INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
CAMPUS DE CAJAZEIRAS
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

JOSÉ KLINSMAN JORGE SILVA

Air CoPilot: UM COPILOTO ARTIFICIAL PARA AUXILIAR OS PILOTOS DE DRONES E PREVENIR ACIDENTES

CAJAZEIRAS
2025

José Klinsman Jorge Silva

Air CoPilot: UM COPILOTO ARTIFICIAL PARA AUXILIAR OS PILOTOS DE DRONES E PREVENIR ACIDENTES

Trabalho de Conclusão de Curso submetido ao Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, Campus Cajazeiras, como requisito parcial para a obtenção do grau de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador: Prof. Dr. Fabio Gomes de Andrade

Cajazeiras
2025

IFPB / Campus Cajazeiras
Coordenação de Biblioteca
Biblioteca Prof. Ribamar da Silva
Catalogação na fonte: Cícero Luciano Félix CRB-15/750

S586a Silva, José Klinsman Jorge.

Air CoPilot : um copiloto artificial para auxiliar os pilotos de drones e prevenir acidentes / José Klinsman Jorge Silva. – Cajazeiras, 2025.
66f. : il.

Trabalho de Conclusão de Curso (Tecnólogo em Análise e Desenvolvimento de Sistemas) – Instituto Federal de Educação, Ciência e Tecnologia da Paraíba, Cajazeiras, 2025.

Orientador: Prof. Dr. Fabio Gomes de Andrade.

1. Desenvolvimento de sistemas. 2. Copiloto artificial. 3. Drone. 4. Veículo aéreo não tripulado. I. Instituto Federal da Paraíba. II. Título.

IFPB/CZ CDU: 004.4(043.2)

MINISTÉRIO DA EDUCAÇÃO
SECRETARIA DE EDUCAÇÃO PROFISSIONAL E TECNOLÓGICA
INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA

JOSÉ KLINSMAN JORGE SILVA

Air CoPilot: UM COPILOTO ARTIFICIAL PARA AUXILIAR OS PILOTOS DE DRONES E PREVENIR ACIDENTES

Trabalho de Conclusão de Curso apresentado junto ao Curso Superior de Tecnologia em Análise e Desenvolvimento de Sistemas do Instituto Federal de Educação, Ciência e Tecnologia da Paraíba - Campus Cajazeiras, como requisito à obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas.

Orientador

Prof. Dr. Fabio Gomes de Andrade

Aprovada em: **21 de Março de 2025.**

Prof. Dr. Fabio Gomes de Andrade - Orientador

Prof. Esp. João Igor Barros Rocha - Avaliador

IFPB - Campus Cajazeiras

Prof. Me. Fábio Abrantes Diniz - Avaliador

IFPB - Cajazeiras

Documento assinado eletronicamente por:

- **Fabio Gomes de Andrade**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 24/03/2025 16:20:29.
- **Joao Igor Barros Rocha**, **PROF ENS BAS TEC TECNOLOGICO-SUBSTITUTO**, em 24/03/2025 17:40:37.
- **Fabio Abrantes Diniz**, **PROFESSOR ENS BASICO TECN TECNOLOGICO**, em 24/03/2025 19:53:47.

Este documento foi emitido pelo SUAP em 24/03/2025. Para comprovar sua autenticidade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/autenticar-documento/ e forneça os dados abaixo:

Código 687207
Verificador: e0b5665a2d
Código de Autenticação:

AGRADECIMENTOS

Agradeço, primeiramente, a Deus pelo dom da vida e por me conceder a oportunidade do conhecimento.

Agradeço a toda a minha família, em especial aos meus pais, que sempre me apoiaram desde o curso técnico.

Agradeço à minha esposa por todo o apoio e incentivo, estando ao meu lado durante toda a minha jornada acadêmica.

Agradeço à minha filha, que foi e continua sendo a maior motivação para meu foco, minha vontade de vencer e de realizar novas conquistas.

Agradeço ao meu professor orientador, Prof. Dr. Fabio Gomes de Andrade, por toda a paciência e pelo tempo dedicado.

Agradeço ao meu professor coorientador, Prof. Me. George Candeia de Sousa Medeiros, por todo o apoio.

Agradeço ao meu amigo Alef Almeida, que sempre tornou meu horário de trabalho flexível para que eu pudesse conciliar minha rotina de trabalho e estudo, além de todos os conselhos.

Por fim, agradeço ao time acadêmico do IFPB pelo excelente trabalho de ensino.

RESUMO

O uso de drones no Brasil tem experimentado um crescimento exponencial nos últimos anos. Segundo a ANAC (Agência Nacional de Aviação Civil), em 2020 havia mais de 78.000 drones no país, sendo cerca de 60% utilizados para fins recreativos e o restante para atividades profissionais como agricultura e filmagens aéreas. A maioria dos drones utilizados atualmente não possui mecanismos adequados de segurança, tornando o uso de VANTs (Veículos Aéreos Não Tripulados) uma prática potencialmente arriscada, tanto para os pilotos quanto para os civis. De acordo com o blog Drone Shield, uma pesquisa sobre acidentes com aeronaves rádio controladas aponta que 32% dos incidentes estão relacionados a filmagens aéreas, uma das áreas mais comuns de atuação desses equipamentos. Os principais fatores que contribuem para quedas ou falhas de voo incluem problemas com a bateria, sensores IMU e sistemas de controle, além de colisões com obstáculos como árvores, redes elétricas e estruturas. Diante desses riscos, este trabalho propõe o desenvolvimento e a implementação do Air CoPilot , um sistema de copiloto artificial baseado em redes neurais. Com objetivo identificar falhas e potenciais riscos durante o voo, alertar o piloto e otimizar a segurança e eficiência da operação de drones, especialmente em ambientes profissionais e com equipamentos de maior porte.

Palavras-chave: Drones. VANTs. ANAC.

ABSTRACT

The use of drones in Brazil has experienced exponential growth in recent years. According to ANAC (National Civil Aviation Agency), in 2020 there were more than 78,000 drones in the country, with around 60% used for recreational purposes and the rest for professional activities such as agriculture and aerial operations. Most drones currently used do not have adequate safety mechanisms, making the use of UAVs (Unmanned Aerial Vehicles) a potentially risky practice, both for pilots and civilians. According to the Drone Shield blog, a survey of accidents involving radio-controlled aircraft indicates that 32% of incidents are related to aerial filming, one of the most common areas of operation of these equipment. The main factors that hinder crashes or flight failures include problems with the battery, IMU sensors and control systems, as well as collisions with obstacles such as trees, power lines and structures. Given these risks, this work proposes the development and implementation of Air CoPilot, an artificial copilot system based on neural networks. It aims to identify failures and potential risks during flight, alert the pilot and improve the safety and efficiency of drone operations, especially in professional environments and with larger equipment.

Keywords: Drones. VANTs. ANAC.

LISTA DE FIGURAS

Figura 1 - Ilustração dos sensores de obstáculos do Mavic 3……10
Figura 2 - Mini computador Orange Pi Zero 2W……..……...19
Figura 3 - Placa HTIT-WSL……...……20
Figura 4 - Funcionamento de um sensor LiDAR 2D……..…24
Figura 5 - Arquitetura do sistema Air CoPilot……..……29
Figura 6 - Módulo de processamento fechado……..……..……32
Figura 7 - Módulo de processamento aberto……..…..……32
Figura 8 - Drone usado nos testes do sistema……...……..……33
Figura 9 - Local onde foi feita a instalação do LiDAR……...……34
Figura 10 - Sensor LiDAR devidamente instalado…….……34
Figura 11 - Local de fixação do módulo de processamento……...…35
Figura 12 - Montagem do módulo receptor de telemetria……..…37
Figura 13 - Trecho de código que coleta a telemetria……..…39
Figura 14 - Função que envia os dados para o Orange Pi Zero 2W……40
Figura 15 - Trecho de código que envia dados para o módulo de visualização…….41
Figura 16 - Implementação das tarefas do sinal SBUS……..42
Figura 17 - Código Python responsável por receber os pacotes de telemetria…..…43
Figura 18 - Construção da rede neural……..……...……45
Figura 19 - Compilando a rede neural……...……45
Figura 20 - Parte do Conjunto de dados primário……...……47
Figura 21 - Trecho de código responsável por treinar a rede neural……48
Figura 22 - Aplicando o algoritmo DBSCAN…….……50
Figura 23 - Código JavaScript para criar conexão WebSocket…….……51
Figura 24 - Código de rotas do servidor……...……53
Figura 25 - Corpo da função que recebe os dados e prepara o JSON……54
Figura 26 - Interface gráfica do sistema com o drone desligado……..……56
Figura 27 - Interface gráfica do sistema com o drone ligado……58
Figura 28 - Interface gráfica do sistema com o drone em voo……59

LISTA DE QUADROS

Quadro 1 - Cronograma de atividades do TCC…..……...……..…..... 17
Quadro 2 - Custos do projeto…….. ……. 37

LISTA DE ABREVIATURAS E SIGLAS

AI	Artificial Intelligence
ANAC	Agência Nacional de Aviação Civil
ANATEL	Agência Nacional de Telecomunicações
API	Application Programming Interface
CPU	Central Processing Unit
CSV	Comma-Separated Values
DBSCAN	Density-Based Spatial Clustering of Applications with Noise
DECEA	Departamento de Controle do Espaço Aéreo
DJI	Da-Jiang Innovations
ESP	Espressif Systems
GPIO	General Purpose Input/Output
GPS	Global Positioning System
GUI	Graphical User Interface
HTML	HyperText Markup Language
HTTP	Hypertext Transfer Protocol
I2C	Inter-Integrated Circuit
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
JSON	JavaScript Object Notation
LiDAR	Light Detection and Ranging
LoRa	Long Range
ML	Machine Learning
RC	Remote Control
RSSI	Received Signal Strength Indicator
SISANT	Sistema de Aeronaves Não Tripuladas
TCC	Trabalho de Conclusão de Curso
UART	Universal Asynchronous Receiver/Transmitter
UAV	Unmanned Aerial Vehicle
USB	Universal Serial Bus
VANT	Veículo Aéreo Não Tripulado

SUMÁRIO

1. INTRODUÇÃO **9**
1.1 Motivação 12
1.2 Objetivos 13
1.2.1 Objetivo Geral 13
1.2.2 Objetivos Específicos 13
1.3 Trabalhos Relacionados 14
1.4 Metodologia 15
1.5 Cronograma de atividades 16
1.6 Organização do Documento 17
2. FUNDAMENTAÇÃO TEÓRICA **18**
2.1 O hardware 18
2.1.1 Placa Orange Pi Zero 2W 18
2.1.2 Heltec Wireless Stick Lite (HTIT-WSL) 19
2.1.3 LiDAR LD14P (Light Detection and Ranging) 20
2.2 O software 23
2.2.1 Linguagem de programação Python 23
2.2.2 Rede Neural de Classificação 24
2.2.3 Algoritmo de clusterização DBSCAN 25
2.2.4 Framework Arduino Espressif 26
3. A SOLUÇÃO PROPOSTA **27**
3.1 Stakeholders 27
3.2 Requisitos Funcionais 27
3.3 Arquitetura do Sistema 28
4. IMPLEMENTAÇÃO **31**
4.1 Montagem do Hardware 31
4.1.1 Módulo de processamento e drone de testes 31
4.1.2 Sensor de obstáculos LiDAR 33
4.1.3 Módulo receptor de telemetria e visualização 36
4.2 Desenvolvimento do software e testes do sistema 38
4.2.1 Firmware do HTIT-WSL que compõe o módulo de processamento 38
4.2.2 Desenvolvimento do módulo de processamento 43
4.2.3 Desenvolvimento do módulo de visualização 51
5. CONCLUSÃO **60**
REFERÊNCIAS **61**
GLOSSÁRIO **64**

1. INTRODUÇÃO

Uma máquina voadora capaz de filmar, explorar, transportar e não só isso, mas também executar tarefas em que a presença do operador não seja algo relevante. Essas características definem os *Unmanned Aerial Vehicle* (UAV)[1] como dispositivos não tripulados que podem ser controlados por sistemas de rádio controle. Eles surgiram da demanda dos militares por equipamentos que permitissem o reconhecimento de áreas e até mesmo o ataque ao inimigo ao mesmo tempo em que oferecessem segurança para os soldados (MAHADEVAN, 2010). No entanto, os dispositivos UAV ficaram popularmente conhecidos como drones ou Veículo Aéreo Não Tripulado (VANT) como são conhecidos no Brasil. Por se tratarem de dispositivos versáteis, com inúmeras possibilidades de aplicações, podendo ser operados em curtas e longas distâncias com altitude de voo variável, os drones podem ser usados em algumas aplicações como, por exemplo:

- **uso recreativo**: essa categoria de uso abrange os usuários sem experiência. Os drones que compõem essa categoria possuem aparência de brinquedo ou replicam algum recurso profissional;
- **uso profissional:** as aeronaves são mais robustas e apresentam características profissionais como câmeras avançadas, sistemas de geoposicionamento, sensores para evitar colisão e rádio controle mais eficiente;
- **uso industrial**: os VANTs dessa categoria são montados exclusivamente para executarem funções específicas, como a aplicação de fertilizantes agrícolas, a produção de imagens cinematográficas, a inspeção de redes elétricas entre outras aplicações. Os drones apresentam peso de decolagem geralmente superior aos drones das demais categorias e possuem sistemas mais avançados de controle e geoposicionamento.

Em meio a tantas aplicações existentes envolvendo o uso de drones, leis tiveram que ser implementadas com o objetivo de regulamentar o uso desses equipamentos. No Brasil, o órgão responsável por registrar e regulamentar o uso de

[1] https://ajuda.decea.mil.br/base-de-conhecimento/qual-a-diferenca-entre-drone-vant-e-rpas/

VANTs é a Agência Nacional de Aviação Civil (ANAC) juntamente com o Departamento de Controle do Espaço Aéreo (DECEA) e a Agência Nacional de Telecomunicações (ANATEL). Em geral, as normas da ANAC são bem simples e se resumem em respeitar a distância-limite de terceiros, que é regulamentada em pelo menos trinta metros horizontais, e seguir as normas do DECEA, que orienta os pilotos a não sobrevoarem áreas de segurança, fazerem uso recreativo apenas em locais destinados para tal aplicação, manterem a manutenção da aeronave atualizada, entre outras orientações. A ANAC[2] classifica os drones em três classes distintas, sendo elas:

- **classe 1:** corresponde aos VANTs com peso de decolagem acima de 150 Kg. A regulamentação brasileira prevê que esses equipamentos sejam submetidos a processos de certificação, como os processos aplicados às aeronaves tripuladas;
- **classe 2**: corresponde às aeronaves com massa de decolagem acima de 25 Kg e abaixo ou igual a 150 Kg. O regulamento estabelece requisitos técnicos exigidos do fabricante, além de registro no Registro Aeronáutico Brasileiro;
- **classe 3:** corresponde aos drones que apresentam peso máximo de decolagem de 25 Kg. Para as aeronaves que operam acima de 120 metros de altitude, o VANT deve ser de projeto autorizado pela ANAC e precisa ser registrado com matrícula e marcas de nacionalidade. Para os drones que sobrevoam abaixo dos 120 metros de altitude deve ser realizado apenas o registro no Sistema de Aeronaves Não Tripuladas (SISANT). Já as aeronaves com massa de até 250g não necessitam de registros, independentemente da sua finalidade.

Segundo dados fornecidos pela ANAC[3], nos quatro primeiros meses de 2022 foram registrados 3.699 novos drones no sistema SISANT, o que representa um aumento aproximado de 4,1% em relação a dezembro de 2021. Em abril de 2022 o SISANT contava com 93.729 VANTs cadastrados, do qual aproximadamente 43,56% são aeronaves usadas para fins profissionais. Com um número de equipamentos profissionais consideravelmente grande, se faz necessário o uso de

[2] https://www.gov.br/anac/pt-br/assuntos/drones/classes-de-drones
[3] https://www.gov.br/anac/pt-br/assuntos/drones/quantidade-de-cadastros

mecanismos para detectar e corrigir falhas e problemas durante o voo e pré voo, a fim de se prevenir acidentes aéreos como quedas, problemas relacionados à bateria, perda de sinal, entre outros problemas.

Um mecanismo bastante importante que já está presente em drones mais sofisticados são os sensores anti-colisão. Eles informam ao controlador de voo a presença de obstáculos e impedem o drone de avançar, evitando, assim, a colisão. Muitos drones da empresa *Da-Jiang Innovations* (DJI)[4] implementam esse sistema. Os sensores em geral são posicionados em locais específicos do corpo do drone e acabam não fornecendo um campo de detecção mais completo. Na Figura 1 é possível perceber o posicionamento dos sensores no modelo de drone *Mavic 3 Cine* fabricado pela DJI, que possui dez sensores distribuídos no corpo da aeronave e, mesmo com tantos sensores, o campo de visão não é inteiramente coberto.

Figura 1 - Ilustração dos sensores de obstáculos do Mavic 3

Fonte: DJI[5]

Contudo, vários drones não apresentam recursos suficientes para a prevenção de acidentes. Isso ocorre especialmente em drones de construção

[4] https://www.dji.com/br
[5] https://www.dji.com/br/mavic-3?site=brandsite&from=eol_guidance

caseira, como os VANTs construídos para executarem tarefas agrícolas, ou até mesmo usados para fins recreativos e profissionais. A falta desses recursos expõe os equipamentos e as pessoas a sérios riscos de acidentes. Logo, se faz necessário o desenvolvimento de uma ferramenta que possa garantir segurança no uso de drones construídos em casa ou aeronaves de origem não comercial, para que assim drones comerciais e aeronaves não comerciais possam compartilhar o mesmo espaço aéreo com segurança.

1.1 Motivação

Conduzir o voo de um VANT de forma segura não é uma tarefa fácil, uma vez que diversos fatores podem contribuir para um incidente, principalmente em um cenário em que drones comerciais e drone de fabricação caseira disputam o mesmo espaço aéreo. Diante disso, a maioria dos acidentes envolvendo VANTs estão relacionados a falhas humanas (WILD; MURRAY; BAXTER, 2016). Isso ocorre porque, na maioria das aeronaves usadas nos dias atuais, o controle operacional do dispositivo fica quase que inteiramente a cargo do piloto. Assim, um simples comando errado, seja ele por desconhecimento ou proposital, pode causar um grave acidente.

Outros fatores que contribuem para incidentes com drones estão ligados a problemas técnicos como, por exemplo, descarga precoce da bateria e danos nos motores. Esses fatores impossibilitam o piloto de realizar ações para reverter o estado da aeronave, o que acaba ocasionando a queda do drone. Já os fatores não técnicos também são agentes que podem influenciar em acidentes. Por exemplo, as condições climáticas podem interferir na comunicação do sistema de Global Positioning System (GPS) da aeronave e esse tipo de falha pode deixar o drone desorientado e o piloto pode perder o controle facilmente.

Pilotar drones em locais proibidos é mais um fator que pode ocasionar acidentes. Um caso[6] importante aconteceu em 2016, no aeroporto de Heathrow, em Londres, quando um avião comercial Airbus A320 da companhia *British Airways* foi atingido por um drone quando se aproximava para o pouso. Felizmente, nada grave aconteceu e a aeronave conseguiu pousar com sucesso. Outro fato[7] envolvendo drones e aviação civil ocorreu em 2022, no aeroporto de East Midlands, na

[6] https://www.bbc.com/portuguese/noticias/2016/04/160417_londres_choque_drone_airbus_fn
[7] https://www.bbc.com/news/uk-england-leicestershire-61763290

Inglaterra, quando dois aviões tiveram sua rota de pouso redirecionada para outro aeroporto, pois havia drones sobrevoando o aeroporto de destino.

Embora drones comerciais mais sofisticados apresentam recursos inteligentes que ajudam o piloto a conduzir o voo de forma segura, esses sistemas geralmente se limitam a detectar e evitar obstáculos, sem auxiliar o piloto durante realização de voos ou abordagens que inibam a decolagem caso algo não esteja funcionando corretamente. Em contrapartida, drones de fabricação caseira ou não comerciais, utilizados tanto para fins recreativos quanto profissionais, também necessitam de mecanismos que os auxiliem durante o voo, especialmente em pousos e decolagens. Esses sistemas devem ser capazes de detectar não apenas obstáculos, mas também falhas internas e condições externas que possam interferir no bom funcionamento do VANT, contribuindo para reduzir os riscos de acidentes.

Buscando solucionar essas questões, este trabalho propõe o desenvolvimento de um sistema fundamentado no uso de redes neurais, que se baseia em três pilares: detectar, avaliar e orientar o piloto sobre problemas que possam comprometer o funcionamento do drone. Esse sistema tem um foco especial em drones industriais e de fabricação caseira, oferecendo maior segurança e eficácia durante o voo, ao prever e agir proativamente diante de riscos identificados.

1.2 Objetivos

Esta seção descreve o objetivo geral e os objetivos específicos definidos para este TCC.

1.2.1 Objetivo Geral

O objetivo geral deste TCC consiste no desenvolvimento de um sistema semiautomático para a detecção e prevenção de falhas, com o objetivo geral de evitar acidentes aéreos com drones de fabricação manual para uso profissional ou industrial.

1.2.2 Objetivos Específicos

O trabalho tem ainda os seguintes objetivos específicos:

- fornecer aos pilotos de drone um sistema que age como um copiloto, auxiliando a condução do drone, analisando a telemetria e orientando o piloto a evitar acidentes;
- fornecer aos pilotos de drones maior precisão para detectar e obstáculos em multidireções e, se necessário, corrigir a rota;
- fornecer um sistema inteligente capaz de determinar, em tempo real, as condições da aeronave e as condições de falhas;
- fornecer um sistema capaz de aprender sozinho a lidar com novos cenários de risco, garantindo que o sistema sempre esteja atualizado;
- fornecer uma interface web universal para que o piloto possa acompanhar o funcionamento do sistema bem como visualizar os dados da telemetria.

1.3 Trabalhos Relacionados

Os drones atuais são compostos por sistemas com a capacidade de detectar e evitar obstáculos. Além disso, eles também têm a capacidade de retornar para o local de decolagem caso percam a comunicação com o sistema de rádio controle.

Os drones da marca DJI são os equipamentos que mais se destacam no quesito prevenção de acidentes e, a cada ano, surgem novos modelos com funcionalidades aperfeiçoadas e inovadoras. O primeiro drone dessa marca a contar com um sistema de prevenção de acidentes foi o *Phantom 4*[8], que possuía sensores frontais e traseiros para identificar obstáculos e impedir a colisão. Depois, a DJI lançou a linha *Mavic*[9], na qual os drones dispunham de sensores mais sofisticados e algoritmos que ajustavam a rota da aeronave, até mesmo com casos de retorno automático para evitar acidentes. Outro produto desenvolvido pela DJI, mas que foi descontinuado, é o *Guidance*[10], que foi um sistema desenvolvido para ser montado em drones que não possuíam recursos para a prevenção de acidentes. O sistema era composto basicamente por uma estrutura retangular com sensores do tipo sonar e infravermelho dispostos estrategicamente no corpo da estrutura e era montado na parte superior dos drones. Além de prevenir colisões, o *Guidance* também contava com recursos que melhoravam a estabilidade do drone. No entanto, o sistema era

[8] https://www.dji.com/br/phantom-4-pro
[9] https://www.dji.com/br/mavic
[10] https://www.dji.com/br/guidance

grande e um pouco complexo para se instalar. Além disso, ele afetava de forma negativa a estética do drone.

O *Skydio 2 Plus*[11] é outro drone com recursos avançados para a prevenção de acidentes envolvendo colisões. Ele possui um avançado sistema de escaneamento, que cria em tempo real um mapa tridimensional do ambiente no qual o Skydio está operando. Por meio desse sistema, o drone consegue evitar objetos de forma precisa e rápida.

Contudo, os recursos preventivos usados hoje em dia nos drones comerciais são de projeto fechado e de uso exclusivo dos fabricantes e modelos específicos de drones, o que deixa as aeronaves de construção própria desprotegidas e propicia a ocorrência de falhas e, consequentemente, acidentes aéreos. Isso ocorre porque a maioria dos controladores de voo usados para a montagem de VANTs oferecem apenas os recursos básicos de estabilidade e modos de voo, não oferecendo estratégias para detectar problemas e até mesmo aplicar correções de falhas automaticamente.

1.4 Metodologia

Para ter êxito no desenvolvimento do projeto foi preciso definir um planejamento que descrevesse os processos de desenvolvimento do sistema proposto que uma vez executado seria possível alcançar o objetivo geral. Para tal desenvolvimento é preciso conhecer todos os propósitos do projeto, bem como conhecer de forma clara os seus objetivos. Assim, para implementar o projeto e desenvolver este TCC, foi definido o desenvolvimento das seguintes atividades:

- **Estudo sobre os riscos envolvendo o uso de drones (A_1)**: durante a execução dessa atividade foi realizado um levantamento sobre os riscos que um drone apresenta para quem está pilotando a aeronave e também para as pessoas que estão na área de operação do drone. Além dessas informações, também foram estudadas as principais falhas técnicas que podem ocasionar a queda do VANT e, consequentemente, um acidente. Durante tal estudo foram analisados relatos de pilotos e matérias sobre incidentes com drones;

[11] https://www.skydio.com/skydio-2-plus

- **Projeto da solução (A_2)**: nesta atividade foi projetada uma solução para reduzir acidentes e falhas com drones. Durante a sua execução foram levantados alguns pontos de como poderia ser feito e como seria feito;
- **Levantamento de requisitos (A_3):** durante a execução desta atividade foram levantados os requisitos funcionais necessários para alcançar o objetivo geral e os objetivos específicos, com o foco em implementar a solução proposta na atividade A_2;
- **Elaboração do documento (A_4)**: nesta atividade foi realizada a escrita do documento deste trabalho de conclusão de curso;
- **Definir a arquitetura de sistema do projeto (A_5):** o objetivo desta atividade era construir uma arquitetura de sistema que, uma vez implementada, permitiria alcançar todos os objetivos do projeto;
- **Definir o hardware embarcado (A_6)**: nesta atividade foi realizado o estudo de todos as partes físicas necessárias para executar os scripts do sistema;
- **Implementação o módulo de controle primário (A_7)**: durante a execução desta atividade foi realizado o desenvolvimento do módulo do controle baseado em redes neurais usado para avaliar o voo;
- **Implementação o módulo secundário (A_8)**: esta atividade consistiu no desenvolvimento do módulo de controle secundário usado para executar as ações do controle primário;
- **Implementação da dashboard dinâmica(A_9)**: durante a execução desta atividade foi realizado o desenvolvimento da última parte do sistema usada para exibir os dados de telemetria e processamento neural para o piloto.

1.5 Cronograma de atividades

As atividades descritas na seção 1.4 deste documento foram devidamente distribuídas no Quadro 1 abaixo para o desenvolvimento nas disciplinas *Trabalho de Conclusão de Curso 1* (TCC 1) e *Trabalho de Conclusão de Curso 2* (TCC 2).

Quadro 1 - Cronograma de atividades do TCC

Atividades	OUT	NOV	DEZ	JAN	FEV	MAR
A1	X					
A2	X					
A3		X				
A4			X			
A5				X		
A6				X		
A7				x	X	
A8					x	X
A9					x	X

Fonte: Autor

1.6 Organização do Documento

O restante deste documento está organizado da seguinte forma: o Capítulo 2 descreve a fundamentação teórica, apresentando os conceitos e tecnologias que foram importantes para a solução da problemática abordada neste trabalho; o Capítulo 3 descreve a solução proposta, apresentando os principais artefatos produzidos durante o seu desenvolvimento, e descrevendo também o processo de implementação; o Capítulo 4 descreve o processo de desenvolvimento de cada parte do sistema; o Capítulo 5, por fim, apresenta as conclusões e as considerações finais.

2. FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são abordados os conceitos, definições e tecnologias que foram utilizados no desenvolvimento deste trabalho. Inicialmente, o capítulo fala sobre o hardware necessário para executar o sistema e, logo em seguida, o capítulo apresenta as tecnologias usadas no desenvolvimento dos códigos.

2.1 O hardware

Essa seção aborda o hardware do sistema Air CoPilot e sua importância na execução do sistema, o que inclui todas as partes físicas envolvidas na montagem.

2.1.1 Placa Orange Pi Zero 2W

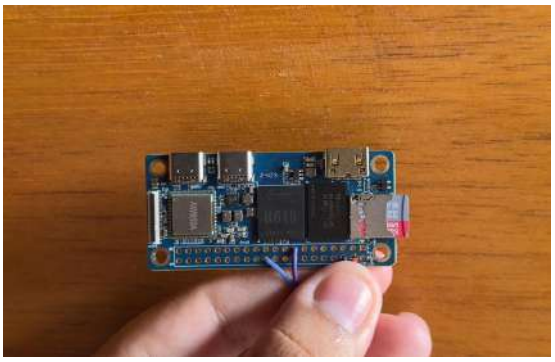
A Orange Pi Zero 2W é uma placa compacta e poderosa, muito usada em projetos que exigem um hardware pequeno, mas com boa capacidade de processamento. Seu design é inspirado na famosa placa Raspberry Pi Zero 2W, mas traz algumas vantagens interessantes, como um processador melhor (Allwinner H618), que oferece um desempenho superior em comparação ao Raspberry. Além disso, ela conta com suporte a Wi-Fi de banda dupla (2.4 GHz e 5 GHz) e Bluetooth 5.0, tornando-a uma excelente opção para servidores compactos e até mesmo pequenos sistemas embarcados.

Outro ponto que torna essa plaquinha atrativa é a quantidade de memória RAM disponível, podendo ser encontrada em versões de 1GB até 4GB DDR4, algo essencial para rodar sistemas operacionais como Armbian, Debian ou Ubuntu de forma mais fluida. Como não há um slot para SSD na placa, é usado um micro SD para gravar o sistema operacional e também servir de armazenamento interno do sistema. Outro benefício de se usar essa placa é o seu conector de 24 pinos compatível com GPIO, permitindo interações com sensores, portas UART, motores e outros periféricos de forma flexível, o que a torna uma boa opção para automação e controle de sistemas autônomos.

Em termos de conectividade, a placa traz duas portas USB-C para alimentação/comunicação, um conector mini HDMI, possibilitando seu uso como um mini PC básico se necessário. Com todos esses recursos, a Orange Pi Zero 2W se destaca como uma opção compacta, mas poderosa e barata, atendendo tanto a desenvolvedores quanto a entusiastas que buscam um hardware flexível para

diferentes aplicações. Na Figura 2 pode-se comprovar o tamanho compacto da placa Orange Pi Zero 2W.

Figura 2 - Mini computador Orange Pi Zero 2W

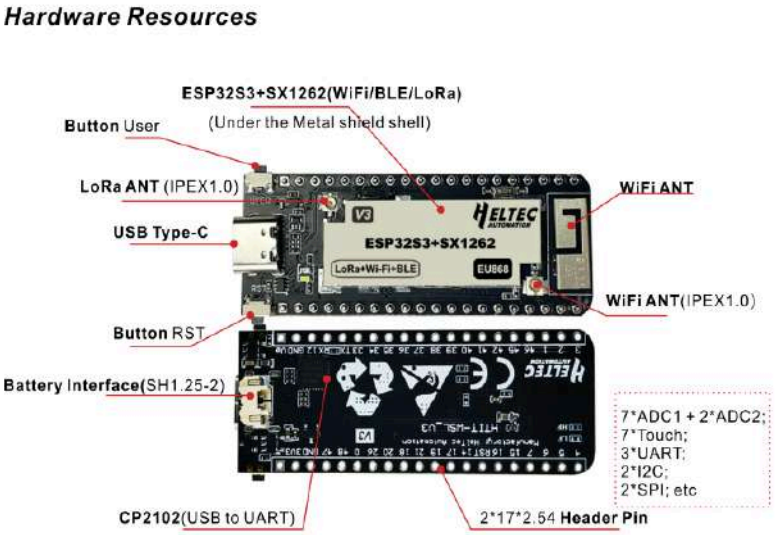


Fonte: Autor

2.1.2 Heltec Wireless Stick Lite (HTIT-WSL)

A placa HTIT-WSL é equipada com o módulo ESP32-S3, um poderoso microcontrolador da Espressif projetado para aplicações de sistemas embarcados com baixo consumo de energia. O ESP32-S3 conta com um processador dual-core Xtensa LX7, que opera a uma frequência de até 240 MHz, o que proporciona excelente capacidade de processamento para tarefas que exigem resposta rápida, como controle de voo, análise de dados de sensores e comunicação sem fio. Além disso, ela possui suporte a Wi-Fi 4 (802.11 b/g/n) e Bluetooth 5 (BLE), permitindo uma conectividade versátil e eficiente. A HTIT-WSL é também equipada com um módulo LoRa (Long Range), que é uma tecnologia de modulação de rádio de baixa potência, mas com longo alcance, capaz de transmitir dados a distâncias de até 15 km, dependendo das condições do ambiente e da configuração do sistema, ideal para transmitir pequenos pacotes de dados de forma eficiente e estável. O módulo LoRa da HTIT-WSL é projetado para operar em frequências sub-GHz, como 868 MHz ou 915 MHz, oferecendo eficiência espectral ao mesmo tempo que consome pouca energia.

Figura 3 - Placa HTIT-WSL



Fonte: *Heltec*[12]

Na Figura 3 é possível visualizar em detalhes a placa HTIT-WSL, destacando partes importantes como os conectores das antenas e os pinos GPIO. Além disso, a HTIT-WSL conta com interfaces SPI, I2C e UART, permitindo a comunicação com uma ampla variedade de sensores e dispositivos externos, o que a torna extremamente versátil para aplicações embarcadas. Seu suporte a entrada e saída de GPIOs programáveis permite que ela seja utilizada para controlar sistemas robóticos ou de automação.

Graças ao seu baixo consumo de energia, a HTIT-WSL é ideal para dispositivos alimentados por bateria, garantindo longa autonomia mesmo em aplicações que exigem comunicação contínua. Quando combinada com o poder de processamento do ESP32-S3, sua capacidade de análise de dados em tempo real e a comunicação eficiente via Wi-Fi, Bluetooth e LoRa fazem dela uma excelente escolha para sistemas embarcados de telemetria remota.

2.1.3 LiDAR LD14P (Light Detection and Ranging)

O LiDAR LD14P é um sensor avançado utilizado para medir distâncias e mapear o ambiente ao redor, disparando feixes de luz infravermelho que, ao se chocarem com objetos no raio de cobertura do sensor, são refletidos de volta e computados na forma de uma coordenada polar (ângulo e distância). Com uma

[12] https://heltec.org/project/wireless-stick-lite-v2

faixa de detecção de até 8 metros e um campo de visão de 360 graus, o LD14P oferece um excelente desempenho na detecção de obstáculos incrivelmente rápido e preciso pois, graças a sua cabeça óptica giratória, ele consegue fazer uma cobertura de 360 graus operando a uma frequência de 6 a 8Hz. Isso significa que o sensor pode gerar uma nuvem de 5760 pontos por segundo operando em uma frequência de 8Hz.

O funcionamento básico de um sensor LiDAR 2D envolve a emissão de pulsos de laser em diferentes direções dentro de um plano bidimensional. Ao atingir um objeto, parte da luz é refletida de volta para o sensor, permitindo que o tempo de retorno seja medido. Como a velocidade da luz é conhecida, é possível calcular com precisão a distância entre o sensor e o objeto detectado. Os sensores LiDAR 2D operam girando um feixe de laser em um único plano horizontal, cobrindo 360 graus, Isso permite a criação de um perfil detalhado do ambiente ao redor.

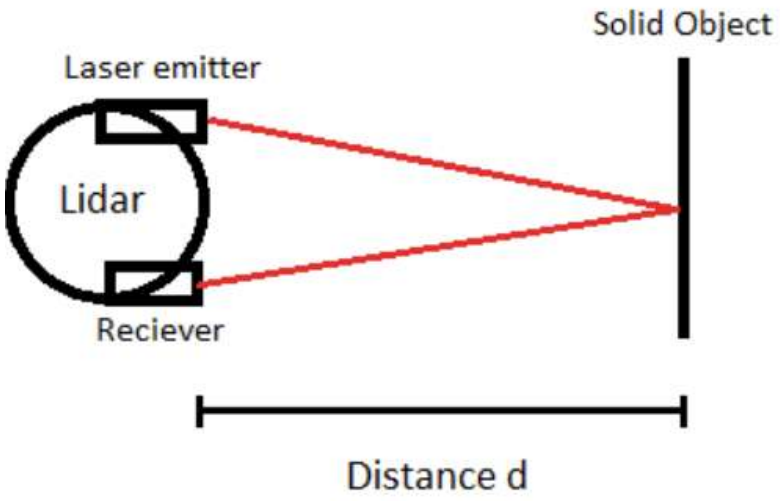
Entre as principais especificações de um sensor LiDAR 2D, destacam-se:

- **alcance**: varia de modelo para modelo e geralmente é calculada em milímetros. No caso do sensor LD14P, o alcance varia de 10 até 8000 mm;
- **resolução angular**: define a precisão das leituras e logo interfere no número de amostragem do sensor e varia com base na taxa de varredura;
- **taxa de varredura**: determina a quantidade de voltas que a cabeça rotatória irá executar por segundo. No caso do LD14P, ele vem configurado de fábrica para operar em uma frequência de 6 Hz. Isso indica uma taxa de 6 varreduras por segundo, mas pode ser configurado para até 8 Hz;
- **comprimento de onda do laser**: com um valor que comumente varia entre 850 e 1550 nm, o comprimento da onda do laser influencia a interação com diferentes materiais, pois há materiais mais reflexivos e menos reflexivos;
- **resistência à luz externa**: um ponto muito importante que deve ser levado seriamente em consideração, principalmente para aplicações externas, é a tolerância à luz externa. Isso vai indicar o quanto de influência luminosa o sensor vai suportar sem influenciar nos

resultados. No LD14P, o nível máximo de luz que ele pode trabalhar é de 80 klux, o que indica que a luz do dia, na maioria dos cenários, não afetaria o sensor.

A Figura 4 ilustra o princípio de funcionamento de um sensor LiDAR, mostrando o percurso do laser até um objeto e o seu retorno ao sensor. O emissor de laser envia um feixe de luz em direção ao ambiente e, ao encontrar um obstáculo, parte desse feixe é refletida de volta para o receptor do sensor. Com base no tempo decorrido entre a emissão e a recepção do sinal refletido, o sistema calcula a distância entre o sensor e o objeto, utilizando a velocidade da luz como referência.

Figura 4 - Funcionamento de um sensor LiDAR 2D



Fonte: ROS-based implementation of a model car with a LiDAR and camera setup[13]

[13] https://uu.diva-portal.org/smash/get/diva2:1773842/FULLTEXT01.pdf

2.2 O software

Essa seção aborda o software do sistema Air CoPilot e sua importância na execução do sistema, o que inclui todas as tecnologias de desenvolvimento utilizadas.

2.2.1 Linguagem de programação Python

O Python é uma linguagem de programação de alto nível, interpretada, reconhecida por sua simplicidade, legibilidade e versatilidade. Criada por Guido van Rossum no final dos anos 1980, foi oficialmente lançada em 1991, com o objetivo de ser uma linguagem acessível, de fácil aprendizado e que ao mesmo tempo fosse poderosa o suficiente para tarefas complexas. Van Rossum teve a intenção de criar uma linguagem que tornasse a programação mais intuitiva e eficiente, permitindo ao desenvolvedor focar mais na resolução de problemas do que em detalhes técnicos. O nome da linguagem foi uma inspiração do grupo de comédia britânico Monty Python, como uma forma de refletir a ideia de que a programação pode ser algo divertido e criativo.

Desde o seu lançamento, o Python passou por atualizações importantes, tornando-se uma das linguagens de programação mais populares em diversas áreas, como tecnologia e acadêmico. O Python 2, lançado nos anos 2000, foi amplamente adotado, mas a chegada do Python 3, em 2008, trouxe melhorias substanciais, incluindo o gerenciamento de memória otimizado, manipulação aprimorada de strings e suporte expandido para programação assíncrona. Com a descontinuação do Python 2 em 2020, o Python 3 se consolidou como o novo padrão, oferecendo maior compatibilidade e eficiência para projetos futuros.

Uma das principais razões para o sucesso contínuo do Python é a sua biblioteca padrão extensa, que possibilita aos desenvolvedores realizar uma ampla gama de tarefas sem a necessidade de criar soluções do zero. Além disso, o Python é altamente compatível com outras linguagens como C, C++ e Java, o que o torna flexível e capaz de ser otimizado para situações que demandam alto desempenho. Ele é amplamente utilizado em várias áreas, incluindo desenvolvimento web, automação, ciência de dados, inteligência artificial, computação gráfica, segurança cibernética e até mesmo no desenvolvimento de jogos.

2.2.2 Rede Neural de Classificação

As redes neurais de classificação são modelos amplamente utilizados em aprendizado supervisionado para resolver problemas de categorização de dados. Esses sistemas são compostos por camadas de neurônios artificiais, que tentam simular o comportamento do cérebro humano, e tem como objetivo principal processar grandes volumes de dados e os interpretar como se fosse um humano. De acordo com Goodfellow et al. (2016), as redes neurais são especialmente adequadas para detectar padrões complexos em grandes volumes de dados, algo que seria extremamente difícil ou até impossível de realizar manualmente.

A principal característica das redes neurais de classificação é a sua capacidade de aprender funções complexas que mapeiam entradas para as saídas desejadas, sendo treinadas com dados rotulados e reais que representam os padrões. Durante o processo de treinamento, o modelo ajusta os seus pesos e viéses utilizando algoritmos de otimização como o Gradiente Descendente, que visa minimizar a função de custo que mede o erro entre a saída prevista e a saída real (BENGIO, 2012). Entre os principais algoritmos usados para redes neurais de classificação, destacam-se o Perceptron Multicamadas (MLP), a Rede Neural Convolucional (CNN), amplamente utilizada em visão computacional, e a Rede Neural Recorrente (RNN), indicada para dados sequenciais, como séries temporais e texto (LECUN et al., 2015).

No contexto de implementação, o Python vem sendo uma das linguagens mais populares para o desenvolvimento de RN devido à sua simplicidade e à ampla gama de bibliotecas de aprendizado de máquina. Bibliotecas como TensorFlow, Keras e PyTorch oferecem ferramentas robustas e ao mesmo tempo eficientes para construir, treinar e avaliar modelos de redes neurais. O TensorFlow, por exemplo, é uma plataforma de código aberto e escalável, com suporte a computação distribuída e integração com dispositivos de processamento gráfico (GPU), acelerando significativamente o treinamento de modelos com grandes volumes de dados (ABADI et al., 2016). Já o Keras, é uma API de alto nível que executa sobre o TensorFlow, oferece uma interface intuitiva e fácil para desenvolver redes neurais rapidamente pois seu principal objetivo é implementar o TensorFlow de forma simples. O PyTorch, por sua vez, se destaca pela flexibilidade e facilidade de uso, especialmente em ambientes de pesquisa, permitindo que os desenvolvedores alterem o modelo de forma dinâmica durante o treinamento.

2.2.3 Algoritmo de clusterização DBSCAN

O algoritmo DBSCAN (Density-Based Spatial Clustering of Applications with Noise) é uma técnica usada para agrupar dados em espaços de alta densidade. Ele basicamente recebe um conjunto de dados e devolve um conjunto de grupos. Ao contrário de algoritmos tradicionais de clusterização, como o K-means, que exigem que o número de clusters seja definido previamente, o DBSCAN é baseado em densidade, o que significa que ele pode identificar clusters de forma adaptativa, sem a necessidade de especificar o número de grupos a priori (ESTER et al., 1996). O DBSCAN agrupa pontos de dados em regiões de alta densidade, ao mesmo tempo em que os pontos em regiões de baixa densidade são considerados outliers (pontos de ruídos).

O funcionamento do DBSCAN é baseado em dois parâmetros principais: ε (epsilon), que configura a distância máxima entre dois pontos para que eles possam ser considerados como parte do mesmo cluster ou grupo, e *MinPts*, que define o número mínimo de pontos que devem estar dentro do raio epsilon para formar um cluster (SILVA, 2007). O algoritmo começa com a escolha de um ponto arbitrário e verifica se ele possui o número mínimo de pontos dentro do raio ε. Se tiver, um novo cluster é iniciado, e o algoritmo continua expandindo esse cluster até que todos os pontos acessíveis a partir do ponto inicial sejam agrupados. Se o ponto não tiver pontos suficientes em sua vizinhança, é considerado ruído e o algoritmo passa para o próximo ponto não processado.

Uma das principais vantagens do DBSCAN é a sua capacidade de detectar clusters de forma não esférica, o que o torna ideal para lidar com dados complexos, onde os clusters podem ter formas irregulares, algo que métodos como K-means não conseguem fazer de maneira eficiente. Além disso, o DBSCAN pode lidar bem com outliers, ignorando pontos que não pertencem a nenhum cluster e não os forçando a se agruparem com outros pontos. Isso o torna especialmente útil em áreas como análise de dados espaciais e detecção de padrões, onde a presença de pontos aleatórios pode distorcer os resultados em outras técnicas de clusterização.

A implementação do DBSCAN pode ser feita com Python por meio de diversas bibliotecas, sendo a mais popular a scikit-learn, que fornece uma implementação eficiente e de fácil utilização do algoritmo.

O uso de Python para implementar o DBSCAN é vantajoso devido à sua simplicidade e à ampla gama de ferramentas e bibliotecas disponíveis para análise e visualização de dados, como NumPy, Pandas, Matplotlib e Seaborn. A biblioteca scikit-learn implementa o algoritmo de forms altamente otimizada, permitindo ao usuário facilmente ajustar os parâmetros de epsilon e MinPts, além de oferecer ferramentas para avaliação e visualização dos resultados de clusterização, como a média da distância intra-cluster e a qualidade de agrupamento.

2.2.4 Framework Arduino Espressif

O framework Arduino Espressif é uma plataforma de desenvolvimento baseada em microcontroladores da Espressif, como o ESP32. Ele é amplamente utilizado na criação de projetos de sistemas embarcados e de internet das coisas. O uso do framework Arduino, que é uma interface simplificada para programar microcontroladores, facilita a implementação de projetos eletrônicos, tornando-os acessíveis até para iniciantes. Ao combinar a flexibilidade do Arduino com o poder do ESP32, o framework permite que desenvolvedores criem soluções com conectividade sem fio de maneira rápida e eficaz.

Para o desenvolvimento das soluções e projetos com ESP32 é usado o plugin PlatformIO para a IDE Visual Studio Code, que permite programar os microcontroladores Espressif de maneira mais eficiente e com uma integração mais poderosa, oferecendo suporte a diversos ambientes de desenvolvimento e compiladores. O uso do PlatformIO também facilita a integração com ferramentas de controle de versão, como o Git, e oferece funcionalidades como debugging, monitoramento serial e atualização OTA. Além disso, o framework permite que as bibliotecas sejam facilmente estendidas e personalizadas conforme as necessidades específicas do projeto. Por exemplo, é possível integrar facilmente sensores de temperatura, movimento, ou até mesmo módulos LoRa (Long Range) para comunicação em longas distâncias, tornando o sistema mais flexível e adequado para ambientes variados. As comunicações seguras também são suportadas nativamente, com protocolos como TLS e SSL, essenciais para proteger dados trocados entre dispositivos e servidores (SILVA et al., 2019).

3. A SOLUÇÃO PROPOSTA

Este capítulo apresenta a implementação do sistema proposto por este TCC. De início, a seção 3.1 descreve os stakeholders do sistema *Air CoPilot.* A seção 3.2 descreve os requisitos funcionais. Em sequência, a seção 3.3 apresenta a arquitetura do sistema, na qual é possível observar todos os módulos que o compõem.

3.1 Stakeholders

Os *stakeholders* do sistema desenvolvido neste TCC são basicamente os pilotos de drones. No entanto, eles podem ser divididos em dois grupos: os pilotos experientes e os pilotos em fase de treinamento.

Os dois grupos de *stakeholders* têm necessidades e expectativas diferentes em relação ao sistema. Os pilotos experientes podem usar o sistema para complementar as suas habilidades, fornecendo informações adicionais sobre o estado do drone e possíveis riscos durante o voo. Para eles, o sistema pode atuar como uma camada extra de segurança, alertando sobre falhas e auxiliando na tomada de decisões rápidas. Já os pilotos em fase de treinamento podem se beneficiar ainda mais do sistema, pois eles podem fornecer orientações em tempo real, ajudando a melhorar a curva de aprendizado e corrigir erros durante o voo. O Air CoPilot pode agir como um instrutor virtual, guiando o piloto e evitando acidentes enquanto ele ainda desenvolve as habilidades necessárias para controlar o drone de forma segura e eficiente.

3.2 Requisitos Funcionais

Os requisitos funcionais deste trabalho foram definidos pelo autor, levando em consideração a sua experiência prévia com a montagem e pilotagem de drones. De forma a permitir que os objetivos gerais e específicos do trabalho sejam alcançados, foram definidos os seguintes requisitos funcionais:

- **Navegação guiada com Machine Learning (RF1)**: o *Air CoPilot* deve garantir o voo seguro por meio da análise e classificação neural dos dados de telemetria e identificação de obstáculos;
- **Desvio de controle (RF2)**: o Air CoPilot poderá desviar as instruções do controle RC recebidas pelo sistema de rádio e introduzir suas

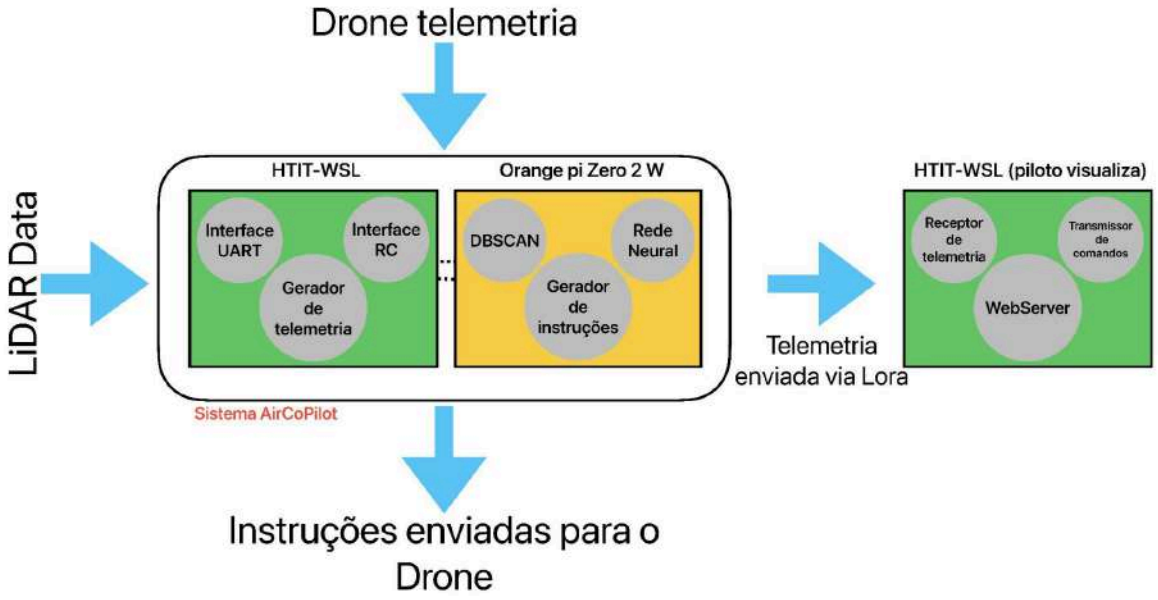
próprias instruções de controle para realizar manobras, após a realização das instruções o sistema retoma o fluxo normal;

- **Ações supervisionadas (RF3)**: o sistema auxiliará o piloto na decolagem e na execução do voo seguro;
- **Visualização da telemetria e status do sistema (RF4)**: o sistema contará com um aplicativo web para exibir os dados do voo e orientar o piloto;
- **Otimização automática do sistema (RF5)**: o sistema coletará dados do voo para criar conjuntos de dados sempre atualizados e prontos para o treinamento da rede neural de classificação, fazendo com que o sistema aprenda “sozinho” a classificar o estado de voo.

3.3 Arquitetura do Sistema

Para o desenvolvimento do projeto, foi necessário desenvolver uma arquitetura de sistema, pois ela fornece um entendimento completo do sistema. Na Figura 5, é possível visualizar a arquitetura projetada para o *Air CoPilot*, na qual há a presença de dois módulos principais compostos por submódulos. Esses módulos são responsáveis por processar as informações captadas pelos sensores do drone, interpretar os dados e gerar comandos corretivos ou sugestões para o piloto. O primeiro módulo está focado na percepção e análise do ambiente, utilizando sensores como LiDAR, GPS e IMU para coletar informações e algoritmos de inteligência artificial para interpretar os dados. O segundo módulo é responsável pelo controle e assistência ao piloto, gerenciando a recepção da telemetria e garantindo que todos os dados sejam atualizados em tempo real na interface Web do sistema.

Figura 5 - Arquitetura do sistema Air CoPilot



Fonte: Autor

No sistema *Air CoPilot,* a placa HTIT-WSL forma um módulo que tem a função coletar os dados de telemetria dos sensores e periféricos do drone e posteriormente enviá-los para processamento no módulo do Orange Pi. A comunicação entre esses módulos é feita de forma física usando o protocolo de comunicação UART para trocar informações, enviando e recebendo dados.

Além disso, a HTIT-WSL também é responsável por ler os comandos de rádio enviados pelo piloto em solo e os encaminhar para o controlador de voo, servindo como uma ponte de comunicação. Essa funcionalidade permite inserir comandos extras ou manipular os comandos recebidos para que o controlador de voo possa executar, o que inclui instruções geradas pelo *Air CoPilot* ou comandos personalizados do piloto.

Por fim, o módulo HTIT-WSL gera pacotes de dados contendo a telemetria atualizada com informações do drone e instruções processadas pelo *Air CoPilot* e os envia para o piloto em solo usando o sistema de transmissão sem fio LoRa integrado à placa.

O módulo Orange pi é então alimentado por todos os dados de telemetria coletados pela HTIT-WSL, que incluem: tensão da bateria, altitude, velocidade, distância do piloto, quantidade de satélites, qualidade do sinal de rádio e dados do LiDAR 2D fixado no corpo do drone. O processamento ocorre no Orange pi de

forma paralela, enquanto o DBSCAN processa a nuvem de pontos gerada pelo LiDAR e retorna os clusters (obstáculos) detectados. Então, a rede neural de classificação realiza previsões do estado atual do drone com base nos dados de entrada. Os status gerados pela rede passam por um filtro lógico para determinar se é necessário gerar alguma instrução nova para o piloto. Por exemplo: se o sistema identificar um obstáculo, o *Air CoPilot* vai instruir o piloto a desviar a rota e evitar o objeto detectado e essas instruções são definidas pelo gerador de instruções.

O módulo de visualização do sistema, é composto por uma segunda placa HTIT-WSL, que é responsável por receber os pacotes de dados de telemetria enviados pelo sistema *Air CoPilot*. Os dados são recebidos e estruturados no formato JSON e enviados para uma página Web para visualização. O módulo de visualização também gera um ponto de acesso Wi-Fi e roda um servidor Web para servir a página e todos os recursos de visualização e controle do sistema.

4. IMPLEMENTAÇÃO

Este capítulo descreve de forma detalhada o processo de desenvolvimento do sistema *Air CoPilot.* Ele foi dividido em duas seções: montagem do hardware e desenvolvimento do software.

4.1 Montagem do Hardware

Esta seção descreve como foi realizada a montagem do hardware que compõe o sistema.

4.1.1 Módulo de processamento e drone de testes

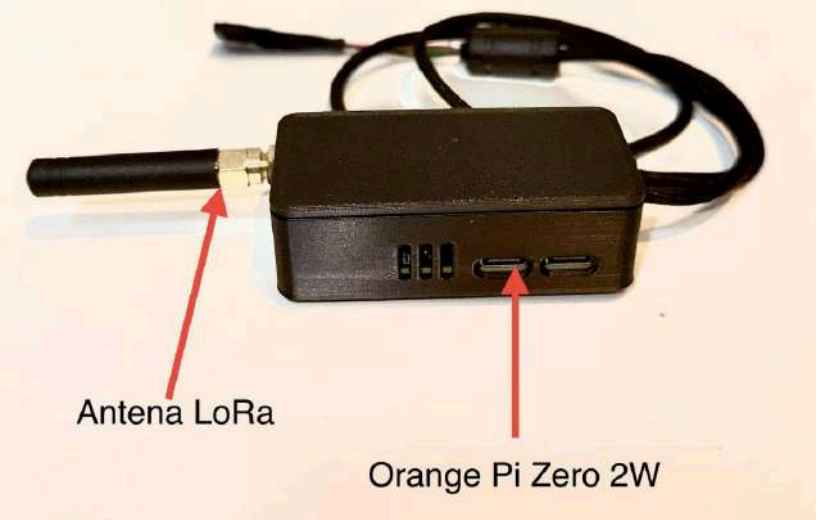
Para transportar junto ao drone todo o sistema de software desenvolvido, foi necessário realizar a montagem de um módulo de processamento usando as duas placas controladoras descritas na arquitetura proposta. As placas usadas para essa montagem foram o microcontrolador HTIT-WSL e o microcomputador Orange Pi Zero 2W. Para que ambas as placas trocassem informações foi realizada a soldagem dos cabos da comunicação UART o TX (Transmitter) e RX (Receiver). Tanto a placa HTIT-WSL quanto a Orange Pi suportam a comunicação UART, e a ligação foi realizada da seguinte forma: pino TX da HTIT-WS ligado no pino RX da Orange Pi e pino RX da HTIT-WS ligado no pino TX da Orange Pi.

Para que a placa HTIT-WS possa ler os dados de telemetria do drone foi preciso desenvolver um cabo de conexão capaz de ligar a placa ao sistema de controle do drone. Para isso foi escolhido um cabo de cinco vias, ou seja, um cabo com outros cinco cabinhos integrados. Cada via do cabo foi soldada em uma GPIO da placa HTIT-WS, distribuídas da seguinte forma: uma para o sinal recebido receptor, outra para a saída do sinal, outra para os dados do GPS, outra para os dados da bateria e uma última para a qualidade de sinal do receptor RC.

A preparação do microcomputador Orange Pi foi feita instalando o sistema operacional em um cartão micro SD, que nada mais é que uma distribuição Linux customizada para funcionar nesse tipo de computador. A versão usada foi uma distribuição Ubuntu composta apenas com os recursos essenciais para o funcionamento do sistema. Após isso, foi realizada a instalação das dependências e configuração da comunicação UART. Por fim, as duas placas foram organizadas em uma pequena caixa plástica feita em impressora 3D. Nas Figuras 6 e 7 é possível

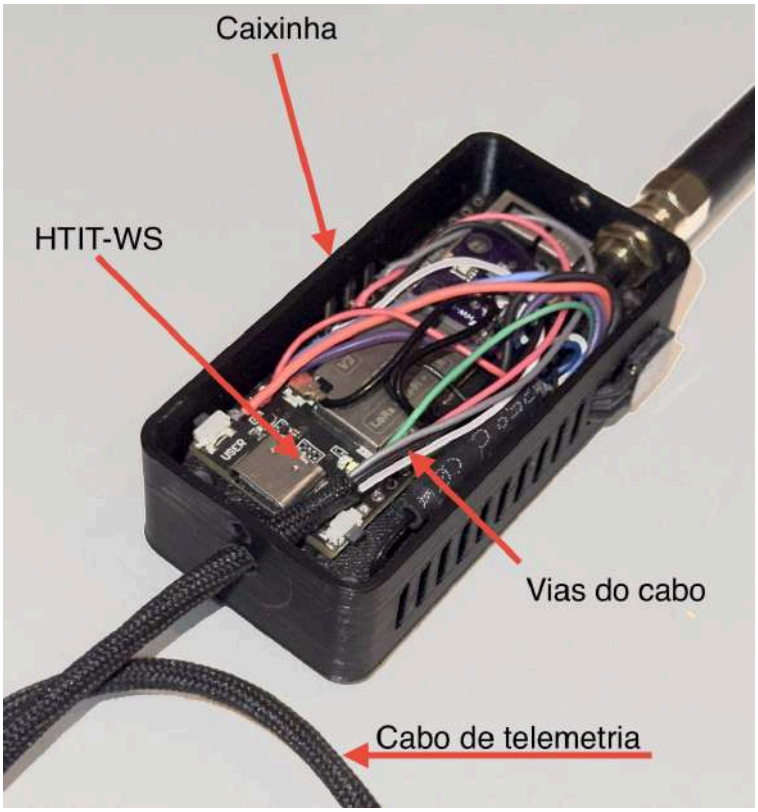
identificar, respectivamente, a organização dos cabos e o posicionamento das placas no interior da caixa.

Figura 6 - Módulo de processamento fechado



Fonte: Autor

Figura 7 - Módulo de processamento aberto



Fonte: Autor

Com o módulo de processamento devidamente montado, foi realizada a instalação em um drone de médio porte, projetado e construído pelo autor e com 90% das peças feitas em impressora 3D. Esse VANT mede 110 centímetros de comprimento entre os braços, 45 centímetros de altura e pesa 4.5 kg. Ele se enquadra na Classe 3 da ANAC, abrangendo drones com peso entre 250 g e 25 kg. O drone usado nos testes usa uma controladora de voo fabricada pela DJI de modelo Naza-M V2. Esse controlador proporciona alta confiança e estabilidade de voo, e também possui um sistema GPS de alta sensibilidade. Por fim, ele também conta com um sistema de rádio controle profissional de longo alcance.

Ao se observar a Figura 8, pode-se constatar o grande tamanho do drone de testes comparado a um drone comercial de modelo Phantom 3 fabricado pela DJI. Contudo, a pilotagem desse drone requer atenção e cuidados pois, por conta do seu peso, ele pode causar danos leves e graves em caso de acidentes. Essas características fazem desse drone um equipamento ideal para acoplar o sistema *Air CoPilot* e testar o seu funcionamento.

Figura 8 - Drone usado nos testes do sistema

Fonte: Autor

4.1.2 Sensor de obstáculos LiDAR

O sensor LD14P, responsável por escanear o entorno do drone e verificar a existência de obstáculos próximos, envia dados dos pontos capturados por meio da interface de comunicação UART TX e RX. Ele também recebe comandos que são usados para ajustar a frequência de leituras, a velocidade e o controle do motor que gira a cabeça óptica do sensor. Para que o sensor tenha uma visão 360 graus do

entorno do drone sem nenhum tipo de interferência, foi escolhido o topo da carenagem protetora do drone, pois é o ponto mais alto da montagem e também é livre de outras peças que possam interferir na leitura do sensor. Na Figura 9 é possível identificar o ponto exato escolhido para fixar o sensor.

Figura 9 - Local onde foi feita a instalação do LiDAR

Fonte: Autor

Com o local de instalação do sensor definido, foi realizada a montagem. Para que o sensor fosse devidamente acomodado na carenagem, foi preciso alterar o projeto 3D da carenagem de proteção de forma que ela tivesse todas as furações necessárias para o encaixe perfeito do LiDAR. Na Figura 10 é possível visualizar como ficou a instalação do sensor na carenagem modificada. Também é possível perceber que o campo de visão do sensor não foi obstruído, garantindo leituras livres de objetos indesejados.

Figura 10 - Sensor LiDAR devidamente instalado

Fonte: Autor

Para finalizar a montagem do sistema no drone foi feita a fixação do módulo de processamento no trem de pouso da aeronave. Essa localização foi escolhida pelo fato do trem de pouso ser um local de ventilação natural, fazendo com que as placas de controle mantenham uma temperatura de trabalho controlada pois, quando o sistema está em pleno funcionamento, a temperatura da CPU da Orange Pi pode passar dos 88 graus; O microcontrolador HTIT-WS também eleva a sua temperatura quando começa a trafegar dados via LoRa, e acaba contribuindo para um aquecimento generalizado de todo o sistema. Então, para evitar travamentos, a solução foi dissipar bem o calor por meio da ventilação. Observando-se a Figura 11, é possível identificar o local de fixação do módulo no trem de pouso.

Figura 11 - Local de fixação do módulo de processamento

Fonte: Autor

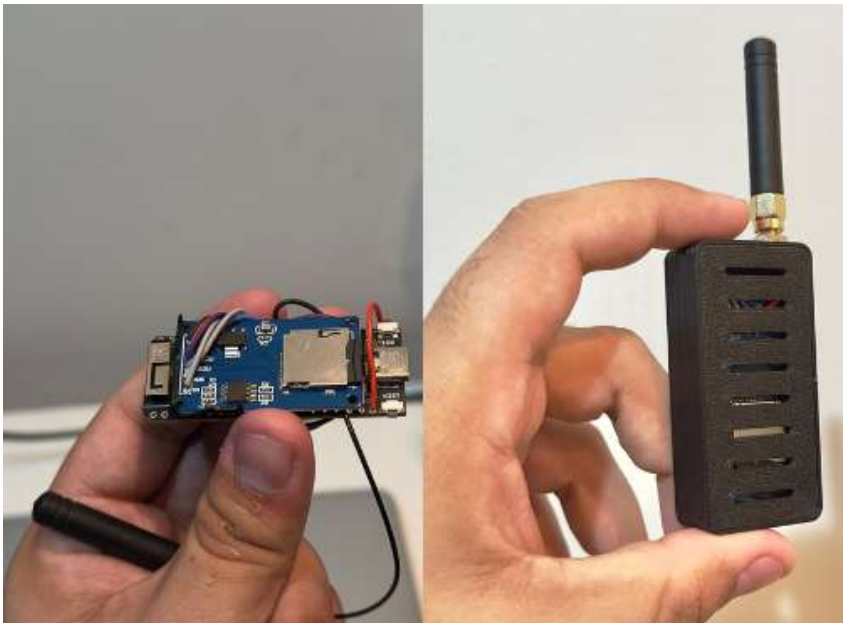
4.1.3 Módulo receptor de telemetria e visualização

O receptor de telemetria do sistema *Air CoPilot* é responsável por receber os dados processados pelo sistema e também os dados de telemetria em tempo real do drone. Ele foi desenvolvido com base no microcontrolador HTIT-WS e em um módulo leitor de cartão micro SD. O leitor de cartão foi ligado fisicamente no microcontrolador, seguindo a interface de comunicação SPI (Serial Peripheral Interface) usada pelo leitor. Nesse tipo de comunicação, são usados quatro pontos de ligação descritas por:

- **MOSI:** Linha de dados usada para enviar informações do mestre para o escravo;
- **MISO**: Linha de dados usada para enviar informações do escravo para o mestre;
- **SCK:** Sinal de clock gerado pelo mestre para sincronizar a comunicação;
- **CS:** Pino usado para habilitar ou desabilitar um dispositivo SPI específico.

A função do micro SD no módulo de recepção é armazenar os arquivos estáticos do servidor, ou seja, todos arquivos responsáveis pela renderização da interface do usuário. Após a realizar as ligações elétricas entre a placa HTIT-WS e o módulo de cartão SD, foi feita a organização do circuito em uma pequena caixa plástica impressa em 3D. Na Figura 12, observa-se as ligações e a montagem final organizada na caixa.

Figura 12 - Montagem do módulo receptor de telemetria



Fonte: Autor

No Quadro 2 é possível visualizar os custos envolvidos no desenvolvimento do projeto de hardware do sistema Air CoPilot (valores sem os impostos de importação), o quadro destaca os valores das peças usadas na montagem dos módulos de controle e visualização bem como as peças usadas na construção do drone de teste usado para validar o sistema.

Quadro 2 - Custos do projeto de hardware do sistema

Peça	Valor	Função
Kit Motores	**R$ 650,00**	**Drone**
Rádio RC	**R$ 1.272,00**	**Drone**
Hélices	**R$ 240,00**	**Drone**
Naza V2	**R$ 840,00**	**Drone**
Frame	**R$ 350,00**	**Drone**
Bateria	**R$ 280,00**	**Drone**
Orange Pi	**R$ 210,00**	**Air CoPilot**
HTIT-WSL	**R$ 76,00**	**Air CoPilot**
LiDAR	**R$ 270,00**	**Air CoPilot**

Fonte: Autor

4.2 Desenvolvimento do software e testes do sistema

Esta seção descreve como foi realizado o desenvolvimento de cada módulo que compõe as partes não físicas do sistema *Air CoPilot*, pontuando as tecnologias usadas.

4.2.1 Firmware do HTIT-WSL que compõe o módulo de processamento

A primeira etapa do desenvolvimento do software teve como objetivo desenvolver o *firmware* da placa HTIT-WSL responsável por coletar a telemetria do drone, ler os comandos do rádio controle e devolver os comandos para a controladora de voo. Além disso, ela serve como ponte de comunicação entre o drone e o sistema *Air CoPilot*. O código foi desenvolvido utilizando-se a IDE de programação *Visual Studio* da Microsoft. Também foi usado o framework *PlatformIO*. Todo o código foi escrito na linguagem de programação C++, que é a linguagem padrão usada pelo *framework*, e o firmware foi estruturado da seguinte forma:

- **Funções de leitura dos dados de telemetria:** responsáveis por ler os dados do drone e organizá-los em structs (estruturas de dados do C++);
- **Protocolo e comunicação UART:** foi escrito um simples protocolo de comunicação para trafegar dados pela interface UART entre a HTIT-WSL e o Orange Pi, além da codificação das funções de leitura e escrita dos dados;
- **Funções de transmissão e recepção:** foram implementadas funções para enviar e receber dados usando a transmissão sem fio de longa distância LoRa. Essas funções são usadas para enviar a telemetria para o módulo de visualização;
- **Funções de leitura e escrita dos comandos de rádio:** foram implementadas funções para receber os comandos enviados pelo rádio controle do drone e também escrever os comandos para a controladora de voo;.

Um trecho do código responsável por coletar os dados da telemetria e organizá-los no struct *aiData* pode ser observado na Figura 13.

Figura 13 - Trecho de código que coleta os dados de telemetria

```
aiData getDataTelemetry()
{
  aiData data;
  if (getHomePoint)
  {
    if (dataGPS.fixStatus == 3)
    {
      data.aircraftHomeDistance = calculateDistance(latHome, lonHome,
dataGPS.latitude, dataGPS.longitude);
      sendDebug(String(data.aircraftHomeDistance));
    }
    else
    {
      data.aircraftHomeDistance = 0;
    }
  }
  else
  {
    data.aircraftHomeDistance = 0;
  }
  data.aircraftBatteryLevel = RC_INTERFACE.getBatteryLevel();
  data.aircraftSignalRCLevel = RC_INTERFACE.RSSI();
  data.aircraftGPSSats = dataGPS.numSats;
  data.aircraftgpsFIX = dataGPS.fixStatus;
  data.aircraftFlightMode = map(RC_INTERFACE.getChannelValue(5),
RC_CHANNEL_MIN, RC_CHANNEL_MAX, 0, 255);
  data.aircraftRCCH1 = map(RC_INTERFACE.getChannelValue(1), RC_CHANNEL_MIN,
RC_CHANNEL_MAX, 0, 255); // Canal RC 1 de 0 a 255
  data.aircraftRCCH2 = map(RC_INTERFACE.getChannelValue(2), RC_CHANNEL_MIN,
RC_CHANNEL_MAX, 0, 255); // Canal RC 2 de 0 a 255
  data.aircraftRCCH3 = map(RC_INTERFACE.getChannelValue(3), RC_CHANNEL_MIN,
RC_CHANNEL_MAX, 0, 255); // Canal RC 3 de 0 a 255
  data.aircraftRCCH4 = map(RC_INTERFACE.getChannelValue(4), RC_CHANNEL_MIN,
RC_CHANNEL_MAX, 0, 255); // Canal RC 4 de 0 a 255
  return data;
}
```

Fonte: Autor

Após a leitura e estruturação dos dados de telemetria, o struct é enviado via UART para processamento no microcomputador Orange Pi Zero 2W. Porém, antes de realizar o envio, é preciso montar a mensagem seguindo o protocolo de comunicação que foi desenvolvido para a troca de mensagens entre as placas.esse protocolo, os bytes 0x55 e 0x50 indicam o começo da mensagem, enquanto que os bytes 0x41, 0x2E e 0x49 indicam o final do mensagem. Entre esses dois conjuntos de bytes são distribuídas as informações do struct. Essa técnica de organizar a mensagem é importante para garantir a integridade dos dados enviados, evitando o

envio de pacotes corrompidos. Por fim, um novo pacote é criado contendo todo o conteúdo do struct e também os bytes fixos que indicam o início e fim. Na Figura 14 é possível observar o corpo da função responsável por enviar o pacote.

Figura 14 - Função que envia os dados para o Orange Pi Zero 2W

```
void sendDataForAIProcessing(aiData dataForProcessing)
{
  uint8_t pack[80];
  int index = 0;
  pack[index++] = 0x55;
  pack[index++] = 0x50;
  uint8_t byte1Dist = *((uint8_t *)&dataForProcessing.aircraftHomeDistance);
  uint8_t byte2Dist = *((uint8_t *)&dataForProcessing.aircraftHomeDistance + 1);
  uint8_t byte3Dist = *((uint8_t *)&dataForProcessing.aircraftHomeDistance + 2);
  uint8_t byte4Dist = *((uint8_t *)&dataForProcessing.aircraftHomeDistance + 3);
  uint8_t byte1Bat = *((uint8_t *)&dataForProcessing.aircraftBatteryLevel);
  uint8_t byte2Bat = *((uint8_t *)&dataForProcessing.aircraftBatteryLevel + 1);
  uint8_t byte3Bat = *((uint8_t *)&dataForProcessing.aircraftBatteryLevel + 2);
  uint8_t byte4Bat = *((uint8_t *)&dataForProcessing.aircraftBatteryLevel + 3);
  // Armazena os 4 bytes no pacote com deslocamento
  pack[index++] = byte1Dist;
  pack[index++] = byte2Dist;
  pack[index++] = byte3Dist;
  pack[index++] = byte4Dist;
  pack[index++] = byte1Bat;
  pack[index++] = byte2Bat;
  pack[index++] = byte3Bat;
  pack[index++] = byte4Bat;
  pack[index++] = dataForProcessing.aircraftSignalRCLevel;
  pack[index++] = dataForProcessing.aircraftGPSSats;
  pack[index++] = dataForProcessing.aircraftgpsFIX;
  pack[index++] = dataForProcessing.aircraftFlightMode;
  pack[index++] = dataForProcessing.aircraftRCCH1;
  pack[index++] = dataForProcessing.aircraftRCCH2;
  pack[index++] = dataForProcessing.aircraftRCCH3;
  pack[index++] = dataForProcessing.aircraftRCCH4;
  pack[index++] = 0x41;
  pack[index++] = 0x2e;
  pack[index++] = 0x49;
  Serial2.write(pack, index);
}
```

Fonte: Autor

Em paralelo a isso, uma tarefa (função assíncrona) preenche um struct do tipo *dataTelemetry* para ser enviado via LoRa. Ele é usado para enviar dados do

drone, instruções e status do módulo de processamento para o módulo de visualização. Essa estrutura possui quase os mesmos dados de telemetria da *aiData.* Porém, ela contém dados processados pelo Orange Pi. O tamanho final desse struct em bytes não pode ser muito grande, pois a proposta da transmissão sem fio LoRa é enviar dados para longas distâncias e não enviar grandes volumes de dados. Se essa regra for quebrada, a velocidade de transmissão é afetada, podendo até ocorrer a perda de bytes durante a transferência. Na Figura 15 é possível observar uma parte do código da tarefa responsável por preparar e enviar a estrutura *dataTelemetry.*

Figura 15 - Trecho de código que envia dados para o módulo de visualização

```
uint8_t data[sizeof(dataAircraftTelemetry) + 4];
memset(data, 0xFF, sizeof(data));
data[0] = 0x55;
data[1] = 0x50;
memcpy(&data[2], &dataAircraftTelemetry, sizeof(dataAircraftTelemetry));
data[sizeof(dataAircraftTelemetry) - 2] = 0x41;
data[sizeof(dataAircraftTelemetry) - 1] = 0x49;
AI_Lora.sendpackage(data, sizeof(data));
delay(500);
```

Fonte: Autor

Após processar os dados o módulo de processamento Orange Pi Zero 2W envia um pacote para a placa HTIT-WSL contendo diversas informações processadas como o status gerado pela rede neural e as instruções para o piloto. Além disso, também é enviada uma lista composta por todos os obstáculos detectados pelo sensor LiDAR. Para que o piloto possa avaliar o estado de funcionamento do microcomputador, os dados de uso da CPU, temperatura e uso de memória RAM também são enviados.

Por fim, a placa HTIT-WSL lê os comandos do receptor de rádio do drone. Um sinal SBUS gerado pelo receptor contém a informação de canal de controle do drone, os principais são os canais de aceleração e direção responsáveis por controlar a altitude da aeronave e realizar manobras. A placa lê os comandos e gera um novo sinal SBUS com os mesmos comandos espelhados. Dessa forma, a

controladora de voo recebe não os dados do receptor, mas uma cópia deles. Essa técnica permite que comandos sejam manipulados pelo sistema, e também permite a criação de ações personalizadas como, por exemplo, a decolagem automática. Nesse caso, seria o sistema *Air CoPilot* gerando instruções no sinal SBUS com comandos que fazem o drone decolar. Esse é só um exemplo das inúmeras possibilidades de ações que esse sistema pode executar no drone.

A leitura e geração do sinal SBUS é feita repetitivamente por duas tarefas. A diferença entre essas tarefas e as outras descritas anteriormente é a prioridade mais elevada. Isso faz com que o ESP32 mantenha as tarefas em execução independentemente de qualquer fator. Na Figura 16 é apresentada a construção das tarefas *SBUS_RC_EXTERNTask* e *SBUS_RC_INTERNALTask,* respectivamente. Uma delas é responsável por ler, enquanto que a outra é responsável por espelhar o sinal SBUS. Como a tarefa de espelhamento é a que envia de fato os comandos para a controladora de voo, ela possui a maior prioridade entre as tarefas do sistema. Isso acontece porque ela é uma tarefa crítica, ou seja, a sua falha poderá pôr o voo em risco.

Figura 16 - Implementação das tarefas do sinal SBUS

```
xTaskCreate(
        SBUS_RC_EXTERNTask, // Função da task
        "SBUS EXTERN Task", // Nome da task
        10000,              // Tamanho da pilha (em bytes)
        NULL,               // Parâmetro para a task
        2,                  // Prioridade da task
        NULL                // Handle da task
    );

xTaskCreate(
        SBUS_RC_INTERNALTask, // Função da task
        "SBUS INTERNAL Task", // Nome da task
        10000,                // Tamanho da pilha (em bytes)
        NULL,                 // Parâmetro para a task
        3,                    // Prioridade da task
        NULL                  // Handle da task
    );
```

Fonte: Autor

4.2.2 Desenvolvimento do módulo de processamento

O primeiro passo no processamento dos dados de telemetria coletados e enviados pela placa HTIT-WSL é realizar a leitura dos pacotes recebidos. Para isso, foi usada a biblioteca de comunicação serial *pySerial.* Ela implementa o protocolo UART e permite estabelecer comunicação entre as duas placas. Os dados são lidos identificando os bytes de assinatura enviados. Quando os bytes que indicam o início do pacote são recebidos, inicia-se o processo de receber os demais dados até que os bytes do final do pacote cheguem. Isso garante que a mensagem seja recebida por completo. Figura 17 é possível observar um trecho do código encarregado de abrir uma conexão e fazer a leitura dos pacotes, que são posteriormente enviados para a função de decodificação responsável por separar os dados de telemetria do pacote para processamento.

Figura 17 - Código Python responsável por receber os pacotes de telemetria

```python
def run(self):
        serial_interface = serial.Serial(self.serial_port, 115200, timeout=1)
        time.sleep(2)  # Espera para estabilizar a comunicação
        buffer = b""
        while True:
            if serial_interface.in_waiting > 0:
                buffer += serial_interface.read(serial_interface.in_waiting)
                # Processa os pacotes se encontrar start e end sequence
            while self.start_sequence in buffer and self.end_sequence in buffer:
                start_idx = buffer.index(self.start_sequence)
                end_idx = buffer.index(self.end_sequence) + len(self.end_sequence)
                data_packet = buffer[start_idx:end_idx]
                self.package_decode(serial_interface, data_packet)
                buffer = buffer[end_idx:]
```

Fonte: Autor

Para fazer com que o sistema *Air CoPilot* interprete os dados da telemetria da mesma forma que um piloto, foi necessário implementar e treinar uma rede neural artificial com a habilidade de ler os dados e devolver uma avaliação do estado atual do drone. Para isso, foi usada a biblioteca Python *sklearn,* que simplifica várias etapas da implementação, pois ela é basicamente uma interface da *TensorFlow,* ou seja, a sklearn usa a biblioteca TensorFlow internamente para fazer as

implementações de aprendizagem de máquina. A rede neural construída para o projeto é do modelo *feedforward*. Isso significa que os dados que entram na rede fluem em uma única direção para a saída, sem passar por ciclos.

A estrutura da rede desenvolvida é composta por quatro camadas densas totalmente conectadas entre si. Essas camadas são compostas da seguinte forma:

- **Primeira camada:** é formada por 200 neurônios, sendo responsável pela entrada dos dados. Essa camada trabalha com a função de ativação *ReLU* (Rectified Linear Unit), que ajuda a rede neural a aprender padrões;
- **Segunda camada:** é formada por 100 neurônios, também com a função de ativação ReLu. Porém, nela foi usada a função *Dropout(0.3),* que desativa aleatoriamente 30% dos neurônios durante o treinamento da rede. O uso dessa função evita que ocorra sobreajuste do modelo, que nada mais é do que a rede neural aprendendo demais os dados de treinamento e dificultando as previsões com dados diferentes dos usados no processo de treinamento;
- **Terceira camada:** é formada por 50 neurônios também com a função de ativação ReLU. Porém, nela foi usada a técnica de regularização L2 para evitar sobreajuste. Essa técnica consiste em balancear os pesos, o que gera modelos mais robustos;
- **Quarta camada:** é a saída dos dados, ou seja, a previsão feita pela rede. Ela possui apenas 6 neurônios, que indicam que o modelo está classificando os dados em 6 classes diferentes. Nela foi usada a função de ativação *softmax,* que converte os valores da saída em distribuições de probabilidade entre cada classe. Logo, a classe com maior probabilidade é escolhida como valor de saída da rede;

A Figura 18 mostra o trecho de código responsável por criar a rede neural descrita acima. Nela, é possível identificar facilmente a implementação de cada camada da rede neural, bem como os seus parâmetros como, por exemplo, a função de ativação.

Figura 18 - Construção da rede neural

```
# Construir a rede neural
model = Sequential([
    Dense(200, input_dim=X_train.shape[1], activation='relu',
kernel_regularizer=l2(0.001)),
    Dropout(0.3),
    Dense(100, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.3),
    Dense(50, activation='relu', kernel_regularizer=l2(0.001)),
    Dense(6, activation='softmax')  # Camada de saída para classificação multi-
classe
])
```

Fonte: Autor

Após a definição da rede neural foi necessário compilar o modelo para que fosse possível fazer o seu treinamento. O processo de compilação envolveu a definição do otimizador, a função de perda e das métricas que o modelo usaria durante o treinamento. No caso da rede do sistema *Air CoPilot* foi usado o otimizador Adam, que é responsável por ajustar os pesos da rede neural durante o treinamento. A função de perda Loss foi usada para medir a diferença entre as previsões do modelo e os valores reais esperados. A Figura 19 apresenta o trecho de código responsável por compilar a rede neural.

Figura 19 - Compilando a rede neural

```
# Compilar o modelo
optimizer = Adam(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Configurar EarlyStopping para evitar overfitting
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
```

Fonte: Autor

Por fim, a rede neural foi treinada para reconhecer situações adversas que podem ocorrer com o drone. Treinar a rede significa fazer com que o modelo aprenda a realizar tarefas com base em dados conhecidos, de forma semelhante a uma criança que aprende algo novo todos os dias. Para o treinamento inicial da rede neural foi utilizado um conjunto de dados contendo 1000 exemplos de dados de telemetria, organizados em formato de tabela. Cada linha continha um conjunto de dados com uma ação previamente definida pelo piloto. A formulação desse primeiro conjunto de dados contou com a colaboração de vários pilotos de drones, que forneceram dados de telemetria junto às respectivas ações tomadas em cada situação. A tabela do conjunto de dados é composta por cinco colunas, que representam os seguintes dados:

- **Distância:** é um valor em metros que indica a distância entre o drone e o ponto de decolagem;
- **Bateria:** indica o nível restante de energia do drone, com um valor que varia de 0 a 100%;
- **Satélites:** indica o número atual de satélites conectados ao sistema de GPS do drone. Esse valor indicar o nível de confiança dos dados gerados pelo GPS e, quanto maior o seu valor, mais preciso os dados são;
- **Sinal:** é um valor que varia de 0 a 100% e informa a qualidade da comunicação entre o sistema de controle remoto e o drone;
- **Status**: é uma classificação feita com base na análise das colunas anteriores. Ela significa a ação que o piloto tomaria. Cada ação possui um valor que varia de 0 a 5, representando, respectivamente, os seguintes status:“Normal”, “Retornar ao ponto de origem”, “Melhorar o sinal”, “Pouso de emergência”, “Corrigir a rota para evitar obstáculos” e “Modo manual”;

Na Figura 20 é possível observar dez amostras dos dados retirados do conjunto de dados utilizado para o primeiro treinamento da rede neural. Também é possível identificar as cinco colunas com os seus respectivos valores. Analisando a segunda linha de dados, observa-se que a distância entre o drone e o ponto de decolagem é de 1139 metros, o nível da bateria é de 33%, há sete satélites conectados e a intensidade do sinal é de apenas 44%. Diante desses fatores, a ação tomada foi "2", que corresponde a um pouso de emergência.

Figura 20 - Parte do conjuntos de dados primário

1	Distância	Bateria	Satélites	Sinal	Status
2	533	58	4	72	5
3	1139	33	7	44	2
4	439	90	8	100	0
5	680	66	5	79	5
6	907	65	5	71	5
7	1405	34	8	39	3
8	55	83	12	86	0
9	530	51	7	63	2
10	143	99	9	91	0

Fonte: Autor

Para chegar a essa decisão, o piloto analisou a distância do drone (neste caso 1139 metros), que pode ser considerada relativamente distante. Outro fator crucial foi o nível da bateria, de apenas 33%, tornando arriscada a tentativa de retorno, pois haveria risco de queda por falta de energia. Além disso, a baixa intensidade do sinal poderia resultar em desconexão e perda de controle. Com base nesses dados, a melhor escolha foi realizar o pouso de emergência. Seguindo essa mesma lógica, as outras 999 linhas do conjunto de dados foram preenchidas com valores e respectivos status e, após essa etapa, o modelo foi treinado.

No código apresentado na Figura 21, os dados do conjunto de dados são extraídos da tabela CSV, separados e normalizados. A chamada para a função

model.fit realiza o treinamento com um valor de 100 *epochs*, o que significa que o modelo será alimentado 100 vezes com os dados do conjunto de dados. O treinamento é encerrado quando a eficiência máxima é atingida e, no final, a acurácia do modelo é impressa no log do script. A acurácia indica o quão precisa a rede neural se tornou após o treino. No primeiro treinamento, utilizando os 1000 exemplos do dataset, foi possível obter uma acurácia de 75%. Embora esse seja um bom resultado, ainda estava abaixo do necessário para um sistema mais assertivo. Para aumentar a precisão, foi necessário ampliar o tamanho do conjunto de dados.

Figura 21 - Trecho de código responsável por treinar a rede neural

```
# Carregar os dados do CSV
df = pd.read_csv('neuralNetwork/dataSet_filtered.csv')
# Separar as colunas de entrada (X) e saída (y)
X = df[['Distância', 'Bateria', 'Satélites', 'Sinal']]
y = df['Status']
# Normalizar os dados e converter para float32 (compatível com TFLite)
scaler = StandardScaler()
X_normalized = scaler.fit_transform(X).astype(np.float32)
# Dividir os dados em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X_normalized, y,
test_size=0.2, random_state=42, stratify=y)
# Treinar o modelo
history = model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=
(X_test, y_test), callbacks=[early_stopping])
# Avaliar o modelo no conjunto de teste
loss, accuracy = model.evaluate(X_test, y_test)
print(f'Precisão no conjunto de teste: {accuracy * 100:.2f}%')
```

Fonte: Autor

Para isso, foi desenvolvido um *script* simples que gerava dados de telemetria de forma aleatória e os enviava para classificação na rede neural treinada anteriormente. Após a geração era realizada uma revisão manual dos status previstos pela rede, garantindo a qualidade dos dados classificados. Esse processo foi repetido várias vezes até que a tabela de exemplos alcançasse 8000 linhas de dados. Por fim, o modelo foi novamente treinado com o novo conjunto de dados, atingindo uma acurácia de 95%, tornando o sistema significativamente mais acurado. Para um bom desempenho no Orange Pi Zero 2W, o modelo recém treinado foi convertido para o formato *tflite*, uma versão mais leve do TensorFlow

que é excelente para executar em sistemas embarcados com recursos computacionais limitados.

A última etapa de processamento do sistema *Air CoPilot* é a realização da leitura e formação da nuvem de pontos capturados pelo sensor LiDAR. Para isso, o sensor envia dados de forma constante que são recebidos pelo script Python por meio da comunicação serial. Esses dados consistem na leitura de distância e ângulo e, para cada grau dos 360 graus que compõem uma volta completa, o sensor realiza duas capturas para cada grau. Assim, no final de uma volta completa, a nuvem de pontos gerada é de 720 capturas.

A cada segundo, o LiDAR dá 6 voltas, o que faz com que esse processo gere uma grande quantidade de dados. Para identificar, em meio a tantos dados, a presença de obstáculos, foi usado o algoritmo de machine learning DBSCAN. A função do DBSCAN nesse caso de uso é receber a nuvem de pontos e devolver uma lista de clusters. Na Figura 22 é possível observar o corpo da função *generateClusters,* que recebe como parâmetro uma lista de pontos que representam a nuvem gerada pelo sensor. Porém, antes de realizar a clusterização dos dados, é feita uma conversão entre coordenadas polares (ângulo e distância) em coordenadas cartesianas(X e Y). Essa conversão é feita para que o algoritmo processe os dados de forma mais eficiente. Depois dessa primeira etapa é feita a chamada do DBSCAN, que é implementado por meio da biblioteca Python sklearn. Na primeira chamada são configurados o *eps* (raio mínimo entre os pontos) e o *min_samples* (quantidade mínima de pontos para ser considerado um cluster) e então o método *fit_predict* realiza de fato a clusterização dos dados que são organizados em um dicionário e retornados

Figura 22 - Aplicando o algoritmo DBSCAN

```python
def generateClusters(self, lidar_data):
        # Verifica se os dados estão vazios
        if not lidar_data:
            return {}

        # Converte os dados polares (ângulo, distância) para cartesianos
(x, y)
        obstacle_points = [
            (distance * np.cos(np.radians(angle)), distance *
np.sin(np.radians(angle)))
            for angle, distance in lidar_data
        ]

        # Verifica se há pontos de obstáculos
        if not obstacle_points:
            return {}

        # Converte para numpy array
        obstacle_points = np.array(obstacle_points)

        # Aplica o algoritmo DBSCAN para identificar clusters
        db = DBSCAN(eps=0.1, min_samples=10)
        labels = db.fit_predict(obstacle_points)

        # Calcula o centro de cada cluster
        clusters = []
        for label in set(labels):
            if label == -1:
                continue  # Ignora ruídos
            cluster_points = obstacle_points[labels == label]
            cluster_center = cluster_points.mean(axis=0)
            clusters.append({
                "center": cluster_center.tolist()
            })
        return clusters
```

Fonte: Autor

Por fim, os clusters mais próximos do drone são enviados para a placa HTIT-WSL, que os organiza e os envia para o módulo de visualização para que o piloto possa realizar as manobras de desvio. Quando a placa HTIT-WSL recebe algum cluster potencialmente próximo de colidir com o drone, ela gera um comando

no sinal SBUS que mantém a aeronave imóvel, evitando, assim, a colisão. O controle do drone só é devolvido após o piloto aceitar o aviso de obstáculo detectado.

4.2.3 Desenvolvimento do módulo de visualização

A visualização dos dados processados e da telemetria de voo do drone é um fator essencial para que o piloto consiga usar o sistema *Air CoPilot.* Diante dessa necessidade, foi desenvolvido um módulo de visualização universal, que é acessível por diferentes dispositivos, como smartphones, tablets e computadores, sem a necessidade de instalação de aplicativos. A solução adotada foi a utilização de uma interface Web, que permite o acesso por meio de um navegador de internet, garantindo maior compatibilidade e escalabilidade ao sistema.

A interface gráfica recebe todos os dados de telemetria da placa HTIT-WSL que compõem o módulo de visualização. A comunicação ocorre por meio do protocolo *WebSocket*, permitindo uma troca de mensagens contínua e em tempo real. O processamento desses dados é realizado pelo script da interface Web, que interpreta os dados recebidos e os exibe dinamicamente na tela. Na Figura 23 é possível visualizar o trecho de código JavaScript responsável por criar uma nova conexão WebSocket e iniciar o evento de recepção de mensagens. O código recebe os dados em formato JSON e atualiza a DOM (Document Object Model) por meio do acesso *getElementById.*

Figura 23 - Código JavaScript para criar conexão WebSocket

```
const socket = new WebSocket(`ws://${window.location.host}/ws`);
socket.onmessage = function (event) {
        const jsonObject = JSON.parse(event.data);
        if ("fmode" in jsonObject) {
            //console.log(jsonObject["rcSignal"])
            document.getElementById("rcSignal").innerHTML = jsonObject["rcSignal"] + "%"
            document.getElementById("loraSignal").innerHTML = jsonObject["loraSignal"] + "%"
            document.getElementById("fmode").innerHTML = jsonObject["fmode"]
            if(jsonObject["rcSignal"] <= 60){
                piscarElemento("rcSignal")
            }
```

Fonte: Autor

Para servir a interface gráfica e seus arquivos estáticos, foi usado um servidor Web acessível por meio de um ponto de acesso Wi-Fi chamado "*AirCoPilot_Dashboard*", que é gerado pela placa HTIT-WSL. A implementação do servidor foi feita por meio da biblioteca *ESPAsyncWebServer*, uma ferramenta usada na criação de servidores assíncronos no microcontrolador ESP32. Elaa fornece implementações de rotas e possibilita a criação de WebSockets fundamentais para a comunicação entre a placa e a interface do usuário.

Para o devido funcionamento todos os arquivos estáticos referentes à interface foram armazenados em um cartão micro SD, e o cartão conectado no leitor de micro SD que compõe o módulo de visualização descrito na seção 4.1.3 deste documento. Essa técnica possibilita o armazenamento de vários arquivos no sistema sem comprometer a memória interna do microcontrolador. As rotas do servidor assíncrono, ao serem requisitadas pelo navegador, fazem a leitura do conteúdo do cartão SD e entregam a resposta da requisição. Na Figura 24 é apresentado o código da rota *index* e demais rotas que entregam o HTML da interface. Nela, também é apresentada a rota que entrega arquivos estáticos. Ela é usada quando a página HTML index requisita as dependências como, scripts, modelos 3D e sons

Figura 24 - Código de rotas do servidor

```
server.on("/", HTTP_GET, [](AsyncWebServerRequest *request)
            { request->send(SD, "/index.html", "text/html"); });

server.on("/files", HTTP_GET, [](AsyncWebServerRequest *request)
            { request->send_P(200, "text/html", index_html); });
server.on("/list", HTTP_GET, [](AsyncWebServerRequest *request)
            { request->send(200, "application/json", listFiles()); });
server.on("/upload", HTTP_POST, [](AsyncWebServerRequest *request)
            { request->send(200); }, [](AsyncWebServerRequest *request, const String
&filename, size_t index, uint8_t *data, size_t len, bool final)
            {
        static File file;
        if (index == 0) {
            Serial.printf("Upload iniciado: %s\n", filename.c_str());
            file = SD.open("/" + filename, FILE_WRITE);
        }
        if (file) {
            file.write(data, len);
            if (final) {
                file.close();
                Serial.printf("Upload concluído: %s\n", filename.c_str());
            }
        } });
// Rota para servir arquivos estáticos (CSS, JS, etc.) do SD
server.serveStatic("/", SD, "/").setCacheControl("public");
// Inicia o servidor
server.begin();
```

Fonte: Autor

Além da interface da dashboard, foi um implementado um simples gerenciador de arquivos, que serve para fazer upload de arquivos estáticos para o cartão de memória do módulo de visualização, mantendo a interface sempre atualizada. A recepção dos dados de telemetria enviados pelo módulo de controle junto ao drone é feita pelo módulo LoRa acoplado na placa. Os pacotes recebidos são decodificados e estruturados em um documento JSON, que é enviado pelo WebSocket para o script que abriu a conexão. A Figura 25 mostra o corpo da função que recebe um pacote de bytes, o converte em um struct de telemetria, o estrutura e envia o documento JSON.

Figura 25 - Corpo da função que recebe os dados e prepara o JSON

```
void decodePack(uint8_t *data, int size)
{
  if (size != sizeof(dataTelemetry))
  {
    return;
  }
  memcpy(&telemetry, data, sizeof(telemetry));
  StaticJsonDocument<512> json;
  char jsonData[512];
  json["aiStatus"] = telemetry.aiStatus;
  float batteryVoltage = telemetry.bat / 100.0;
  json["bat"] = batteryVoltage;
  json["dist"] = telemetry.dist;
  json["lidarStatus"] = telemetry.lidarStatus;
  json["gpsFix"] = telemetry.GPSFix;
  json["sats"] = telemetry.sats;
  json["speed"] = telemetry.speed;
  json["fmode"] = telemetry.fmode;
  json["rcSignal"] = telemetry.rcSignal;
  json["loraSignal"] = 90;
  json["cpuUsage"] = telemetry.cpuUse;
  json["cpuMem"] = telemetry.cpuMem;
  json["accuracy"] = 96;
  if (telemetry.lidarStatus)
  {
    json["clusterAngle"] = telemetry.clusterAngle;
    json["clusterDistance"] = telemetry.clusterDist;
  }
  float cpuTemp = telemetry.cpuTemp / 100.0;
  json["cpuTemp"] = cpuTemp;
  float altitude = telemetry.alt / 100.0;
  json["alt"] = altitude;
  Serial.printf("Altitude: %f\n", telemetry.alt);
  size_t len = serializeJson(json, jsonData, sizeof(jsonData));
  if (len > 0)
  {
    ws.printfAll(jsonData);
  }
}
```

Fonte: Autor

Na Figura 25 a chamada do método *ws.printAll(jsonData)* envia de fato o JSON para todos os dispositivos conectados na instância ws que, nesse caso, é o objeto que representa o servidor WebSocket gerado. Dessa forma, a interface gráfica sempre vai ter suas informações atualizadas, pois a função *decodePack* é chamada sempre que um novo pacote de telemetria chega.

A interface gráfica foi projetada para apresentar, de forma intuitiva e organizada, todas as informações essenciais do voo, incluindo dados de telemetria, nível de bateria, obstáculos detectados e comandos automatizados. Para isso, foram implementados diversos elementos gráficos que facilitam a interpretação dos

dados e a interação do piloto com o sistema, como, por exemplo, o painel da telemetria que exibe os principais dados do voo, permitindo ao piloto monitorar parâmetros fundamentais para a navegação segura da aeronave. As informações apresentadas incluem:

- **RC:** indica a intensidade do sinal do transmissor de rádio;
- **SPD:** exibe a velocidade atual do drone em tempo real;
- **FM:** indica se o drone está operando no modo GPS, altitude ou manual;
- **Satélites:** informa a quantidade de satélites conectados ao drone;
- **Altitude:** exibe a altura da aeronave em relação ao solo;
- **Home:** apresenta a distância entre a aeronave e o local de decolagem;
- **CPU-U:** apresenta o uso da CPU do microcomputador Orange Pi Zero 2W;
- **CPU-M:** indica a quantidade de memória RAM usada;
- **CPU-T:** informa a temperatura do processador do microcomputador.

Além dessas informações, há também o monitoramento detalhado da carga da bateria, que é essencial para evitar pousos forçados e falhas durante o voo. Para isso, a interface tem um painel específico que exibe a porcentagem atual da bateria e a estimativa de tempo de voo restante. Na Figura 26 é possível observar a interface gráfica com o drone desligado, nesse cenário os dados não são carregados pois não há comunicação com todas as partes do sistema, estão o status mostrado o painel “AI Copilot” é “Desconectado” informando ao piloto a falta de comunicação.

Figura 26 - Interface gráfica do sistema com o drone desligado

Fonte: Autor

Ao se analisar a Figura 26, observa-se outros elementos como o *Radar,* que é uma representação animada do funcionamento do sensor LiDAR pois, quando o sistema recebe os dados, esse elemento tem a sua animação ativada para representar as varreduras e, caso obstáculos sejam detectados, eles serão exibidos como pontos no gráfico do elemento.

O painel denominado *AI Copilot* é exclusivo para a exibição dos estados processados pela rede neural do sistema. Ele é atualizado com essas informações quando o drone decola. Enquanto a aeronave não levantar voo ou estiver desligada, o status a ser exibido é “Não decolado". Nos casos em que o drone foi devidamente ligado mas está em um ambiente crítico, com obstáculos próximos, ou com algum problema técnico, o status a ser exibido é “*Decolagem não autorizada*”. Um painel dinâmico denominado a *Orientação em tempo real* exibe um modelo 3D representando o drone que se move em tempo real conforme os comandos do piloto. Essa simulação foi desenvolvida utilizando a biblioteca gráfica *Three.js*, permitindo visualizar mudanças na inclinação, rotação e altitude do drone.

A interface conta ainda com um gráfico que exibe a variação da altitude do drone ao longo do tempo. Esse gráfico foi desenvolvido com a biblioteca Chart.js, permitindo uma análise mais detalhada da altitude do voo. Com essa funcionalidade, o piloto pode identificar variações inesperadas na altitude e realizar correções caso necessário. A interface termina com um conjunto de botões interativos que permitem ao piloto enviar comandos diretos ao drone. Entre os principais controles implementados, destacam-se:

- **Auto TakeOff:** aciona a decolagem automática do drone;
- **Auto Land:** comando para pouso automatizado;
- **+10M e -10M:** ajustam a altitude do drone em incrementos de 10 metros, permitindo ajustes rápidos durante o voo;
- **Modos de voo (GPS, Altitude e Manual):** permitem alternar entre diferentes configurações de controle da aeronave, possibilitando uma maior flexibilidade na operação.

Na Figura 27 pode-se observar o sistema funcionando com a interface gráfica exibindo os dados de telemetria e o status processado pelo módulo de processamento AI. No teste realizado, o drone foi ligado dentro de casa e, por conta disso, o status gerado foi “Decolagem não autorizada” , pois o sistema de detecção de obstáculos identificou objetos próximos ao drone, o que torna inseguro realizar a decolagem.

Nesse tipo de abordagem o sistema *Air CoPilot* não repassa nenhum comando de acionamento de motores para a controladora de voo, a comunicação será liberada apenas quando o status estiver com o valor “Não decolado”, que indica a que todos os sistemas do drone estão em funcionamento aguardando apenas o comando. Mesmo com a aeronave em modo de espera, se o copiloto artificial identificar alguma coisa errada, como pouca carga na bateria ou sinal fraco, a decolagem será abortada. Após a decolagem, o gráfico da curva de altitude começa a ser atualizado e a exibir as variações de altura. O status deixa de ser “Não decolado” e passa a exibir o status atual do copiloto artificial.

Figura 27 - Interface gráfica do sistema com o drone ligado em solo

Fonte: Autor

Na Figura 27 é apresentada a interface gráfica do sistema Air CoPilot com o drone ligado, nesse cenário o módulo de processamento está recebendo os dados de telemetria do drone e processando na rede neural, ao mesmo tempo está lendo o fluxo de dados do sensor LiDAR e gerando clusters com o DBSCAN e no final do ciclo enviado as informações processadas para a placa HTIT-WS que organiza e envia os dados para a interface gráfica. Como o drone foi ligado em um local com obstáculos próximos, o copiloto artificial bloqueia a decolagem e exibe o status “Decolagem não autorizada”, esse status só é definido graças às leituras realizadas pelo LiDAR que detectou a proximidade com os objetos.

Por fim o último cenário mostrado na Figura 28 é o drone em pleno voo, a interface é atualizada periodicamente para manter o dados atualizados, o status da rede neural é atualizado no painel “AI Copilot”, sempre que algo crítico acontecer as bordas da tela piscam em vermelho para chamar a atenção do piloto além do alerta visual é lançado na interface, efeitos sonoros também são reproduzidos. Os principais alertas ocorrem quando um status diferente de “Normal” é previsto pela

rede neural, também quando o nível de bateria está abaixo de 40%, ou quando algum dado do painel de telemetria cai o seu valor para abaixo do normal, como por exemplo o número de satélites abaixo de seis que torna crítico voar em modo GPS ou quando a intensidade de sinal é menor de 60%.

Figura 28 - Interface gráfica do sistema com o drone em voo

Fonte: Autor

5. CONCLUSÃO

Diante de um cenário em que os drones comerciais e de construção caseira disputam o mesmo espaço aéreo, o risco de acidentes envolvendo esses dispositivos se torna um problema cada vez mais recorrente. Ao contrário dos drones comerciais, as aeronaves de construção caseira possuem poucos ou nenhum recurso que possa ser usado para auxiliar o piloto e, assim, evitar acidentes. A falta desses recursos deixa o total controle do drone a cargo do piloto, que pode facilmente dar um comando errado e colidir o drone com um obstáculo ou, em um pior caso, com uma pessoa.

Com o objetivo de contornar esses problemas, foi proposto o desenvolvimento do Air CoPilot, um sistema de hardware/software capaz de identificar falhas e alertar o piloto a tempo. O sistema opera juntamente com o drone durante o voo, processando todos os comandos enviados pelo piloto e os dados da aeronave, a fim de prestar assistência ao piloto.

O sistema foi desenvolvido seguindo todos os requisitos funcionais propostos, tornando possível criar um copiloto artificial que reúne hardware e software em um único sistema, cumprindo o objetivo geral de fornecer assistência de voo aos pilotos de drones.

Para o futuro do projeto, novas implementações e melhorias poderão ser feitas para tornar o sistema mais robusto e eficiente, como a otimização do algoritmo Python para processar instruções, assumir o controle total da aeronave e resolver problemas; expandir o projeto para outras controladoras de voo além da DJI Naza M-V2; e implementar recursos mais avançados de otimização da rede neural, como o autoaprendizado do modelo.

REFERÊNCIAS

AHMED, Mohiuddin; SERAJ, Raihan; ISLAM, Syed Mohammed Shamsul. The k-means algorithm: A comprehensive survey and performance evaluation. Electronics, v. 9, n. 8, p. 1295, 2020. Disponível em: https://doi.org/10.3390/electronics9081295. Acesso em: 13 jan 2025.

ARNOLD, Taylor B. kerasR: R interface to the Keras deep learning library. Journal of Open Source Software, v. 3, n. 22, p. 296, 2018. DOI: https://doi.org/10.21105/joss.00296. Acesso em: 22 jan 2025.

ESPRESSIF SYSTEMS. Arduino-ESP32: Framework for Arduino on ESP32. 2021. Disponível em: https://docs.espressif.com/projects/arduino-esp32/en/latest/. Acesso em: 8 jan 2024.

HARRIS, Charles R. et al. Array programming with NumPy. Nature, v. 585, p. 357-362, 2020. Disponível em: https://doi.org/10.1038/s41586-020-2649-2. Acesso em: 11 jan 2025.

HOKUYO AUTOMATIC CO. LiDAR Sensor Specifications. Disponível em: https://www.hokuyo-aut.jp. Acesso em: 20 dez 2024.

HUNTER, John D. Matplotlib: A 2D graphics environment. 2007. Disponível em: https://matplotlib.org/. Acesso em: 11 jan 2025.

MAHADEVAN, Prem. The Military Utility of Drones. CSS Analyses in Security Policy, v. 78, n. 2296–0244, p. 1–3, 2010. Disponível em: https://doi.org/10.3929/ethz-a-006253833. Acesso em: 5 maio 2022.

MUSGRAVE, Kevin; BELONGIE, Serge; LIM, Ser-Nam. PyTorch Metric Learning. arXiv, 2020. DOI: https://doi.org/10.48550/arXiv.2008.09164. Acesso em: 24 jan 2025.

O'SHEA, Keiron; NASH, Ryan. An introduction to convolutional neural networks. arXiv, 2015. DOI: https://doi.org/10.48550/arXiv.1511.08458. Acesso em: 3 fev 2025.

PANG, Bo; NIJKAMP, Erik; WU, Ying Nian. Deep learning with TensorFlow: A review. Educational and Psychological Measurement, v. 45, n. 2, p. 1-12, 2019. Disponível em: https://doi.org/10.3102/107699861987276. Acesso em: 22 jan 2025.

PASCANU, Razvan; GULCEHRE, Caglar; CHO, Kyunghyun; BENGIO, Yoshua. How to construct deep recurrent neural networks. arXiv, 2013. DOI: https://doi.org/10.48550/arXiv.1312.6026. Acesso em: 3 fev 2025.

REBACK, Jeff; McKINNEY, Wes; JBROCKMENDEL et al. Pandas 1.0.5. Zenodo, 17 jun. 2020. Disponível em: https://doi.org/10.5281/zenodo.3898987. Acesso em: 11 jan 2025.

RUDER, Sebastian. An overview of gradient descent optimization algorithms. 2016. Disponível em: https://arxiv.org/abs/1609.04747. Acesso em: 1 jan 2025.

SCHUBERT, Erich; SANDER, Jörg; ESTER, Martin; KRIEGEL, Hans Peter; XU, Xiaowei. DBSCAN revisited, revisited: why and how you should (still) use DBSCAN. ACM Transactions on Database Systems, v. 42, n. 3, p. 1-21, 2017. Disponível em: https://doi.org/10.1145/3068335. Acesso em: 13 jan 2025.

SICK AG. Technical Overview of TIM LiDAR Sensors. Disponível em: https://www.sick.com. Acesso em: 20 dez 2024.

SLAMTEC. RPLiDAR A1 Datasheet. Disponível em: https://www.slamtec.com. Acesso em: 18 dez 2024.

TAI, H.; MAS, J. Multilayer Perceptron (MLP). In: CAMACHO OLMEDO, M.; PAEGELOW, M.; MAS, J. F.; ESCOBAR, F. (Eds.). Geomatic Approaches for Modeling Land Change Scenarios. Cham: Springer, 2018. p. 451-455. Disponível em: https://doi.org/10.1007/978-3-319-60801-3_27. Acesso em: 6 jan 2025.

VAN ROSSUM, Guido; DRAKE, Fred L. Jr. The Python Language Reference. Release 2.6.4. Python Software Foundation, 2010. Disponível em: https://www.cse.unr.edu/~sushil/class/381/notes/python/docs-pdf/reference.pdf. Acesso em: 8 jan 2025.

WASKOM, Michael L. Seaborn: Statistical data visualization. Journal of Open Source Software, v. 4, n. 40, p. 3021, 2019. Disponível em: https://doi.org/10.21105/joss.03021. Acesso em: 12 jan 2025.

WILD, G.; MURRAY, J.; BAXTER, G. Exploring Civil Drone Accidents and Incidents to Help Prevent Potential Air Disasters. Aerospace, v. 3, n. 3, p. 22, 2016. Disponível em: https://doi.org/10.3390/aerospace3030022. Acesso em: 1 jun. 2022.

ZHANG, G.P. Neural networks for classification: a survey. IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), v. 32, n. 4, p. 451-462, 2002. DOI: https://doi.org/10.1109/5326.897072. Acesso em: 4 fev 2025.

GLOSSÁRIO

AI (Artificial Intelligence): Inteligência Artificial. Campo da computação que busca desenvolver sistemas capazes de realizar tarefas que normalmente exigiriam inteligência humana, como aprendizado, tomada de decisão e reconhecimento de padrões.

ANAC (Agência Nacional de Aviação Civil): Órgão regulador da aviação civil no Brasil, responsável por estabelecer normas e fiscalizar o uso de aeronaves, incluindo drones.

ANATEL (Agência Nacional de Telecomunicações): Agência responsável pela regulamentação e fiscalização dos serviços de telecomunicações no Brasil, incluindo a homologação de equipamentos de radiofrequência, como os utilizados em drones.

API (Application Programming Interface): Interface de Programação de Aplicações. Conjunto de regras que permite que softwares diferentes se comuniquem entre si.

CPU (Central Processing Unit): Unidade Central de Processamento. Componente responsável por executar instruções de software e realizar cálculos em um sistema computacional.

CSV (Comma-Separated Values): Formato de arquivo utilizado para armazenar dados tabulares, onde os valores são separados por vírgulas.

DBSCAN (Density-Based Spatial Clustering of Applications with Noise): Algoritmo de agrupamento baseado na densidade de pontos, utilizado para identificar clusters e ruídos em conjuntos de dados.

DJI (Da-Jiang Innovations): Empresa chinesa líder na fabricação de drones para uso recreativo e profissional.

ESP (Espressif Systems): Empresa especializada na fabricação de microcontroladores e módulos de comunicação sem fio, como os chips ESP32 e ESP8266.

GPIO (General Purpose Input/Output): Pinos de entrada e saída de propósito geral, utilizados em microcontroladores para interagir com sensores e atuadores.

GPS (Global Positioning System): Sistema de Posicionamento Global. Tecnologia utilizada para determinar a localização de um dispositivo na Terra por meio de satélites.

GUI (Graphical User Interface): Interface Gráfica do Usuário. Elementos visuais que permitem a interação entre o usuário e o sistema.

HTML (HyperText Markup Language): Linguagem de marcação utilizada para estruturar páginas da web.

HTTP (Hypertext Transfer Protocol): Protocolo de comunicação utilizado para transferir dados na internet.

I2C (Inter-Integrated Circuit): Protocolo de comunicação utilizado para conectar dispositivos eletrônicos, como sensores e microcontroladores.

IDE (Integrated Development Environment): Ambiente de Desenvolvimento Integrado. Software que fornece ferramentas para a programação, como editores de código e depuradores.

JSON (JavaScript Object Notation): Formato de intercâmbio de dados leve e estruturado, utilizado principalmente em aplicações web.

LiDAR (Light Detection and Ranging): Tecnologia que usa feixes de laser para medir distâncias e criar mapas tridimensionais do ambiente.

LoRa (Long Range): Tecnologia de comunicação sem fio de longo alcance e baixo consumo de energia, utilizada em redes IoT (Internet das Coisas).

ML (Machine Learning): Aprendizado de Máquina. Subcampo da Inteligência Artificial que desenvolve algoritmos capazes de aprender padrões a partir de dados.

RC (Remote Control): Controle Remoto. Dispositivo utilizado para enviar comandos para drones e outros equipamentos sem fio.

RSSI (Received Signal Strength Indicator): Indicador de Intensidade do Sinal Recebido. Mede a potência do sinal de rádio recebido por um dispositivo.

SISANT (Sistema de Aeronaves Não Tripuladas): Plataforma da ANAC para registro de drones no Brasil.

TCC (Trabalho de Conclusão de Curso): Projeto acadêmico desenvolvido como requisito para a conclusão de um curso superior.

UART (Universal Asynchronous Receiver/Transmitter): Protocolo de comunicação serial assíncrona utilizado para transmitir dados entre dispositivos eletrônicos.

INSTITUTO FEDERAL DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA DA PARAÍBA
Campus Cajazeiras - Código INEP: 25008978
Rua José Antônio da Silva, 300, Jardim Oásis, CEP 58.900-000, Cajazeiras (PB)
CNPJ: 10.783.898/0005-07 - Telefone: (83) 3532-4100

Documento Digitalizado Ostensivo (Público)

TCC

Assunto:	TCC
Assinado por:	Klinsman Jorge
Tipo do Documento:	Relatório
Situação:	Finalizado
Nível de Acesso:	Ostensivo (Público)
Tipo do Conferência:	Cópia Simples

Documento assinado eletronicamente por:

- **Jose Klinsman Jorge Silva**, **ALUNO (202012010032) DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS - CAJAZEIRAS**, em 25/03/2025 16:59:35.

Este documento foi armazenado no SUAP em 25/03/2025. Para comprovar sua integridade, faça a leitura do QRCode ao lado ou acesse https://suap.ifpb.edu.br/verificar-documento-externo/ e forneça os dados abaixo:

Código Verificador: 1434131
Código de Autenticação: b52d8f39ac