando parametros...");
  if (SD.exists("/parametros.txt")) {
    Serial.println("Arquivo encontrado, lendo valores...");
    lerArquivo("/parametros.txt");
  } else {
    Serial.println("Arquivo não encontrado, criando e salvando valores...");
    salvarArquivo("/parametros.txt");
  }

  Serial.println("Verificando launch...");
  if (SD.exists("/launch.txt")) {
    Serial.println("Arquivo encontrado, lendo valores...");
    lerArquivo("/launch.txt");
  } else {
    Serial.println("Arquivo não encontrado, criando e salvando valores...");
    salvarArquivo("/launch.txt");
  }
}

void lerArquivo(const char *path) {
  File file = SD.open(path);
  if (file) {
    String linha;
    int cont = 0;

    if (strcmp(path, "/parametros.txt") == 0) {
      while (file.available()) {
        linha = file.readStringUntil('\n');
        float valor = linha.toFloat();

        if (cont == 0) {
          metodo = int(valor);
```

```
        if (metodo == 3) altitudeRef = 0.5;
      } else if (cont == 1) {
        altitudeRef = valor;
      } else if (cont == 2) {
        altitudeRec = valor;
      }
      cont++;
    }
    file.close();
    Serial.printf("Valores lidos: metodo = %d, altitudeRef = %f, altitudeRec = %f\n", metodo,
altitudeRef, altitudeRec);

  } else if (strcmp(path, "/launch.txt") == 0) {
/**/
    while (file.available()) {
      linha = file.readStringUntil('\n');
      float valor = linha.toInt();

      if (cont == 0) {
        launch = valor;
      } else if (cont == 1 && launch) {
        estado = valor;
      }else if (cont == 2 && launch) {
        flagEstadoDescendo = bool(valor);
      }else if (cont == 2 && launch) {
        altRef = valor;
      }
      cont++;
    }

    file.close();
    Serial.printf("Valores lidos: launch = %d, estado = %d\n", launch, estado);

  } else if (strcmp(path, "/laststatistics.txt") == 0) {
    while (file.available()) {
      linha = file.readStringUntil('\n');
      float valor = linha.toInt();

      if (cont == 0) {
        lastApogeu = valor;
      } else if (cont == 1) {
        lastRec = valor;
      }
      cont++;
    }
  }

 } else {
   Serial.println("Falha ao abrir o arquivo para leitura");
 }
```

```
}

void resgataLastStatistics() {
  if (SD.exists("/laststatistics.txt")) {
    Serial.println("Arquivo encontrado, lendo valores...");
    lerArquivo("/laststatistics.txt");
  }
}

void salvarArquivo(const char *path) {
  File file = SD.open(path, FILE_WRITE);
  if (strcmp(path, "/parametros.txt") == 0) {
    if (file) {
      file.println(metodo);
      file.println(altitudeRef);
      file.println(altitudeRec);
      file.close();
      Serial.println("Valores salvos com sucesso.");
    } else {
      Serial.println("Falha ao abrir o arquivo para escrita");
    }
  } else if (strcmp(path, "/launch.txt") == 0) {
    if (file) {
      file.println(launch);
      file.println(estado);
      file.println(flagEstadoDescendo);
      file.println(altRef);
      file.close();
      Serial.println("Valores salvos com sucesso.");
    } else {
      Serial.println("Falha ao abrir o arquivo para escrita");
    }
  }
}

void salvaDados() {
  myFile = SD.open(nomeArq, FILE_APPEND);
  myFile.println(String(millis() - timer2) + " " + String(altitude) + " " + String(pressure_P) + "
" + String(acX) + " " + String(acY) + " " + String(acZ) + " " + String(velGX) + " " +
String(velGY) + " " + String(velGZ) + " " + String(angX) + " " + String(angY) + " " +
String(angZ) + " " + String(digitalRead(skib)));
  myFile.close();
}
```

APÊNDICE D – controleEventos.ino (Controle da ativação de eventos durante o voo)

```
//SIMULACAO
extern float simulaLancamento();

// BIBLIOTECAS
#include <Wire.h>           //biblioteca para controle da comunicacao i2c
#include <MPU6050_tockn.h>  //biblioteca para controle do acelerometro
#include <ICP101xx.h>       //biblioteca para controle do barometro

MPU6050 mpu6050(Wire);  //instanciando o objeto de controle do acelerometro
ICP101xx ICP(Wire);     //instanciando o objeto de controle do barometro

//Declaracao das funcoes
void leituraDados();
float altitudeCalc();
void salvaDados();

//DECLARACAO DE VARIAVEIS
//Variaveis para os dados do acelerometro
float acX = 0.0,
      acY = 0.0,
      acZ = 0.0,
      velGX = 0.0,
      velGY = 0.0,
      velGZ = 0.0,
      angX = 0.0,
      angY = 0.0,
      angZ = 0.0;

//Variaveis para os dados do barometro
float pressure_P = 0.0,
      temperature_C = 0.0,
      altitude = 0.0,
      altRef = 0.0; //valor de referencia para altitude relativa a partir do nivel do solo

bool flagEstadoDescendo = true;

//Variaveis para filtro media movel
const int N = 100;  // Número de valores para calcular a média móvel
float valores[N];   // Array para armazenar os últimos N valores
int indice = 0;     // Índice atual no array
int soma = 0;       // Soma dos últimos N valores

// Funções para controle de eventos
void leituraDados() {
  //dados do acelerometro
  mpu6050.update(); //atualiza o calculo do acelerometro
  //aceleracao na direcao x, y e z em funcao de g
  acX = mpu6050.getAccX();
  acY = mpu6050.getAccY();
```

```
  acZ = mpu6050.getAccZ();
  //velocidade angular em (°/s) de x, y e z
  velGX = mpu6050.getGyroX();
  velGY = mpu6050.getGyroY();
  velGZ = mpu6050.getGyroZ();
  //Estes valores representam a inclinação calculada usando os dados do giroscópio e
acelerometro em relação aos eixos X, Y e Z
  angX = mpu6050.getAngleX();
  angY = mpu6050.getAngleY();
  angZ = mpu6050.getAngleZ();

  //dados do barometro
  if (ICP.getData(pressure_P, temperature_C) == 0) {
    altitude = altitudeCalc(pressure_P, temperature_C);
  }
}

float altitudeCalc(float hPa, float temperature) {
  // Hypsometric Equation (Max Altitude < 11 Km above sea level)
  float local_pressure = hPa * 0.01;
  float sea_level_pressure = 1013.25;                    // hPa
  float pressure_ratio = sea_level_pressure / local_pressure;  // sea level pressure = 1013.25
hPa
  float h = (((pow(pressure_ratio, 1 / 5.257)) - 1) * (temperature)) / 0.0065;
  return h;
}

// Funções para inicialização do sistema
void iniciaSensores() {
  //iniciando comunicacao i2c
  Wire.begin();
  //iniciando acelerometro
  mpu6050.begin();
  Serial.println("MPU6050 inicializado...");
  //mpu6050.calcGyroOffsets(true);
  mpu6050.setGyroOffsets(-4.36, 1.85, -0.38);

  //iniciando o barometro icp10100
  if(!ICP.begin()){
    tone(buzzer, 1000, 200);
    digitalWrite(led, HIGH);
  Serial.println("Erro ao inicializar ICP10100...");
    while(1);
  }

  Serial.println("ICP10100 inicializado...");
  // Start a first measurement cycle, immediately returning control.
  // Optional: Measurement mode
  //    sensor.FAST: ~3ms
  //    sensor.NORMAL: ~7ms (default)
```

```
  //    sensor.ACCURATE: ~24ms
  //    sensor.VERY_ACCURATE: ~95ms
  ICP.start();

  timer2 = millis();  //referencia para contar o tempo de funcionamento do sistema
}

void calculaAltitudeReferencia() {
  // Inicializa os valores do array com 0
  unsigned int timer = millis();
  int i = 0;
  while (i < N) {
    if (millis() - timer >= 10 && ICP.getData(pressure_P, temperature_C) == 0) {
      altitude = altitudeCalc(pressure_P, temperature_C);
      // Subtrai o valor mais antigo da soma
      soma -= valores[indice];
      // Adiciona o novo valor à soma
      soma += altitude;
      // Armazena o novo valor no array
      valores[indice] = altitude;
      // Incrementa o índice, garantindo que ele permaneça dentro dos limites do array
      indice = (indice + 1) % N;
      i++;
      timer = millis();
    }
  }
  altRef = soma / N;
}

float calMediaMovel(float valor) {
  // Subtrai o valor mais antigo da soma
  soma -= valores[indice];
  // Adiciona o novo valor à soma
  soma += valor;
  // Armazena o novo valor no array
  valores[indice] = valor;
  // Incrementa o índice, garantindo que ele permaneça dentro dos limites do array
  indice = (indice + 1) % N;
  // Calcula a média móvel
  return (float)soma / N;
}
```

APÊNDICE E – EstadosVoo.ino (Controle dos estados do sistema)

```
//Declaracao das funcoes
void estadoPreLancamento();
void estadoSubindo();
void estadodescendo();
void estadoRecuperacao();
void estadoPouso();
extern void leituraDados();

extern const int led;
extern const int buzzer;
extern bool displayON;

unsigned int timer1 = 0;    //timer para controle do led e buzzer

bool led_buzzer = false;  //flag para controle do led e buzzer

extern float altitudeRec;  //barometro
extern float altitudeMin;
extern float altitudeMax;
extern float altitudeRef;
extern float altRef;
extern float lastApogeu;
extern float lastRec;

extern bool launch; //verifica a conclusao do lancamento anterior
extern bool flagEstadoDescendo;

// Funções para controle dos estados do sistema
void estadoPreLancamento() {
  leituraDados();
  //Indica com luz e som o aguardo do lancamento
  if ((millis()  - timer1 > 1000) && (displayON == false)) {
    led_buzzer = !led_buzzer;

    if (led_buzzer) {
      tone(buzzer, 1000, 200);
      digitalWrite(led, HIGH);
    } else {
      noTone(buzzer);
      digitalWrite(led, LOW);
    }

    timer1 = millis();
  } else if (displayON) {
    noTone(buzzer);
    digitalWrite(led, LOW);
  }

  //Verifica a altitude
```

```
  if (ICP.getData(pressure_P, temperature_C) == 0) {
    altitude = altitudeCalc(pressure_P, temperature_C);
  }

  calMediaMovel(altitude - altRef);

  if (altitude - altRef >= altitudeMin) {
    estado = 1;
    launch = true;
    noTone(buzzer);
    digitalWrite(led, LOW);
    displayON = false;
    criaArqDados();
    salvarArquivo("/launch.txt");
    Serial.println("SUBINDO...");
  }
}

void estadoSubindo() {
  leituraDados();
  salvaDados();

  if (altitude - altRef > altitudeMax) altitudeMax = altitude - altRef;
  if (abs(altitude - altRef) - abs(calMediaMovel(altitude - altRef)) < -15) {
    estado = 2;
    lastApogeu = altitudeMax;
    salvarArquivo("/launch.txt");
    Serial.println("DESCENDO1...");
  }
}

void estadodescendo() {
  leituraDados();
  salvaDados();

  switch (metodo) {
    case 1:
      if (flagEstadoDescendo && abs(altitude - altRef) <= altitudeRec) {
        estado = 3;
        lastRec = abs(altitude - altRef);
      }
      break;

    case 2:
      if (flagEstadoDescendo && abs(altitude - altRef) <= altitudeMax * altitudeRef) {
        estado = 3;
        lastRec = abs(altitude - altRef);
      }
      break;
```

```
    case 3:
      if (flagEstadoDescendo && abs(altitude - altRef) <= altitudeMax * 0.5) {
        estado = 3;
        lastRec = abs(altitude - altRef);
      }
      break;
  }

  if (!flagEstadoDescendo && abs(altitude - altRef) <= 20) {  // encerra o voo, estado pouso
    estado = 4;
   // salvarArquivo("/launch.txt");
  }
}

void estadoRecuperacao() {
  digitalWrite(skib, HIGH);
  Serial.println("RECUPERACAO ATIVADA!");
  estado = 2;
  flagEstadoDescendo = false;
  File file = SD.open("/laststatistics.txt", FILE_WRITE);
  if (file) {
    file.println(lastApogeu);
    file.println(lastRec);
    file.close();
  }
  salvarArquivo("/launch.txt");
  Serial.println("DESCENDO2...");
}

void estadoPouso() {
  digitalWrite(skib, LOW);
  myFile.close();
  estado = 100;
  launch = false;
  salvarArquivo("/launch.txt");
  Serial.println("POUSO...");
  while (1)
    ;
}
```

APÊNDICE F – IHM.ino (Controle do display, para simulação)

```
#include <Adafruit_GFX.h>      //biblioteca para controle do display
#include <Adafruit_SSD1306.h>  //bibioteca para controle do display

//SIMULACAO (coeficientes y = ax^2 + bx)
extern float a;
extern float b;
extern bool flagSimulacao;  //habilita o inicio da simulacao

extern byte estado; //estados do sistema

//Importando variaveis dos sensores
//Acelerometro
extern float acX;
extern float acY;
extern float acZ;
extern float velGX;
extern float velGY;
extern float velGZ;
extern float angX;
extern float angY;
extern float angZ;
//Barometro
extern float altitudeRec;
extern float altitudeMin;
extern float altitudeRef;
extern float lastApogeu;
extern float lastRec;
extern float altitude;
extern float temperature_C;
extern float pressure_P;
//Sinalizacoes
extern const int led;
extern const int buzzer;

//Prototipo das funcoes
void displayIHM();
void iniciadisplay();
extern void leituraDados();

// Define o pino do botão subir, confirmar e descer
const int encAPin = 12,  // pino do Encoder A
  encBPin = 13,          // pino do Encoder B
  bttPin = 14;           // pino do botao

// Matrizes construturas do menu
const char menutitulos[][20] = { "Menu princiapl", "Acionamento", "Sensores",
"LastStatistics" };
const char menuOptions[][4][20]  = { { "1 - LaunchMode", "2 - Acionamento", "3 - Sensores",
"4 - LastStatistics" },  // opc == 0
```

```
                          { "1 - Voltar", "2 - Altitude", "3 - %", "4 - Automatico" },   // opc == 1
                          { "1 - Voltar", "2 - Acelerometro", "3 - Barometro", "" },     // opc == 2
                          { "1 - Voltar", "Alt[m]:", "", "" },                           // opc == 3
                          { "1 - Voltar", "Alt[%]:", "", "" },                           // opc == 4
                          { "1 - Voltar", "A:", "G:", "*:" },                            // opc == 5
                          { "1 - Voltar", "Temp[C]:  ", "Pres[Pa]:  ", "Alt[m]:  " },    // opc == 6
                          { "1 - Voltar", NULL, NULL, NULL } };                          // opc == 7

//Variaveis para controle da IHM
int numOptions = 4;          //controla o espaço de movimento do cursor
int menu = 0;                //indica o menu exibido na parte superior
int opc = 0;                 //enumera as telas
int selectedIndex = 0;       //controla o item selecionado na tela
bool flag = false;           //verifica se houve algum comando
bool displayON = false;      //liga e desliga a tela
int tempoSTB = 30 * 1E3;     //tempo para a IHM entrar em stand-by [s]
bool editandoValores = false;  //controla o momento de controlar o menu e alterar valores

//Variaveis para controle do metodo de acionamento
int metodo = 3;  // 1 -  altitude, 2 - por %, 3 - automatico

//Parametros do display
#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64
#define SCREEN_ADDRESS 0x3C

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, -
1);  //instaciando objeto para controle do display

void iniciadisplay() {
  if (!display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS)) {  //verifica se o
display está conectado
    while (1)
      ;
  }
  Serial.println("Display  iniciado...");

  display.clearDisplay();               //limpa  a tela
  display.setTextColor(SSD1306_WHITE);  //define a cor da escrita
  display.setTextSize(1);               //define o tamanho da fonte
  display.display();                    //atualiza a tela

  pinMode(encAPin, INPUT_PULLUP);
  pinMode(encBPin, INPUT_PULLUP);
  pinMode(bttPin, INPUT_PULLUP);
  //Serial.println("Display  iniciado");
}

void displayIHM() {
  //desliga a tela caso saia do estado inicial do sistema
```

```
if (estado != 0) {
  display.clearDisplay();
  display.display();
  displayON = false;
  return;
}
//Atualiza as leituras dos botoes
bool encA = !digitalRead(encAPin);          //le o valor do encoder A
bool encB = !digitalRead(encBPin);          //le o valor do encoder B
bool btt = !digitalRead(bttPin);            //le o valor do botao
static volatile bool LastA, LastB, LastBtt;  //guarda os estados anteriores dos encoders
static volatile unsigned long int timer4;    //timer para atualizar a tela
static volatile unsigned long int timer3;    //timer para stand-by display

//entra em LaunchMode
if ((btt == LOW && LastBtt == HIGH) && (menu == 0 && selectedIndex == 0 && opc ==
0)) {
  flag = false;
  displayON = false;
  display.clearDisplay();
  display.display();

  Serial.println("\ny = (" + String(a) + ")*x^2 + (" + String(b) + ")x");
  flagSimulacao = true;
  delay(5000);
}
//Controle da tela de acionamento
//opcao voltar [Menu principal <- Acionamento]
if ((btt == LOW && LastBtt == HIGH) && (menu == 1 && selectedIndex == 0) && opc
== 1) {
  menu = 0;
  selectedIndex = 0;
  opc = 0;
  flag = true;
}
//Metodo de acionamento
//opcao voltar [Acionamento <- AltAcionamento]
if ((btt == LOW && LastBtt == HIGH) && (menu == 1 && selectedIndex == 0) && opc
== 3) {
  selectedIndex = 0;
  opc = 1;
  flag = true;
  numOptions = 4;
}
//define metodo de acionamento como alt
if ((btt == LOW && LastBtt == HIGH) && (menu == 1 && selectedIndex == 1 && opc ==
1)) {
  menu = 1;
  selectedIndex = 0;
  opc = 3;
```

```
    flag = true;
    numOptions = 2;
    //metodo = 1;
  }

  //habilita a edicao da altitude de acionamento
  if ((btt == LOW && LastBtt == HIGH) && (menu == 1 && selectedIndex == 1 && opc ==
3)) {
    editandoValores = !editandoValores;
    if (editandoValores == false) {
      metodo = 1;
      salvarArquivo("/parametros.txt");
    }
  }

  //opcao voltar [Acionamento <- %]
  if ((btt == LOW && LastBtt == HIGH) && (menu == 1 && selectedIndex == 0) && opc
== 4) {
    selectedIndex = 0;
    opc = 1;
    flag = true;
    numOptions = 4;
  }
  //define metodo de acionamento como %
  if ((btt == LOW && LastBtt == HIGH) && (menu == 1 && selectedIndex == 2) && opc
== 1) {
    menu = 1;
    selectedIndex = 0;
    opc = 4;
    flag = true;
    numOptions = 2;
    //metodo = 2;
  }

  //habilita a edicao da % de acionamento
  if ((btt == LOW && LastBtt == HIGH) && (menu == 1 && selectedIndex == 1 && opc ==
4)) {
    editandoValores = !editandoValores;
    if (editandoValores == false) {
      metodo = 2;
      salvarArquivo("/parametros.txt");
    }
  }

  //definir metodo de acionamento como automatico
  if ((btt == LOW && LastBtt == HIGH) && (menu == 1 && selectedIndex == 3) && opc
== 1) {
    flag = true;
    metodo = 3;
    altitudeRef = 0.5;
```

```
    salvarArquivo("/parametros.txt");
  }
  //Controle da tela de sensores
  //opcao volta [Menu principal <- sensores]
  if ((btt == LOW && LastBtt == HIGH) && (menu == 2 && selectedIndex == 0) && opc
== 2) {
    menu = 0;
    selectedIndex = 0;
    opc = 0;
    flag = true;
    numOptions = 4;
  }

  //opcao voltar [sensores <- acelerometro]
  if ((btt == LOW && LastBtt == HIGH) && (menu == 2 && selectedIndex == 0) && opc
== 5) {
    selectedIndex = 0;
    opc = 2;
    flag = true;
    numOptions = 3;
  }

  //opcao acelerometro [sensores -> acelerometro]
  if ((btt == LOW && LastBtt == HIGH) && (menu == 2 && selectedIndex == 1) && opc
== 2) {
    selectedIndex = 0;
    opc = 5;
    flag = true;
    numOptions = 1;
  }

  //opcao voltar [sensores <- barometro]
  if ((btt == LOW && LastBtt == HIGH) && (menu == 2 && selectedIndex == 0) && opc
== 6) {
    selectedIndex = 0;
    opc = 2;
    flag = true;
    numOptions = 3;
  }

  //opcao acelerometro [sensores -> barometro]
  if ((btt == LOW && LastBtt == HIGH) && (menu == 2 && selectedIndex == 2) && opc
== 2) {
    selectedIndex = 0;
    opc = 6;
    flag = true;
    numOptions = 1;
  }

  //entra na tela acionamento
```

```
  if ((btt == LOW && LastBtt == HIGH) && (menu == 0 && selectedIndex == 1)) {
    menu = 1;
    selectedIndex = 0;
    opc = 1;
    flag = true;
  }

  //entra na tela sensores
  if ((btt == LOW && LastBtt == HIGH) && (menu == 0 && selectedIndex == 2)) {
    menu = 2;
    selectedIndex = 0;
    opc = 2;
    flag = true;
    numOptions = 3;
  }

  //volta  tela inicial <- LastStatistics
  if ((btt == LOW && LastBtt == HIGH) && (menu == 3 && selectedIndex == 0 && opc ==
7)) {
    menu = 0;
    selectedIndex = 0;
    opc = 0;
    flag = true;
    numOptions = 4;
  }

  //entra na tela LastStatistics
  if ((btt == LOW && LastBtt == HIGH) && (menu == 0 && selectedIndex == 3 && opc ==
0)) {
    menu = 3;
    selectedIndex = 0;
    opc = 7;
    flag = true;
    numOptions = 1;

    resgataLastStatistics(); //atualiza os valores do ultimo lancamento com base nos dados em
memoria
  }

  //controle da interacao da chave seletora com a IHM
  if ((millis()  - timer4 >= 30) && (LastA != encA || LastB != encB)) {  //verifica o seletor
    if (editandoValores == false) {
      if (encA == false && encB == true) {  // incrementa o seletor
        selectedIndex = (selectedIndex + 1) % numOptions;
        flag = true;
      } else if (encA == true && encB == false) {  // decrementa o seletor
        selectedIndex--;
        if (selectedIndex < 0) selectedIndex = numOptions - 1;
        flag = true;
      }
```

```
    } else {
      if (opc == 3) {                                    //mudando a altitude de acionamento da rec
        if (encA == false && encB == true) {                    // incrementa o seletor
          altitudeRec += altitudeRec < 3000 ? altitudeMin : 0.0;  //altitude maxima de
acionamento de 3km
          flag = true;
        } else if (encA == true && encB == false) {  // decrementa o seletor
          altitudeRec -= altitudeRec > altitudeMin ? altitudeMin : 0.0;
          if (altitudeRec < 50.0) altitudeRec = 50.0;
          flag = true;
        }
      }

      if (opc == 4) {                                 //mudando o % de acionamento da rec
        if (encA == false && encB == true) {               // incrementa o seletor
          altitudeRef += altitudeRef < 0.95 ? 0.05 : 0.0;  //altitude maxima de acionamento de
3km
          flag = true;
        } else if (encA == true && encB == false) {  // decrementa o seletor
          altitudeRef -= altitudeRef > 0.5 ? 0.05 : 0.0;
          if (altitudeRef < 0.5) altitudeRef = 0.5;
          flag = true;
        }
      }
    }
  }

  //ativa o stand by
  if (millis() - timer3 >= tempoSTB && displayON && opc != 5) {
    display.clearDisplay();
    display.display();
    //Serial.println("STB  ON");
    displayON = false;
  }

  //constroi a tela
  if (flag && (millis() - timer4 >= 80)) {                          //constroi apenas se alguma
mudanca for feita e apos um certo tempo entre a ultima atualizacao
    display.clearDisplay();                                      //limpa  a tela
    display.setTextColor(SSD1306_WHITE);                              //define texto na cor
normal
    display.setCursor(0, 0);                                     //posiciona o cursor no inicio do
tela
    display.println(menutitulos[menu]);                               //escreve o nome do menu no
cabecalho da tela
    display.println("-------------------");                           //divide o cabecalho das opcoes
    for (int i = 0; i < (numOptions == 1 ? numOptions + 3 : numOptions); i++) {  // controla a
animacao do seletor
      if (i == selectedIndex) {                                  // faz o item selecionado inverter
as cores
```

```
      display.setTextColor(SSD1306_BLACK, SSD1306_WHITE);
      display.print("> ");
    } else {
      display.setTextColor(SSD1306_WHITE);
      display.print("  ");
    }

    if (opc == 1 && (metodo == i)) {  //escreve as opcoes destacando o metodo de
acionamento da recuperacao
      display.print(menuOptions[opc][i]);
      display.print(" (*)");
    } else {
      display.print(menuOptions[opc][i]);
    }

    if (opc == 5) {  //exibe os dados do acelerometro
      if (i == 1) display.print(String(acX) + ";" + String(acY) + ";" + String(acZ));
      if (i == 2) display.print(String(velGX) + ";" + String(velGY) + ";" + String(velGZ));
      if (i == 3) display.print(String(int(angX)) + ";" + String(int(angY)) + ";" +
String(int(angZ)));
    }

    if (opc == 6) {  //exibe os dados do barometro
      //if (i == 1) display.print(pressure_P);
      //if (i == 2) display.print(temperature_C);
      //if (i == 3) display.print(altitude);
      if (i == 1) display.print(pressure_P);
      if (i == 2) display.print(15);
      if (i == 3) display.print(altitude);
    }

    if (opc == 3) {  //exibe a altitudeRec na respectiva tela de acionamento
      if (i == 1) display.print(altitudeRec);
    }

    if (opc == 4) {  //exibe a % na respectiva tela de acionamento
      if (i == 1) display.print(altitudeRef);
    }

    if (opc == 7) { //exibe os dados do ultimo lancamento
      if (i == 1 && (lastApogeu == 0.0 || lastRec == 0.0)) {
        display.print("Sem lancamentos \n anteriores...");
      } else {
        if (i == 1) display.print("Apogeu: " + String(lastApogeu));
        if (i == 2) display.print("Rec: " + String(lastRec));
      }
    }

    display.println();
  }
```

```
    display.display();  //atualiza a tela

    if (opc != 5 && opc != 6) {
      flag = false;  //comeca a contar o timer para desligar a tela
    } else {
      flag = true;  //para poder atualizar os valores dos sensores no display
    }
    timer3 = millis();  //atualiza o timer do stand-by
    timer4 = millis();  //atualiza o timer do seletor
    displayON = true;
  }
  //guarda os valores anteriores do seletor
  LastA = encA;
  LastB = encB;
  LastBtt = btt;
}
```

APÊNDICE G – controleEventos.ino (Controle da ativação de eventos durante o voo, para simulação)

```
//SIMULACAO
extern float simulaLancamento();

// BIBLIOTECAS
#include <Wire.h>          //biblioteca para controle da comunicacao i2c
#include <MPU6050_tockn.h>  //biblioteca para controle do acelerometro
#include <ICP101xx.h>      //biblioteca para controle do barometro

MPU6050 mpu6050(Wire);  //instanciando o objeto de controle do acelerometro
ICP101xx ICP(Wire);     //instanciando o objeto de controle do barometro

//Declaracao das funcoes
void leituraDados();
float altitudeCalc();
void salvaDados();

//DECLARACAO DE VARIAVEIS
//Variaveis para os dados do acelerometro
float acX = 0.0,
      acY = 0.0,
      acZ = 0.0,
      velGX = 0.0,
      velGY = 0.0,
      velGZ = 0.0,
      angX = 0.0,
      angY = 0.0,
      angZ = 0.0;

//Variaveis para os dados do barometro
float pressure_P = 0.0,
      temperature_C = 0.0,
      altitude = 0.0,
      altRef = 0.0; //valor de referencia para altitude relativa a partir do nivel do solo

bool flagEstadoDescendo = true;

//Variaveis para filtro media movel
const int N = 100; // Número de valores para calcular a média móvel
float valores[N];  // Array para armazenar os últimos N valores
int indice = 0;    // Índice atual no array
int soma = 0;      // Soma dos últimos N valores

// Funções para controle de eventos
void leituraDados() {
  //dados do acelerometro
  mpu6050.update(); //atualiza o calculo do acelerometro
  //aceleracao na direcao x, y e z em funcao de g
```

```
  acX = mpu6050.getAccX();
  acY = mpu6050.getAccY();
  acZ = mpu6050.getAccZ();
  //velocidade angular em (°/s) de x, y e z
  velGX = mpu6050.getGyroX();
  velGY = mpu6050.getGyroY();
  velGZ = mpu6050.getGyroZ();
   //Estes valores representam a inclinação calculada usando os dados do giroscópio e
acelerometro em relação aos eixos X, Y e Z
  angX = mpu6050.getGyroAngleX();
  angY = mpu6050.getGyroAngleY();
  angZ = mpu6050.getGyroAngleZ();

  //dados do barometro
  /*
  if (ICP.getData(pressure_P, temperature_C) == 0) {
    altitude = altitudeCalc(pressure_P, temperature_C);
  }
  */
  altitude = simulaLancamento();
}

float altitudeCalc(float hPa, float temperature) {
  // Hypsometric Equation (Max Altitude < 11 Km above sea level)
  float local_pressure = hPa * 0.01;
  float sea_level_pressure = 1013.25;                    // hPa
  float pressure_ratio = sea_level_pressure / local_pressure;  // sea level pressure = 1013.25 hPa
  float h = (((pow(pressure_ratio, 1 / 5.257)) - 1) * (temperature)) / 0.0065;
  return h;
}

// Funções para inicialização do sistema
void iniciaSensores() {
  //iniciando comunicacao i2c
  Wire.begin();
  //testando conexao com os sensores
  const byte mpu6050Address = 0x68; // or 0x69
  const byte oledAddress = 0x3C;    // or 0x3D
  const byte icp10100Address = 0x63;

  checkI2CDevice(mpu6050Address, "MPU6050");
  checkI2CDevice(oledAddress, "OLED Display");
  //checkI2CDevice(icp10100Address, "ICP-10100");

  //iniciando acelerometro
  mpu6050.begin();

  Serial.println("MPU6050 inicializado...");
  //mpu6050.calcGyroOffsets(true);
  mpu6050.setGyroOffsets(-4.36, 1.85, -0.38);
```

```
  //iniciando o barometro icp10100
  //ICP.begin();
  //if(!ICP.begin()){
  //  tone(buzzer, 1000, 200);
  //  digitalWrite(led, HIGH);
  //Serial.println("Erro ao inicializar ICP10100...");
  //  while(1);
  //}
  Serial.println("ICP10100 inicializado...");
  // Start a first measurement cycle, immediately returning control.
  // Optional: Measurement mode
  //    sensor.FAST: ~3ms
  //    sensor.NORMAL: ~7ms (default)
  //    sensor.ACCURATE: ~24ms
  //    sensor.VERY_ACCURATE: ~95ms
  //ICP.start();

  timer2 = millis();  //referencia para contar o tempo de funcionamento do sistema
}

void checkI2CDevice(byte address, const char* deviceName) {
  Serial.print("Verificando ");
  Serial.print(deviceName);
  Serial.print(" no endereco: 0x");
  if (address < 16) {
    Serial.print("0");
  }
  Serial.println(address, HEX);

  Wire.beginTransmission(address);
  byte error = Wire.endTransmission();

  if (error == 0) {
    Serial.print(deviceName);
    Serial.println(" conectado.");
  } else {
    Serial.print("Erro: ");
    Serial.print(deviceName);
    Serial.println(" não encontrado.");
    while (1) {
      tone(buzzer, 1000, 200);
      digitalWrite(led, HIGH);
    }
  }
}

void calculaAltitudeReferencia() {
  // Inicializa os valores do array com 0
  unsigned int timer = millis();
```

```
  int i = 0;
  while (i < N) {
    if (millis()  - timer >= 10 && ICP.getData(pressure_P, temperature_C) == 0) {
      //altitude = altitudeCalc(pressure_P, temperature_C);
      altitude = simulaLancamento();
      // Subtrai o valor mais antigo da soma
      soma -= valores[indice];
      // Adiciona o novo valor à soma
      soma += altitude;
      // Armazena o novo valor no array
      valores[indice] = altitude;
      // Incrementa o índice, garantindo que ele permaneça dentro dos limites do array
      indice = (indice + 1) % N;
      i++;
      timer = millis();
    }
  }
  altRef = soma / N;
}
float calMediaMovel(float valor) {
  // Subtrai o valor mais antigo da soma
  soma -= valores[indice];
  // Adiciona o novo valor à soma
  soma += valor;
  // Armazena o novo valor no array
  valores[indice] = valor;
  // Incrementa o índice, garantindo que ele permaneça dentro dos limites do array
  indice = (indice + 1) % N;
  // Calcula a média móvel
  return (float)soma / N;
}
```