VINÍCIUS MAURÍCIO DE ALMEIDA

Framework Modular para Detecção e Desvio de Objetos em Veículos Aéreos não Tripulados

São Paulo

2023

VINÍCIUS MAURÍCIO DE ALMEIDA

Framework Modular para Detecção e Desvio de Objetos em Veículos Aéreos não Tripulados

Versão Corrigida

Dissertação apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do título de Mestre em Ciências

Orientador: Prof. Dr. Paulo Sérgio Cugnasca

São Paulo

2023

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catalogação-na-publicação

almeida, vinicius
Framework Modular para Detecção e Desvio de Objetos em Veículos Aéreos não Tripulados / V. almeida -- versão corr. -- São Paulo, 2023.
156 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.VANT 2.sense and avoid 3.Framework 4.detecção de conflito 5.Sensores de aeronaves I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

AGRADECIMENTOS

Este trabalho só pôde ser possível pela contribuição de diversas pessoas, algumas que chegaram a contribuir sem mesmo saber de sua ajuda. Sem essas pessoas as ideias presentes neste texto não teriam sido criadas, ou não se teria encontrado determinação para fazer o trabalho necessário para que essas ideias se tornassem reais neste texto.

Por isso sou imensamente grato ao professor Paulo Sérgio Cugnasca, pelo seu excelente trabalho de orientação, pela paciência e por não desistir deste trabalho. Sem ele esta dissertação não teria sido possível. Também agradeço aos demais professores pertencentes ao Grupo de Análise de Segurança (GAS), os quais tive o prazer de conhecer.

Agradeço também a todos os alunos integrantes do GAS. As conversas que tivemos no laboratório foram importantes para ascenderem ótimas ideias, além de permitir ganhar diversos novos conhecimentos. Toda essa trocar de informação e momentos geraram parte do conteúdo presente nesta dissertação, além de descontrair em momentos de estresse devido às demandas da vida acadêmica.

Agradeço também a minha família, ao meu pai, à minha mãe e à minha irmã que sempre me apoiaram e incentivaram a seguir com meus estudos, mesmo que para isso, em algum momento, tivesse que abrir mão de momentos com eles.

E, por fim, quero agradecer aos diversos amigos que tornaram esse longo trajeto de estudos e trabalho mais fácil, seja com pequenos momentos de descontração, conversas ou incentivo.

RESUMO

O número de VANTs (Veículos Aéreos Não Tripulado) bem como o seu uso vêm aumentando consideravelmente nos últimos anos. Porém, sua inserção em um espaço aéreo não segregado pode se tornar uma ameaça às demais aeronaves que compartilham o mesmo espaço, pois permite o surgimento de cenários de risco de colisões entre elas. Para garantir que essa inserção ocorra em segurança são necessários testes que verifiquem a acurácia, a taxa de falhas e a confiabilidade de algoritmos de resolução de conflitos que poderiam ocasionar colisões. Por outro lado, testes com VANTs reais são muito custosos, demorados e geram riscos operacionais. Assim, realizá-los em simuladores torna-se uma boa alternativa para a avaliação desse processo. Este trabalho de pesquisa tem como objetivo a criação de um *framework* de simulação de técnicas de detecção e desvio de colisões (*sense and avoid*) por VANTs que permita calcular o risco de colisão entre estas aeronaves. Este *framework* foi concebido de modo modular, permitindo testes com diferentes tipos de sensores, diferentes técnicas de detecção de outros objetos e diferentes algoritmos de desvio, tendo em vista propiciar a busca de um conjunto de sensores e algoritmos que demonstrem o melhor resultado.

Palavras-Chave: VANTs, detecção, desvio, simulação.

ABSTRACT

The number of UAVs (Unmanned Aerial Vehicles) and their use has increased considerably in recent years. However, their insertion in a non-segregated airspace can become a threat to other aircraft that share the same space, as it allows the emergence of a risk of collision between them. To ensure that this insertion occurs safely, they are tested to verify the accuracy, failure rate and reliability of conflict resolution algorithms. On the other hand, testing with real UAVs is very costly, time-consuming and creates operational risks. Thus, conducting them in simulators becomes a good alternative for evaluating this process. This research work aims to create a framework for simulating the detection and avoidance collisions (feel and avoid) by UAVs that allows calculating the risk of collision between these aircraft. This framework was designed in a modular way, allowing tests with different types of sensors, different techniques for detecting other objects and different deviation algorithms in order to provide the search for a set of sensors and algorithms that demonstrate best results.

Keywords: UAVs, detection, avoidance, simulation.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURA E SIGLAS

ACAS	*Airborne Collison Avoidance System*
ACAS-Xu	*Airborne Collision Avoidance System-Xu*
ADS-B	*Automatic Dependent Surveillance-Broadcast*
AirSim	*Aerial Informatics and Robotics Simulation*
ATM	*Air Traffic Management*
ATCo	*Air Traffic Controller*
Eurocontrol	*European Organization for the Safety of Air Navigation*
FAA	*Federal Aviation Administration*
GPS	*Global Position System*
ICAO	*International Civil Aviation Organization*
IMU	*Inertial Measurement Unit*
KFC	*Kernelized Correlation Filters*
LIDAR	*Light Detection And Ranging*
MavLink	*Micro Air Vehicle Link*
MIL	*Multiple Instance Learning*
MINOS	*Multimodal Indoor Simulator*
MMW Radar	*Millimeter Wave Radar*
NASA	*National Aeronautics and Space Administration*
OACI	*Organização Internacional da Aviação Civil*
RF	*Random Forest*
SVM	*Support Vector Machine*
TCAS	*Traffic alert and Collision Avoidance System*
TLD	*Tracking Learning Detect*
UAV	*Unmanned Aerial Vehicle*
UTM	*Unmanned aircraft system Traffic Management*
VANT	Veículo Aéreo Não Tripulado

Sumário

1. Introdução

Devido à grande versatilidade dos VANTs (Veículos Aéreos Não Tripulados), estes veículos vêm ganhando diversas aplicações, tanto na área comercial, ou seja, com fins lucrativos, quanto no uso amador, com fins de divertimento. Na Figura 1 pode-se observar a distribuição do uso de VANTs para fins comerciais no Estados Unidos, considerando apenas veículos com valores acima de 2.500 dólares (FAA - Federal Aviation Administration, 2018, p. 101).

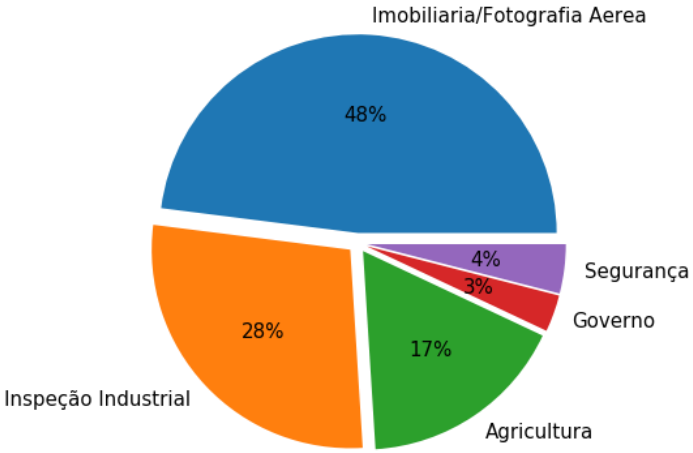

Figura 1: Uso de VANTs para fins comerciais nos EUA em 2017. Adaptado de (FAA - Federal Aviation Administration, 2018, p. 101)**.**

Esta versatilidade impulsionou um crescimento considerável tanto no número de veículos quanto no número de voos de VANTs, segundo a FAA (*Federal Aviation Administration*), que estimou que, em 2019, o número de VANTs nos Estados Unidos chegou a 1,705 milhões de unidades, sendo que destes, 1,32 milhões seriam usados para fins amadores[1] e 385 mil para fins comerciais. A perspectiva é que este número esteja entre 1,99 a 2,78 milhões em 2023, conforme ilustra a Figura 2, que apresenta

[1] Na estimativa foram considerados amadores aqueles veículos com custos abaixo de 2.500 dólares.

a tendência projetada pela FAA de evolução do número de VANTs nos próximos anos (FAA - Federal Aviation Administration, 2019, p. 1–107).

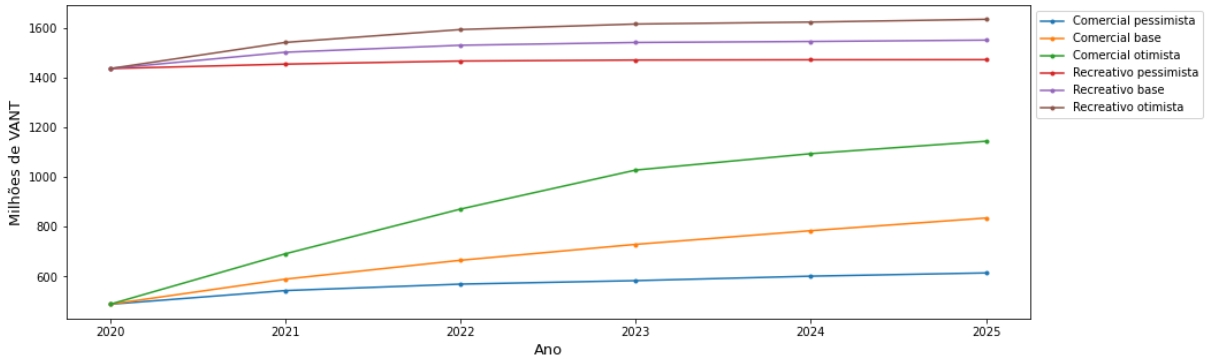

Figura 2: Projeção do número de VANTs nos EUA nos próximos anos (FAA - Federal Aviation Administration, 2021)**.**

Observa-se, nessa previsão, que há um grande aumento na quantidade de veículos comerciais, enquanto que ocorre uma estabilização nos modelos amadores. Esta é uma tendência observada na coleta de dados de 2018, pois houve uma queda no número mensal de novos registros de VANTs amadores, apresentando uma possível saturação nos veículos utilizados para este fim, enquanto que houve uma alta no número de novos registros mensais de veículos comerciais, impulsionados pela baixa de preços e criação de novas aplicações para o uso destes tipos de veículos.

No Brasil, o número de VANTs também está em constante crescimento. Em abril de 2022, a ANAC registrou um número total de 93.729 VANTs, divididos entre 52.906 para uso recreativo e 40.823 para uso profissional, como mostra a Figura 3 (ANAC 2020).

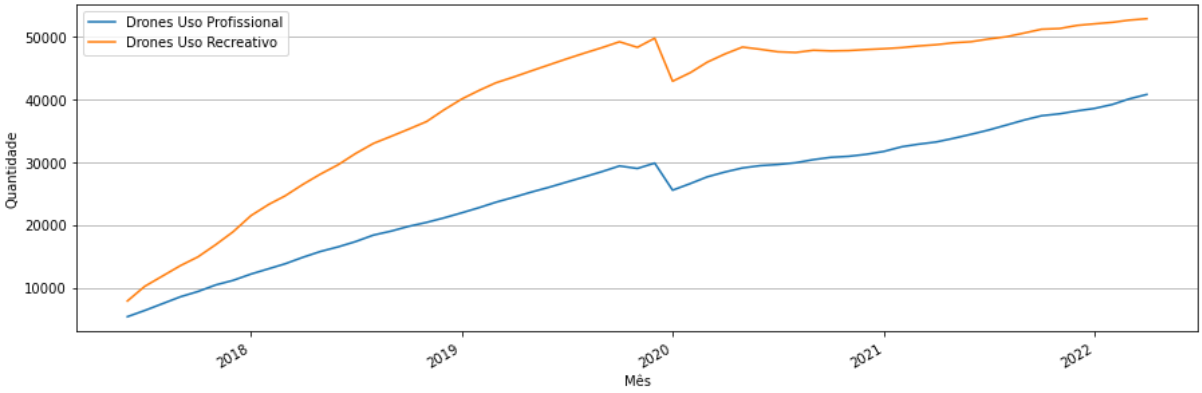

Figura 3: Número de *drones* cadastrados pela ANAC. Dados retirados da ANAC por meio do *link* (https://www.gov.br/anac/pt-br/assuntos/drones/quantidade-de-cadastros).

Porém, o aumento do uso deste tipo de veículo cria novos desafios, pois sua operação indiscriminada no espaço aéreo pode gerar risco de colisão com outras aeronaves. Para lidar com este tipo de risco, e permitir que a inserção de VANTs no espaço aéreo ocorra de modo seguro, torna-se necessário o uso de uma tecnologia mais robusta de prevenção de conflitos entre veículos aéreos envolvendo VANTs.

Neste trabalho de pesquisa é proposto um *framework* para a prevenção de colisão entre veículos aéreos, focados em VANTs. Tal *framework* visa ser concebido de maneira modular, permitindo utilizar diferentes sensores para a detecção de objetos (fixos ou móveis) e diferentes algoritmos para permitir o desvio do VANT para evitar a colisão. Neste trabalho propõem-se, também, um modelo de interação do *framework* com um ambiente de simulação.

1.1. Motivação

A inserção de um VANT em espaço aéreo que não seja segregado[2] pode se tornar uma ameaça às demais aeronaves que compartilham o mesmo espaço, pois permite

[2] Espaço Aéreo Segregado: Área Restrita, publicada em NOTAM – *Notice to Airman* (*mensagem que tem por finalidade divulgar alterações e restrições temporárias que possam ter impacto nas operações aéreas* [https://ajuda.decea.gov.br/base-de-conhecimento/o-que-e-notam]), no qual o uso do espaço

o surgimento de cenários de risco de colisões entre elas (Ramalingam et al., 2011, p. 448–55). Para garantir a segurança em possíveis cenários, o sistema de tráfego aéreo atual necessita da existência de pilotos embarcados nas aeronaves, os quais são responsáveis por parte da detecção de aeronaves em situação de conflito e pelas manobras necessárias para evitar uma possível colisão. Desse modo, para que a inserção de VANTs no espaço aéreo não segregado possa ocorrer de modo seguro, a ICAO (*International Civil Aviation Organization* ou OACI – Organização Internacional da Aviação Civil) exige que esses veículos tenham capacidade de impedir colisões entre aeronaves de forma similar, ou até melhor, do que aquela em que todas as aeronaves possuam pilotos humanos embarcados (ICAO, 2009).

A capacidade de os pilotos reconhecerem e resolverem conflitos ao pilotar uma aeronave tripulada recebe o nome de "*see and avoid"* (ver e evitar) (Lacher et al., 2007, p. 1–10), podendo ser dividida em duas fases distintas: a fase detecção de conflito, ou em outras palavras, de detecção de uma aeronave em rota de colisão (ação de "ver"); e a fase de resolução de conflito, composto das ações de criação e execução do plano de fuga (ação de "evitar"). Na fase de detecção de conflito, o piloto utiliza-se tanto de sua visão quanto do auxílio de dados oriundos de vários instrumentos e sensores presentes na aeronave. Já a resolução de conflito é feita com base no julgamento do piloto sobre a detecção e na sua habilidade para preparar e executar um plano de desvio, visando a solucionar o conflito.

Várias pesquisas (por exemplo, Bharati et al., 2018, p. 1–1; Sapkota et al., 2016, p. 1556–61; Fasano et al., 2008) vêm sendo conduzidas com o objetivo de se construir um sistema que simule as mesmas habilidades em um VANT daquelas encontradas em uma aeronave com um piloto embarcado. Essa técnica recebe o nome de "*sense*

---

aéreo é exclusivo para usuário específico, não compartilhado com outras aeronaves, sejam tripuladas ou não tripuladas. ANAC 2019, disponível em http://www2.anac.gov.br/anacpedia/por_ing/tr4131.htm. Acesso em: 05 nov 2019.

and avoid" ("sentir e evitar") e, assim como a habilidade do piloto, ela é dividida em duas fases. A primeira fase, o sensoriamento ("*sense*"), em que, por meio de um conjunto de sensores o VANT se torna capaz de identificar aeronaves ou outros obstáculos a sua volta, e a segunda fase, a resolução de conflito ("*avoid*"), na qual com o uso de algoritmos de inteligência artificial é possível se traçar uma nova rota em que o veículo desvia desses obstáculos sem interferir, ou interferindo o mínimo possível, na sua missão inicial.

Em pesquisas realizadas, ambos, o sensoriamento e o desvio, podem ser realizados de diversas formas. Na fase de detecção pode-se utilizar de diversos tipos de sensores, bem como de algoritmos para detecção de objetos. Dentre os sensores, os mais utilizados em diferentes pesquisas estão as câmeras visuais (Bharati et al., 2018, p. 1–1), as câmeras termais (Carrio et al., 2017), os radares (De Haag et al., 2016, p. 1–11), o LIDAR (sigla em inglês, *Light Detection And Ranging*) (Ramasamy et al., 2016, p. 344–58), os sensores acústicos (Kapoor et al., 2018, p. 1–25), o TCAS (*Traffic Collision Avoidance System*) ou, ainda, fusões de diferentes combinações desses sensores (Fasano et al., 2008, p. 338–60). Na fase de resolução de conflito pode-se variar os algoritmos de desvio e, neste contexto, algumas pesquisas utilizam diferentes técnicas de inteligência artificial, como algoritmos de aprendizagem supervisionada, algoritmos de aprendizagem por reforço (MATSUMOTO, 2013, p. 1689–99), planejamento automático (Jung, 2013), heurísticas (Nikolos et al., 2003, p. 898–912) e processos de decisão markovianos (Yu and Zhang, 2015, p. 152–66), dentre outros.

Para garantir a segurança destes sistemas são necessários testes que verifiquem sua acurácia, taxa de falhas e confiabilidade. Porém, testes com os VANTs reais seriam muito custosos, demorados e gerariam riscos de acidentes. Assim, simulações tornam-se uma boa alternativa para a avaliação desses sistemas. Para permitir testes com diferentes sensores, como câmera e radares, um simulador com este propósito deve disponibilizar de forma realista imagens 3D do evento simulado e a dinâmica utilizada no voo de um VANT.

1.2. Objetivo

O objetivo principal deste trabalho é desenvolver um sistema modular de *sense and avoid* aplicado a VANTs, concebido em formato de *framework*, permitindo desta maneira a utilização de diferentes tipos de sensores e algoritmos em conjunto, bem como suas trocas, sem a necessitar de alterações em todo o sistema.

Com o intuito de construir o *framework* modular é necessário que ambas as fases (a de detecção de conflito e a de desvio) trabalhem de forma independente e coesa. Para isso, necessita-se de uma modelagem do sistema seguindo alguns padrões de Engenharia de *Software*.

Para permitir a utilização de diferentes tipos de sensores e algoritmos de desvio, em cada fase, e de maneira simples, pode-se utilizar de um protocolo genérico de comunicação entre as fases, de modo que seja flexível permitir o uso de diversos tipos de sensores para detecção de conflito e de diversos tipos de algoritmos de desvio, que possam ser intercambiados de acordo com a necessidade.

Assim, as criações do *framework*, com base em técnicas de Engenharia de *Software*, bem como do protocolo genérico de comunicação entre as fases tornam-se passos importantes para se conseguir atingir o objetivo principal da pesquisa, constituindo-se de objetivos secundários do trabalho.

Por fim, faz-se necessário criar um ambiente de testes que permita verificar a viabilidade do *framework* proposto. Com esta finalidade é importante a construção da interação do *framework* proposto com um simulador de voo de VANTs, permitindo-se testar diferentes tipos de sensores e de técnicas de desvios de maneira simples e ágil. Essa interação é um objetivo secundário adicional deste projeto de pesquisa.

1.3. Justificativa

Devido ao grande número de tipos de sensores capazes de serem utilizados na detecção de objetos por aeronaves, assim como as várias abordagens possíveis para um algoritmo de desvio de objetos em risco de colisão, um modelo genérico de *sense and avoid* em veículos aéreos se torna uma boa alternativa para permitir a utilização destas múltiplas abordagens.

Com a criação de um *framework* modular é possível suprir esta demanda, permitindo o fornecimento de suporte à utilização de algoritmos diferentes, por meio do uso de técnicas de Engenharia de *Software.* Modelando este *framework* para que a fase de detecção se torne expansível, permite-se a utilização de diferentes sensores ao mesmo tempo, possibilitando que as vantagens de alguns sensores possam cobrir as desvantagens de outros em determinadas situações.

Após a criação deste *framework* são necessários testes que provem sua eficácia. Para garantir a segurança, confiabilidade e precisão de um sistema de *sense and avoid* são necessários que os testes contenham diferentes cenários. Porém testes com veículos reais ou diversos sensores diferentes geram custos e riscos. Assim, realizar testes em ambientes simulados são necessários para suprir estas dificuldades.

Com a criação de um ambiente de simulação que tenha interação com o modelo proposto torna-se possível a realização de diversos testes, alterando-se sensores, algoritmos e cenários.

1.4. Contribuição

Com a construção de um *framework* é criada uma estrutura de código genérica, que pode ser utilizada por diferentes sistemas ou veículos (sejam eles simulados ou reais). O *framework* cria um fluxo de operações, determinando a função de cada etapa do processo de *sense and avoid* e, como estas etapas se relacionam entre si. Porém, ele não determina como cada etapa executa esta função, permitindo que um programador (no papel de usuário do *framework*) combine diferentes técnicas para executá-lo.

Assim, a criação no modelo genérico facilita a programação de um sistema de *sense and avoid*, ao mesmo tempo que permite que diferentes sistemas e veículos utilizem o mesmo fluxo de execução, facilitando a comparação entre diferentes técnicas de sensoriamento e de algoritmos de desvio.

A propriedade modular deste *framework* permite a troca de sensores de modo simples, permitindo que um modelo de veículo tenha as alterações que o programador (usuário do *framework*) ou piloto julgar necessárias. Assim, o usuário do *framework* que considerar ineficiente o teste do sistema de *sense and avoid* para um certo veículo poderá acoplar a ele um novo sensor ou, caso decida ter um modelo mais simples, poderá retirar um sensor desse veículo.

Esta propriedade modular do *framework* também atribui uma maior confiabilidade à detecção de um veículo com vários sensores, pois caso ocorra falha de um sensor, o sistema poderá automaticamente utilizar um segundo sensor para suprir sua demanda, adaptando seu sistema de detecção de maneira a não necessitar do sensor em falha.

A integração deste modelo com um ambiente de simulação permite a criação de testes do sistema sem a necessidade da construção de um veículo, reduzindo os custos e os riscos deste processo. Ela também viabiliza a comparação de diferentes tipos de sensores e técnicas de desvio, o que possibilita, em um trabalho futuro, testes com a intenção de se identificar qual a melhor combinação desses componentes para um determinado sistema de *sense and avoid*.

1.5. Estrutura do trabalho

Este trabalho está divido em nove capítulos. O primeiro capítulo, Introdução, contextualiza o problema abordado e apresenta o objetivo pretendido. No segundo capítulo, Revisão da Literatura, são apresentados técnicas e *frameworks* atualmente utilizados no contexto de *sense and avoid*, dividindo-se essa apresentação em

revisões sobre técnicas de detecção de objetos em risco de colisão com aeronaves e técnicas de desvio destes objetos pelas aeronaves (VANTs). No capítulo 3, Sensores de Detecção, são apresentados os tipos de sensores utilizados para detecção de outras aeronaves ou objetos, mostrando seus funcionamentos e as individualidades de cada um; neste capítulo destacam-se, ainda, os sensores cooperativos e os sensores não cooperativos. No capítulo 4 é apresentado um conjunto de algoritmos criados para a utilização destes sensores como um sistema de detecção. No capítulo 5, Arquitetura de *Sense and Avoid*, é discursado sobre o modelo proposto neste trabalho, exibindo-se seu fluxo, suas fases, seus protocolos e sua estrutura de classes a partir de um diagrama UML, estando ele dividido em duas visões: a visão de alto nível, na qual é apresentada a interação entre os subsistemas do *framework*, e a visão aprofundada, na qual se apresenta detalhes de sua implementação. O capítulo 6 tem foco na simulação de voos, apresentando os simuladores existentes, com grande foco no AirSim, escolhido para os testes deste trabalho. O capítulo 7 apresenta as estratégias de testes utilizadas, explicando o cenário de teste criado, as métricas utilizadas e os vários algoritmos analisados para demonstrar a modularidade e flexibilidade do *framework*. No capítulo 8 são apresentados os resultados dos testes, exibindo-se quantitativamente os resultados dos algoritmos, para verificar quais apresentaram os melhores desempenhos. Por fim, no capítulo 9, discutem-se os resultados obtidos e são apresentadas as conclusões que se podem tirar a partir deles. Ainda, no Anexo A, encontram-se algumas informações adicionais a respeito do *framework* desenvolvido e alguns aspectos do seu uso.

2. Revisão da literatura

Sense and avoid é uma área de pesquisa relativamente recente, porém já conta com vários trabalhos em andamento e alguns concluídos. Apesar de a detecção e o desvio serem duas fases complementares desta técnica, pesquisas encontradas geralmente focam em apenas uma dessas fases, e costumam ser testadas com VANTs reais ou em ambientes com simulações parciais, nos quais simulam-se somente aspectos essenciais para testar a fase da técnica em questão. Deste modo, para uma revisão de literatura completa é necessário estudar separadamente cada fase, a detecção e o desvio, verificando-se quais os sensores estão sendo utilizados para detecção e quais as técnicas estão sendo aplicadas a eles, além de quais algoritmo de desvio estão sendo utilizados.

Neste capítulo são apresentadas as pesquisas recentes nessa área, separando-se os trabalhos de detecção de conflito e os trabalhos de desvio para evitar colisão.

2.1. Detecção de conflito

A detecção de conflito é a fase em que há maior diversidade de pesquisas, pois existem várias maneiras de se realizá-la, podendo-se alterar o tipo de sensor ou o algoritmo utilizado. Em (Lacher et al., 2007, p. 1–10) e (Ramasamy and Sabatini, 2015) é descrita uma série de sensores que podem ser utilizados para detecções por aeronaves. Esses sensores são divididos em duas diferentes categorias, conforme a técnica utilizada para a detecção: a cooperativa, na qual a identificação de colisão é feita por meio de troca de informações entre as aeronaves, e a não cooperativa, na qual uma aeronave identifica objetos usando somente dados oriundos de seus próprios sensores, sem a necessidade de comunicação externa.

As detecções cooperativas correspondem aos modelos mais simples, pois neste tipo de detecção não é necessário identificar a posição da outra aeronave, mas apenas ler e interpretar os dados enviados por ela, os quais contêm informações sobre sua posição, direção e velocidade. Por se tratar de um modelo mais simples, este tipo de

detecção também é a mais estudada e consolidada, possuindo níveis de segurança e confiabilidade que permitem que ela, atualmente, seja amplamente utilizada na aviação, chegando a fazer parte de alguns equipamentos obrigatórios em aviões de grande porte. Dentre estes equipamentos estão incluídos o *Automatic Dependent Surveillance-Broadcast*[3] (ADS-B) e o *Traffic Alert and Collision Avoidance System*[4] (TCAS) (FAA - Federal Aviation Administration, 2011, p. 1–50).

Em (Kuchar, 2005, p. 30) é feita uma análise de aspectos de segurança sobre a aplicação do TCAS como sistema de prevenção de colisão no VANT "*Global Hawk*", ilustrado na Figura 4. Este sistema é capaz de identificar aeronaves equipadas com *transponder* por intermédio de troca de informações entre elas. Kuchar apresenta uma análise dividida em duas fases: a primeira fase, chamada de *loop externo*, utiliza-se de uma árvore de falhas para verificar situações que levam um VANT a um cenário de colisão; a segunda fase, chamada de *loop interno*, implementa simulações com a técnica de Monte Carlo para encontrar possíveis falhas que levariam o VANT a uma colisão com outras aeronaves. Nesses testes foram encontradas taxas de falhas com valores dentro dos permitidos pela ICAO (*International Civil Aviation Organization*) e pela Eurocontrol (*European Organization for the Safety of Air Navigation*). Porém, como os modelos de teste utilizados eram específicos para aeronaves tripuladas, é necessário adaptá-los para serem utilizados em VANTs.

---

[3] ADS-B: Tecnologia cooperativa de vigilância de aeronaves por meio de troca de informações entre aeronaves e entre a aeronave e a torre de controle (FAA - Federal Aviation Administration, 2011, p. 1–50).

[4] TCAS: Sistema de auxílio na resolução de conflitos, fornecendo um aviso de possíveis colisões e sugestões de manobras evasivas (FAA - Federal Aviation Administration, 2011, p. 1–50).

Figura 4: Ilustração do VANT "*Global Hawk*" (Watts et al., 2012, p. 1671–92)**.**

O TCAS é uma tecnologia em constante atualização e hoje conta com duas versões diferentes: TCAS e TCAS II. Porém, há uma terceira versão em estudo, já em fase inicial de uso, que está sendo criada para ser substituta das versões anteriores, e que segue um conceito de tecnologia conhecido como *Airborne Collison Avoidance System* (ACAS) (Eurocontrol, 2017). Este conceito descreve um novo sistema de detecção, o qual utiliza principalmente o *transponder* como forma de comunicação similar ao encontrado nos TCAS, porém é capaz de utilizar outros sensores, tornando-se um modelo hibrido entre o sistema cooperativo e o não cooperativo. Esta tecnologia contará com uma versão exclusiva para o uso em VANTs, denominada ACAS-Xu, com diferentes técnicas para resolução de conflito e diferentes modelos de comunicação. Esta tecnologia ainda está em fase de desenvolvimento e ainda não existem dados de testes a respeito de sua eficácia ou acurácia na detecção de outras aeronaves.

Porém, a detecção cooperativa não pode ser vista como um modelo definitivo de *sense and avoid*, pois para o seu funcionamento é necessário que ambas as aeronaves tenham os sensores necessários para a troca de informação, algo que não é atualmente a realidade em aeronaves mais simples, como monomotores,

planadores, pequenos VANTs, ou em outros objetos voadores, como balões e dirigíveis. Sendo assim, para um sistema de detecção completo torna-se necessário que os VANTs tenham também sistemas de detecção não cooperativa, permitindo a identificação de objetos em risco de colisão apenas com o uso de seus próprios dados, sem necessitar de comunicação com um agente externo.

Técnicas de detecção não cooperativa variam conforme o sensor utilizado. Dentre elas, a visão computacional está entre as mais estudadas, por necessitar apenas de uma câmera, que é um sensor relativamente leve e barato. Dentre os trabalhos relacionados com *sense and avoid* utilizando câmeras, um dos primeiros resultados foi descrito por Ross (Ross et al., 2013, p. 1765–72), que utilizando o algoritmo de aprendizado supervisionado DAgger, o qual aprendeu a reconhecer a colisão e evitá-la a partir de dados de voos feitos por um piloto, possibilitou a um VANT sobrevoar uma floresta, desviando-o de árvores presentes em seu caminho. Porém, seu algoritmo limita-se a desvio de obstáculos fixos.

Na mesma época, Pestana et al. (Pestana et al., 2013) utilizou o algoritmo OpenTLD para reconhecer objetos em movimento e rastreá-los, apresentando bons resultados, mesmo quando o objeto “se perde” da câmera em poucos momentos. Mas esta técnica se limita a objetos com baixas velocidades, diferentemente do ocorrido no espaço aéreo. Outra desvantagem deste trabalho é seu foco apenas no rastreamento, não incluindo a detecção, sendo necessário informar ao algoritmo, para o seu funcionamento, a localização inicial do objeto a ser rastreado.

Em uma pesquisa mais recente Wu (Wu et al., 2017, p. 1–9) utiliza um algoritmo para que um VANT reconheça aeronaves em movimento, tendo conseguido boa precisão e altas taxas de sucesso na detecção, sendo considerado o estado da arte para detecção de aeronaves. Seu algoritmo tem a detecção baseada em filtros de Kalman e possui duas fases distintas de escaneamento da imagem.

Apesar da ampla quantidade de pesquisas, a visão computacional, quando aplicada em câmeras RGB, tem limitações em ambientes com baixa visibilidade ou que

apresentem mudanças bruscas de iluminação, fato que ocorre à noite ou durante alguma condição climática adversa, como durante nevoeiros. Para suprir estas limitações, e na tentativa de se criar um sistema mais robusto, alguns autores optam por utilizar outros tipos de sensores ou um conjunto deles, criando algoritmos de fusão de sensores ("*sensor fusion*"). Propõe-se, em Carrio et al. (Carrio et al., 2017), uma técnica utilizando câmeras infravermelhas em conjunto com sensores ADS-B. Com essas câmeras capta-se o calor emitido pelos objetos e, com isso, é possível cobrir as falhas ocorridas por falta de iluminação ou visibilidade nas câmeras RGB convencionais. Seus testes apresentaram bons resultados em dias nublados, porém apresentaram um grande número de falhas em períodos noturnos, pois as aeronaves não retinham muito o calor nessas condições, atrapalhando o processo de detecção pelas câmeras termais.

Em (Fasano et al., 2008, p. 338–60) descreve-se e testa-se um sistema de fusão de sensores, utilizando nele um radar do tipo *Ka-band*[5] em conjunto com câmeras visuais RGB e câmeras infravermelhas, na tentativa de se detectar uma situação de conflito e desviar o VANT controlado de uma possível colisão com uma outra aeronave.

Em uma série de artigos (Ramasamy et al., 2014, p. 271–76) e (Ramasamy et al., 2016, p. 344–58) descrevem um conjunto de sensores de detecção promissores para uso em VANTs, avaliando as vantagens e desvantagens de sua utilização. Dentre os sensores estudados, estão o LIDAR, o MMW Radar, os sensores acústicos, as câmeras termais e óticas, além dos sensores cooperativos. É apresentada uma modelagem de fusão de sensores para detecção na qual sensores cooperativos recebem prioridade na detecção e as detecções secundárias são feitas por meio de sensores não cooperativos, dentre eles os sensores acústicos, os radares e as câmeras visuais e termais, que por sua vez são complementares umas às outras.

---

[5] Radar *Ka-band*: Um radar de frequência de rádio alta, utilizando as bandas Ka.

2.2. Comparação dos métodos de detecção

Nos trabalhos apresentados são utilizados vários tipos diferentes de sensores para a detecção, divididos entres cooperativos e não cooperativos. Na Tabela 1 compara-se essas pesquisas conforme os sensores utilizados e os tipos de teste empregados. Observa-se que existe uma grande variedade de sensores estudados para este fim, o que evidencia que neste campo ainda não existe um consenso sobre o melhor grupo de sensores a ser utilizado. Um estudo comparativo entre sensores e técnicas pode ser um bom caminho para se encontrar o grupo ideal de sensores a ser utilizado. Um *framework* que permita o uso de diferentes conjuntos de sensores para detecções pode ser uma ferramenta que auxilie neste tipo de comparação. Outro ponto que pode ser inferido a partir dessa tabela é que dentre os diversos modelos de testes utilizados nas pesquisas não existe um modelo ideal ou padronizado o qual possa ser utilizado para comparar diretamente todas a pesquisas. Assim, um modelo de detecção genérico, que possa se utilizar de vários sensores em conjunto, além de testar várias técnicas diferentes, é uma pesquisa útil e relevante para criação de um modelo de detecção e comparação com os modelos já estudados individualmente.

	Uso de Detecção Cooperativa			Uso de Detecção Não cooperativa				Tipo de Teste
Autores	**TCAS / ACAS**	**ADS-B**	**Comunicação**	**Câmera Visual**	**Câmera Termal**	**RADAR**	**LIDAR**	
(Kuchar, 2005, p. 30)	**X**							Árvore de decisão, simulação
(Eurocontrol, 2017)	**X**		**X**					Em desen-volvimento
(Ross et al., 2013, p. 1765–72)				**X**				Voos reais
(Pestana et al., 2013)				**X**				Voos reais
(Fu et al., 2014, p. 5441–46)				**X**				Simulação, voos reais
(Wu et al., 2017, p. 1–9)				**X**				Simulação, vídeos
(Carrio et al., 2017)		**X**			**X**			Teste de sensores
(Fasano et al., 2008, p. 338–60)				**X**		**X**	**X**	Simulação
(Ramasamy et al., 2016, p. 344–58)	**X**	**X**	**X**	**X**	**X**	**X**	**X**	Simulação

Tabela 1: Comparação das abordagens de detecção de outras aeronaves segundos as referências bibliográficas consultadas.

2.3. Técnicas de desvio

Apesar de crucial para um algoritmo de *sense and avoid*, a detecção é apenas a fase inicial deste tipo de sistema. Após a detecção do objeto em risco de colisão com o VANT é necessário traçar uma rota de desvio deste objeto que permita que a aeronave controlada evite a situação de risco. Porém, essa rota de desvio deve ser otimizada de modo a desviar o VANT o mínimo possível do seu planejamento de rota inicial,

evitando gastos adicionais de tempo e energia (combustível, carga elétrica), ao mesmo tempo em que se mantém a segurança operacional das aeronaves.

Assim como na fase de detecção, também para a operação de desvio podem ser utilizadas diversas técnicas diferentes. Todas as técnicas de desvio se iniciam com os mesmos dados, oriundos dos sensores, alterando-se somente o algoritmo utilizado para replanejamento da rota.

Atualmente o sistema aéreo tem como principal técnica para resolução de conflito um sistema baseado na comunicação e na vigilância, centrada no sistema de controle de tráfego aéreo, o ATC (do inglês, *Air TrafficControl*). Os controladores de tráfego aéreo (ATCos) gerenciam o tráfego, garantindo a separação entre as aeronaves em um nível considerado seguro nas regiões do espaço aéreo controlado, visando reduzir a probabilidade da ocorrência de fatores que possam levar a colisão. Para isto eles atuam com sistemas de vigilância que monitoram os dados das aeronaves, como posição, nível de voo, altitude, horário e outros. Estes dados podem ser recebidos através de radiocomunicação por notificações enviadas pelos pilotos, nos casos conhecidos como *Vigilância Convencional*, ou obtidos de formas independentes, como através de leituras de sensores, por exemplo na *Vigilância por Radar*. Com a leitura desses dados os ATCos constroem um mapa situacional sobre o espaço aéreo controlado, podendo intervir sobre os voos caso necessário, por meio de instruções ou autorizações ao piloto (DECEA, 2016) (Vismari, 2007).

Este sistema utiliza alguns sensores para auxiliar a vigilância, identificação, comunicação das aeronaves, ou na criação de planos de contingência, caso for encontrado um cenário de risco de colisão. Dentre eles destaca-se o TCAS, um tipo de sensor cooperativo, presente em ambas as aeronaves, que ao detectar um cenário de colisão planeja novas rotas para as aeronaves, capazes de impedir que as mesmas entrem em situação de risco de colisão. Tais rotas são calculadas a partir de um conjunto de regras prefixadas, levando-se em conta a posição relativa de cada aeronave, suas capacidades de voo e a existência de outras aeronaves ao redor. Em sua primeira versão, o TCAS calculava uma rota de fuga sem ter conhecimento de

planejamento da outra aeronave, denominada “intrusa”. Porém, em sua versão mais recente, o TCAS II, esta rota pode ser traçada em cooperação com a outra aeronave, com ambas trocando informações sobre a rota pretendida, trançando uma rota que impeça que ambas tomem a mesma direção, evitando a criação um novo cenário de risco (Eurocontrol, 2017). Neste cenário, a resolução de conflito é executada apenas com a troca de informação entre as aeronaves, dispensando-se a comunicação com o ATM, o que torna descentralizada a resolução de conflito.

Em um novo sistema de prevenção de colisão, chamado ACAS-X *(Airborne Collision Avoidance System* - versão X*)*, está em estudo uma nova técnica de cálculo de rota de desvio, utilizando métodos estocásticos, para estimar os possíveis estados futuro da aeronave a partir de modelos probabilísticos de dinâmica das aeronaves ou da detecção. Com base nestes dados, cria-se uma árvore de estados, encontrando-se o estado mais provável da aeronave a partir de técnicas de programação dinâmica. Com isso, este modelo propõe a troca de um sistema baseado em regras por um sistema probabilístico (Kochenderfer et al., 2013, p. 17–33) (Koenig and Howard, 2004, p. 2149–54).

Em diversas pesquisas, diferentes abordagens de resolução de conflito são estudadas. Em (MATSUMOTO, 2013, p. 1689–99) é proposto um sistema baseado em aprendizado por demonstração, no qual o algoritmo treina métodos de resolução de conflito com base em banco de dados baseado em exemplos de execução realizados por um especialista. Em (Yu and Zhang, 2015, p. 152–66) é apresentada uma revisão do estado da arte sobre planejamento de rota para VANTs em condição de risco de colisão ou com base em um planejamento inicial. Nesse trabalho, as abordagens são divididas em cinco grupos, conforme a técnica utilizada: abordagem baseadas em amostragem, abordagem utilizando otimização numérica, abordagem dissociativa, abordagem utilizando heurísticas de inteligência artificial, e outras abordagens.

2.4. UTM

Recentemente, a NASA (*National Aeronautics and Space Administration*), em parceria com a *Federal Aviation Administration* (FAA), está estudando um novo modelo de controle de tráfego aéreo, focado especialmente em aeronaves não tripuladas, chamado de *Unmanned aircraft system Traffic Management* (UTM). Seu foco é prover um sistema para gerenciamento do tráfico aéreo em baixas altitudes, abaixo de 400 pés, no qual os operadores são responsáveis pela coordenação, execução e gerenciamento de operações com base nas regras de voos da FAA (FAA - Federal Aviation Administration, 2018).

Esse sistema irá trabalhar de forma preventiva em situações de conflito, oferecendo serviços de plano de voo, comunicação, separação de aeronaves, dentre outros. Nele, antes de cada voo, os operadores de VANTs devem compartilhar suas intenções de voo, incluindo a rota desejada. Essas intenções são lidas pelos demais operadores, pelos veículos ou pela própria FAA, com a intenção de coordenar operações de prevenção de conflito, criando, assim, um sistema de gerenciamento do espaço aéreo e prevenção de colisão compartilhado.

A arquitetura do UTM está apresentada na Figura 5: Arquitetura UTM. Nela os **operadores do VANT** (*UAS operator*) são responsáveis por enviar dados de suas intenções de voo, com as manobras a serem feitas e os seus horários. Esses dados são recebidos pelo **fornecedor de serviço para o VANT** (*UAS service supplier)*, que irá validar a viabilidade do plano de voo, por intermédio dos dados do sistema **FIMS** (do inglês *Flight Information Management System*) e dos dados públicos, enviando, verificando sua aderência as regras de operações, responsabilidades e possíveis risco em comparação com as demais rotas enviadas. Também serão analisados dados meteorológicos e de vigilância, sendo que estes últimos estarão disponíveis para os operadores dos veículos por meio do **provedor de serviço suplementar** (*Supplemental Data Service Provider)*. Havendo aderência às regras, e estando livre de riscos, o plano de voo é aceito, e dados auxiliares serão enviados ao controlador do VANT.

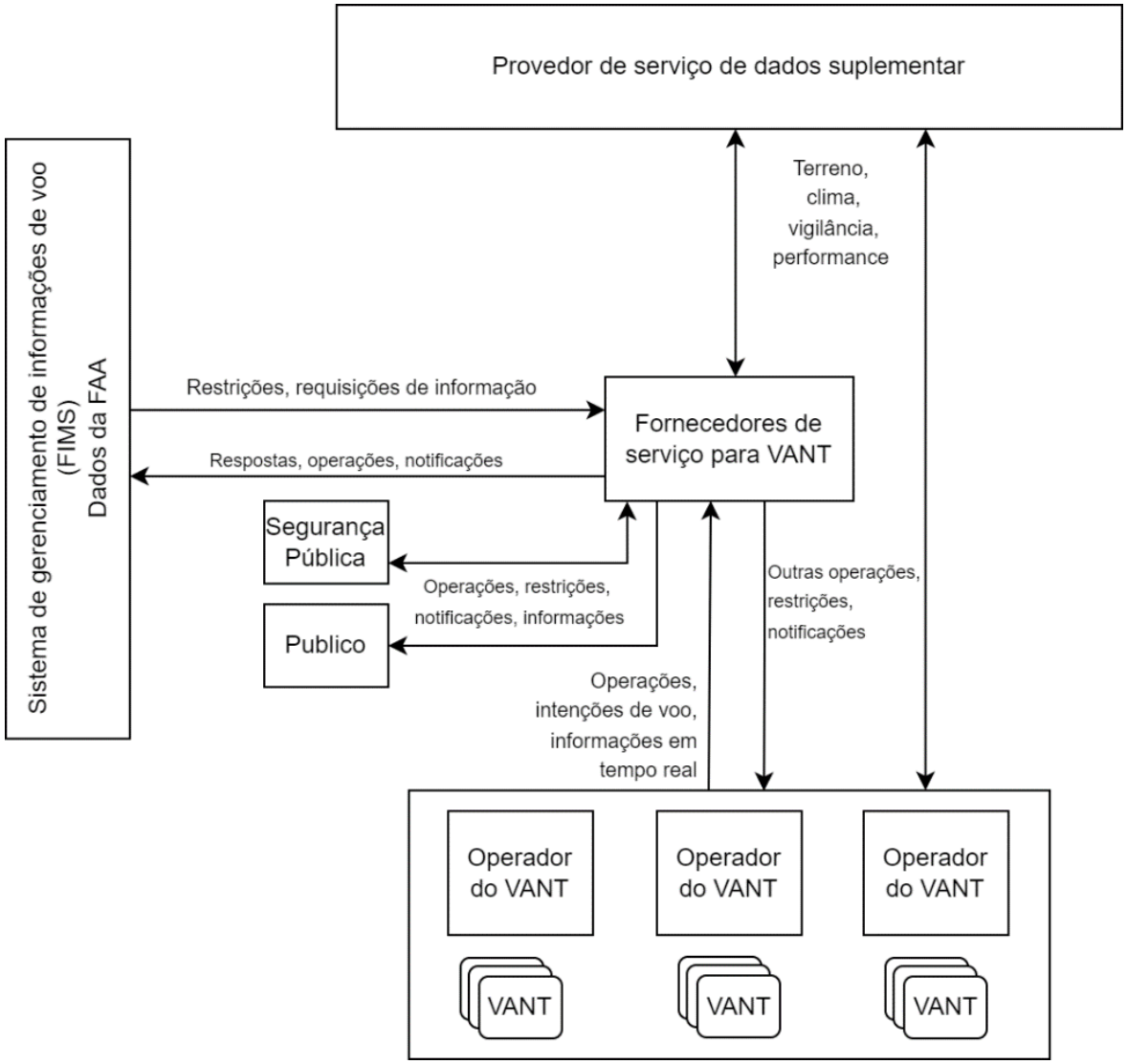

Figura 5: Arquitetura UTM, Adaptado de (FAA - Federal Aviation Administration, 2021, p. 1–317)

A proposta do UTM foca na prevenção de colisão, mitigando a probabilidade de risco, segundo o plano de voo das aeronaves, criando planos de voos livre de colisão. Assim, para o melhor funcionamento deste modelo é necessário que todos os planos de voo sejam previamente enviados ao sistema de controle. Porém, ainda é possível encontrar riscos de colisão com aeronaves ou outros objetos voadores (como balões) que não enviaram seus dados de voo, para o sistema de controle, além de aeronaves que não sigam os planos enviados previamente.

Com isso o sistema UTM abre espaço para estudos de *sense and avoid*, permitindo uma integração do presente trabalho com esse modelo de controle de trafego aéreo. Uma possibilidade de integração é o sistema de *sense and avoid* ter um papel auxiliar no sistema de vigilância do UTM (*UAS service supplier*). Uma aeronave, ao encontrar o um objeto voador ou uma outra aeronave, que não estava presente nos planos de rotas iniciais, pode notificar ao sistema do UTM para que este novo objeto não seja um risco as demais aeronaves. Assim, também será necessário um sistema para replanejamento de rotas presentes nas aeronaves, para evitar colisões com este novo objeto encontrado. Desse modo, o sistema descrito neste trabalho pode funcionar em sinergia com a atual proposta do UTM.

3. Sensores de detecção

Como apresentado no capítulo 2, diferentes técnicas de detecção são utilizadas em pesquisas envolvendo VANTs, sendo que elas se distinguem principalmente pelo sensor e pelo algoritmo utilizados para o tratamento de seus dados. Cada uma destas técnicas apresenta vantagens e desvantagens, além de um modelo específico de detecção e comunicação de seus resultados.

Para a criação de um modelo genérico de detecção de outros objetos ou aeronaves é necessário conhecer a especificidade de cada técnica e como elas comunicam seus resultados, visando criar um protocolo capaz de aglutinar o maior número de técnicas possível. Para isso é fundamental um estudo minucioso de cada sensor. Neste capítulo são apresentados detalhes do funcionamento, modelos de detecção e taxas de falhas dos sensores utilizados para detecção de possíveis cenários de colisão envolvendo VANTs e aeronaves, criando-se assim os alicerces necessários para criação de um modelo genérico de detecção.

Assim como ocorre no processo de detecção, os sensores também podem ser divididos em dois grandes grupos: os sensores cooperativos e os sensores não cooperativos. Os sensores cooperativos são aqueles nos quais a detecção ocorre a partir da troca de dados entre as duas aeronaves, em que uma primeira aeronave envia dados de sua posição e de sua direção, enquanto que uma segunda aeronave recebe estes dados, calculando a trajetória da primeira; ou seja, nesse tipo de sensor, as duas aeronaves trabalham em conjunto, “colaborando” uma com a outra e, para isso, é necessário que ambas tenham sensores cooperativos nelas embutidos. Já os sensores não cooperativos são aqueles que não necessitam da comunicação ou colaboração entre aeronaves, conseguindo detectar a outra aeronave ou objeto em rota de colisão por meio de dados lidos do ambiente ou enviados pelo seu próprio equipamento; desta maneira, apenas a aeronave rastreadora necessita conter o sensor, enquanto que o objeto rastreado não necessita ter ciência dessa operação. A seguir apresentam-se os sensores presentes em VANTs segundo essa classificação.

3.1. Sensores cooperativos

Sensores cooperativos correspondem ao grupo de sensores utilizados na detecção de outras aeronaves, têm sido amplamente estudados pela comunidade científica e são mais confiáveis, pois neste grupo de sensores não é necessário identificar a posição da outra aeronave detectada, apenas ler e extrair os dados enviados por ela, nos quais estão essas informações. Por se tratar de um conhecimento mais consolidado, estes sensores já são amplamente utilizados na aviação, sendo que alguns chegam a ser obrigatórios em aviões de grande porte, com várias características de uso regulamentadas.

Entre os sistemas mais famosos deste grupo estão o *Automatic Dependent Surveillance – Broadcast* (ADS-B), dispositivo obrigatório nos Estados Unidos da América a partir de 2020, o *Traffic Collision Avoidance System* (TCAS), obrigatório desde de 2000, que conta com duas diferentes versões (TCAS e TCASII), e seu futuro substituto, o *Airborne Collison Avoidance System* (ACAS-X), que tem uma versão específica para VANTS, o ACAS-Xu. Estes três sensores são detalhados a seguir.

3.1.1. *Automatic Dependent Surveillance – Broadcast* (ADS-B)

O ADS-B é um sistema de vigilância no qual as aeronaves compartilham dados posicionais, enviando mensagens em *broadcast* por meio de sinais de rádio. Essas mensagens são lidas por um equipamento de recepção, que pode estar presente em outras aeronaves, estações terrestres ou sistema de controle de tráfego aéreo. Ele é a principal tecnologia do programa *NextGen Surveillance*, o qual pretende atualizar os serviços de controle aéreo, tendo em vista melhorar sua flexibilidade, segurança, capacidade e eficiência. A FAA tornou o ADS-B obrigatório nos Estados Unidos da América a partir de 1º de janeiro de 2020 para aviões em operações no espaço aéreo onde a separação é controlada.

Existem dois formatos diferentes de equipamentos ADS-B: o *ADS-B out*, responsável por enviar as mensagens, e o *ADS-B in*, responsável recebê-las. Por utilizar um paradigma de comunicação *broadcast*, não é necessária resposta para uma mensagem recebida. Dessa forma, ambos os equipamentos trabalham de forma independente. Uma aeronave equipada com *ADS-B out* não precisa necessariamente conter o *ADS-B in*; assim, envia dados sem conhecimento de que algum equipamento de outra aeronave ou de uma estação terrestre o esteja recebendo (ICAO, 2013, p. 1–61).

As aeronaves equipadas com ADS-B recebem dados de sua posição via GPS e sensores de navegação inercial, e os compartilham enviando mensagens com informações sobre sua identidade, posição, direção, altitude, vetor de velocidade e variações verticais. A Figura 6 ilustra como é feita a troca de informações entre aeronaves e estações terrestres equipadas com ADS-B.

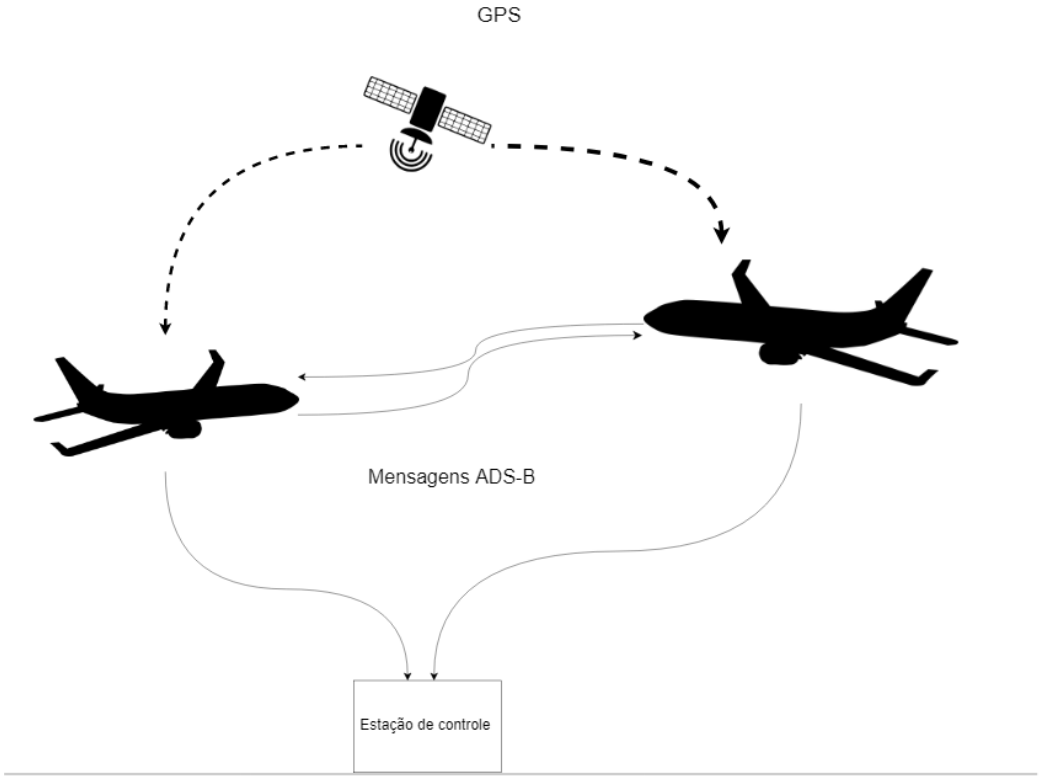

Figura 6: Funcionamento do ADS-B. Adaptado de (ICAO, 2013, p. 1–61)**.**

O envio de mensagens do ADS-B é feito por meio de um *transponder*[6] modo S[7]. Por meio dele as mensagens são transmitidas, em média, a cada 0,5 segundo, utilizando a frequência de 1.090 MHz. O alcance dessa transmissão varia conforme a capacidade da antena, porém encontra-se, geralmente, por volta de 270 a 320 km. O protocolo de envio da mensagem tem um formato de 112 *bits*, dos quais apenas 56 *bits* são utilizados para armazenar o conteúdo da informação, enquanto os demais *bits* são utilizados para indicar o tipo de mensagem (5 *bits*), a capacidade de comunicação do *transponder* (3 *bits*), a identificação da aeronave (24 *bits*) e para armazenar o *check* de paridade (24 *bits*), como apresentado na Figura 7 (McCallie et al., 2011, p. 78–87).

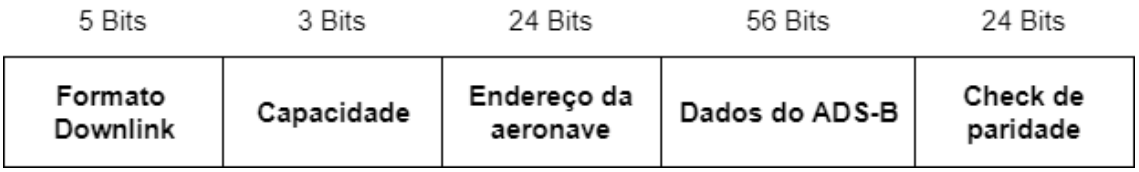

Figura 7: Protocolo ADS-B. Adaptado de (McCallie et al., 2011, p. 78–87)**.**

A informação passada na mensagem (Dados do ADS-B) contém: a) dados sobre a posição da aeronave: latitude, longitude, altura barométrica, altura geográfica; b)

[6] *Transponder*: Transmissor-receptor de radar secundário de bordo que recebe automaticamente sinais de rádio dos interrogadores de solo e que, seletivamente, responde, com um pulso ou grupo de pulsos, somente àquelas interrogações realizadas no modo e código para os quais estiver ajustado (DECEA, https://www.decea.mil.br/index.cfm?i=utilidades&p=glossario&single=2360, acessado em 16/01/2023).

[7] Modo S: Modelo de comunicação que permite envio de mensagens seletivas, tem um *link* bidirecional e utiliza um endereçamento de 24 *bits* (ICAO, 2013, p. 1–61).

dados sobre o deslocamento da aeronave: ângulo *heading*[8], ângulo vertical e velocidade; c) informação sobre se a aeronave está ou não em solo (Sun, 2017).

3.1.2. *Traffic alert and Collision Avoidance System* (TCAS)

O TCAS (*Traffic alert and Collision Avoidance System*) é um sistema criado com o intuito de prevenir colisões entre aeronaves. Ele permite a uma aeronave identificar outras ao seu redor, verificando suas rotas e analisando a existência de risco de uma possível colisão entre elas. Caso o risco de colisão seja confirmado, o TCAS sugere ao piloto uma manobra para evitar o conflito, com ajuste na rota da aeronave.

Assim como no ADS-B, a detecção pelo TCAS ocorre por meio do envio de informações pelas aeronaves ou por um sistema de vigilância; porém, diferente dele, no TCAS ocorre uma comunicação entre os dispositivos equipados nas aeronaves, nos quais um responde à chamada de outro, permitindo, entre outras coisas, coordenar as manobras automáticas de resolução de conflito.

O TCAS também se utiliza de um *transponder* para realizar a comunicação com outro TCAS, sendo compatível com os modelos ATCRBS ou Mode S (como usado pelo ADS-B). Ele envia mensagens de interrogações com intervalo de 1 segundo, na frequência de 1.030MHz. As mensagens, quando lidas por aeronaves próximas, são respondidas com frequência de 1.090MHz. Por também utilizar o Mode S, ele contém um protocolo de comunicação semelhante ao encontrado no ADS-B. Seu alcance chega a 30 nmi[9]. Pelas normas da ICAO, seu limite de rastreamento deve obrigatoriamente ser maior que 30 aeronaves, com uma densidade acima do que 0,3

[8] *Heading*: Direção para a qual a proa, ou ponta do nariz da aeronave, está apontada. Ele normalmente é calculado com base nas direções norte-oeste e leste-sul.

[9] nmi: 1 milha náutica corresponde a 1,852 quilômetros.

aeronaves por milha náutica quadrada (FAA - Federal Aviation Administration, 2011, p. 1–50). A Figura 8 ilustra a arquitetura do TCAS.

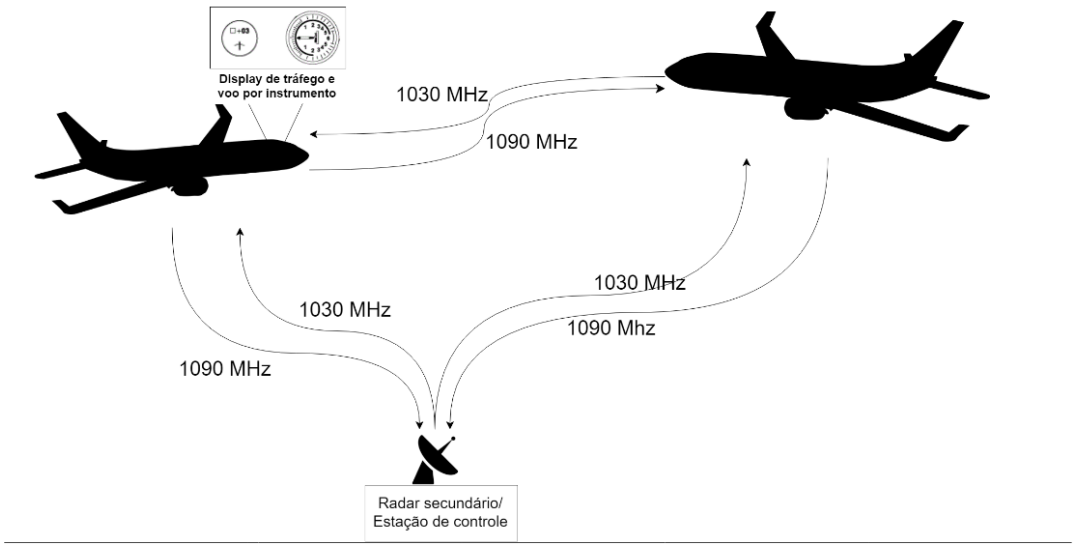

Figura 8: Arquitetura do TCAS (FAA - Federal Aviation Administration, 2011, p. 1–50).

A partir dos dados recebidos da outra aeronave detectada, o TCAS a classifica em um dos seguintes níveis discretos: a) *Other Traffic*, que inclui aeronaves distantes (maior do que 6 nmi na horizontal e 1200 ft na vertical), que não representam risco de colisão; b) *Proximate Traffic*, que inclui aeronaves que não representam risco de colisão, porém estão a uma distância menor do que 6 nmi na horizontal e 1200 ft na vertical; c) *Traffic Advisory* (TA), que representa as outras aeronaves, em área de alerta, em risco de colisão com a aeronave entre 20 a 48 segundos (conforme a altitude); d) *Resolution Advisory* (RA), que representa as outras aeronaves, em área de perigo, correspondente as outras aeronaves em risco eminente de colisão com a aeronave, entre 15 a 35 segundos (conforme a altitude); e) Área de Colisão, que representa a área de colisão eminente, na qual o tempo de colisão é menor do que 15 segundos. A Figura 9 ilustra estas últimas três áreas citadas.

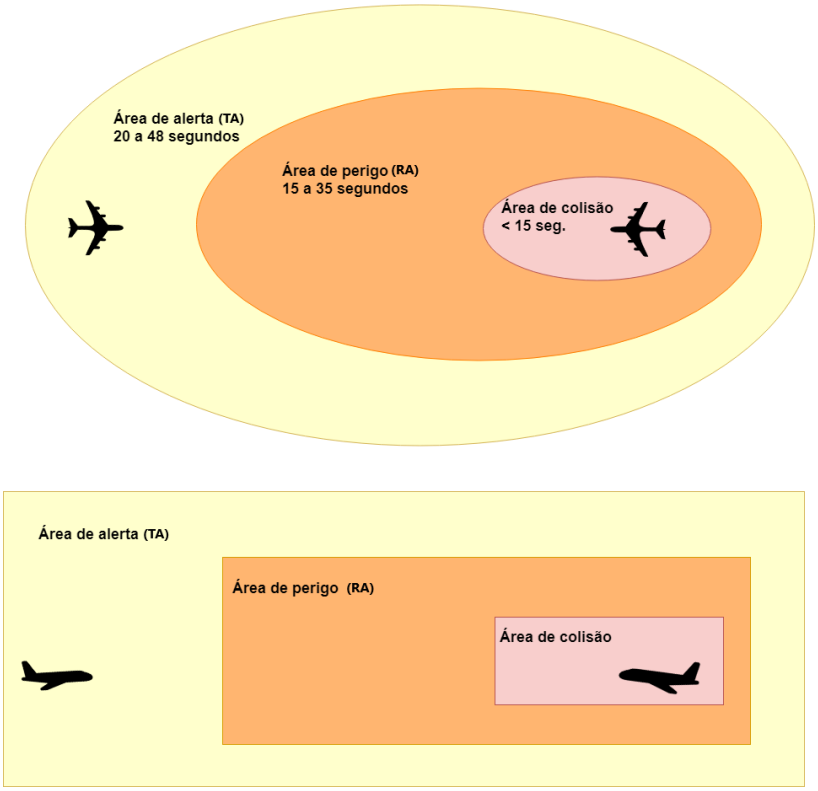

Figura 9: Áreas do TCAS: a) primeira imagem – vista superior; b) segunda imagem – vista lateral. Adaptado de (FAA - Federal Aviation Administration, 2011, p. 1–50)**.**

A localização das aeronaves “intrusas” é mostrada ao piloto por meio do *traffic display*, sendo que para isso são utilizados diferentes símbolos e cores, conforme sua classificação, segundo os níveis descritos. Em casos de risco iminente de colisão, também ocorre um aviso sonoro, avisando sobre o tipo de risco (TA ou RA), sugerindo uma manobra de resolução do conflito (Eurocontrol, 2017).

3.2. Sensores não cooperativos

Devido ao fato de a complexidade de se identificar a posição de uma aeronave ser maior do que a de receber dados do seu posicionamento, os sensores não cooperativos geralmente apresentam taxas de falhas maiores do que as dos sensores cooperativos, e por esse motivo constituem-se em uma área de estudo menos consolidada em aplicações no espaço aéreo. Muitos dos estudos sobre este tipo de

sensor estão centrados em se criar técnicas que permitam a eles alcançar probabilidade de falhas similares às dos sensores cooperativos.

Há uma grande variedade de sensores não cooperativos, sendo que eles também estão divididos em dois grandes grupos: os sensores passivos e os sensores ativos. Os sensores passivos são aqueles que leem dados oriundos do ambiente à sua volta sem necessitar emitir nenhum tipo de sinal, como por exemplo as câmeras, enquanto que os sensores ativos são aqueles que leem dados a partir da reflexão ou refração de sinais enviados pelo próprio sensor. Podem ser citados como exemplo de sensores ativos os radares, que emitem ondas eletromagnéticas e interpretam os tempos de reflexão das mesmas (Ramasamy et al., 2014, p. 271–76). A Figura 10 apresenta a diferença entre estes dois tipos de sensores: ativos e passivos.

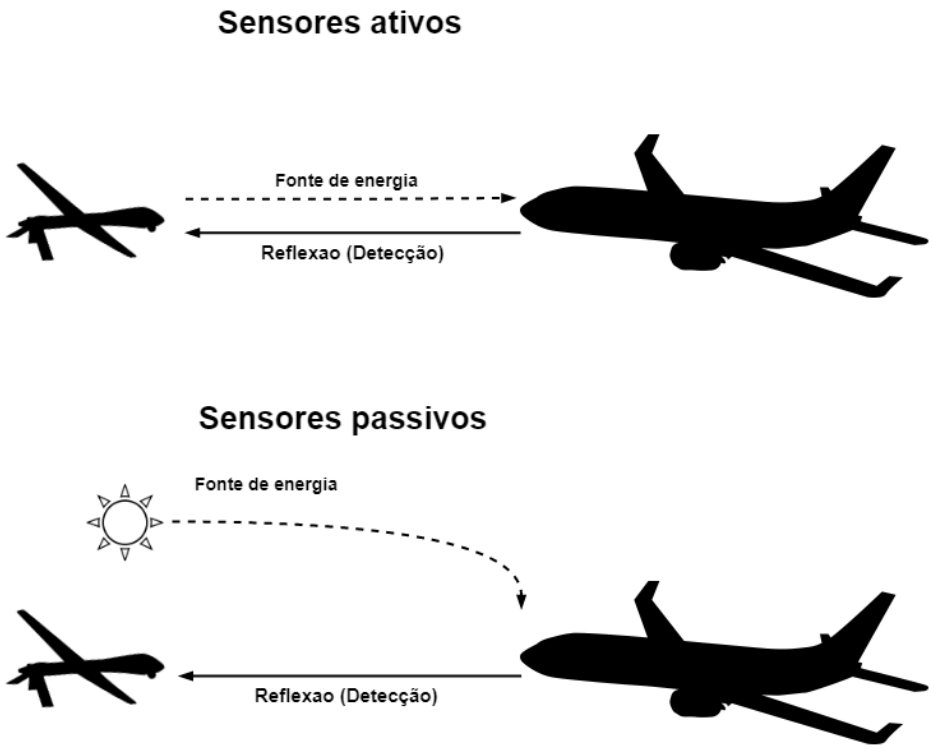

Figura 10: Sensores ativos e passivos. Adaptado de (Shakhatreh et al., 2018, p. 1–58)**.**

Entre os sensores ativos estudados neste trabalho estão os radares, mais especificamente o MMW Radar (*Millimetre Wave Radar*) e o LIDAR (*Light Detection*

And Ranging). Por outro lado, no grupo dos sensores passivos encontram-se as câmeras, que podem captar várias frequências diferentes do espectro de luz. Neste trabalho de pesquisa são abordadas as câmeras visuais, que capturam o espectro de luz visível, as câmeras termais e as câmeras de profundidade. Todos esses sensores são descritos em detalhes na sequência.

3.2.1. MMW Radar

O MMW Radar (*Millimetre Wave Radar*) é um radar de altas frequências de rádio e, dentre elas, podem ser citadas as bandas Ka, V, W e G. Os radares deste tipo são disponíveis em vários modelos, com diversos tamanhos, acurácias e campos de visão, funcionando por meio da emissão de uma onda direcionada e reconhecendo objetos por meio do cálculo do tempo de reflexão da onda emitida. Deste modo, um MMW Radar retorna dados contendo a distância dos objetos encontrados em uma única direção, ou seja, uma lista com a distância dos objetos próximos. A Figura 11 ilustra o funcionamento deste tipo de radar.

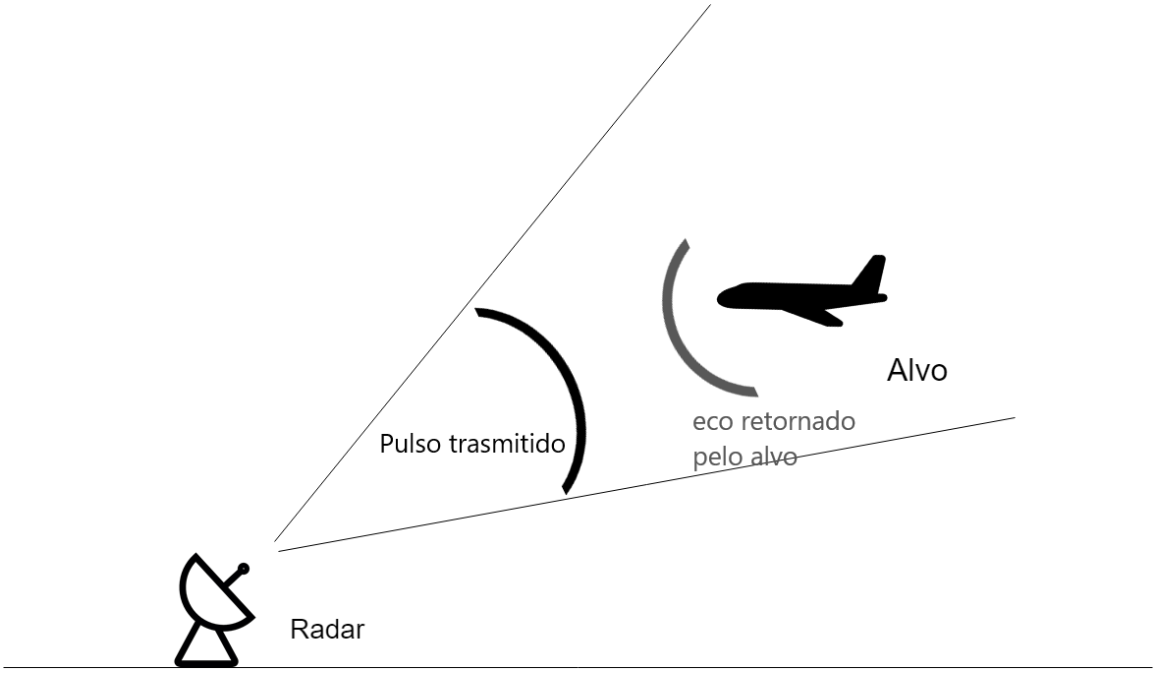

Figura 11: Ilustração de detecção por Radar.

3.2.2. LIDAR (*Light Detection And Ranging*)

O LIDAR (*Light Detection And Ranging*) corresponde a um dos sensores mais promissores para a utilização em VANTs, devido à sua grande acurácia e ao seu tamanho reduzido. Este sensor faz uso de um *laser* infravermelho pulsado para obter informações sobre as distâncias dos objetos sendo rastreados, a partir do cálculo da diferença entre os instantes de emissão e de detecção do pulso de *laser* emitido (Fasano et al., 2014, p. 430–40).

O LIDAR retorna dados em 3 dimensões, como uma nuvem de pontos (*cloud point*), ou seja, uma coleção de pontos, sendo que cada ponto tem valor distinto nos eixos *x*, *y* e *z* (vide Figura 12(a) e Figura 12(b)). Uma única coleta do LIDAR pode conter dados lidos oriundos de vários pulsos de *laser* enviados em instantes de tempo diferentes (como apresentado na Figura 12(c)). A partir desde valores é possível aplicar algoritmos que permitam identificar objetos considerando seus formatos e distâncias. Seu campo de visão pode variar tanto horizontal quanto verticalmente, podendo-se ter sensores que alcançam ângulos de 360º em uma das direções (horizontal ou vertical).

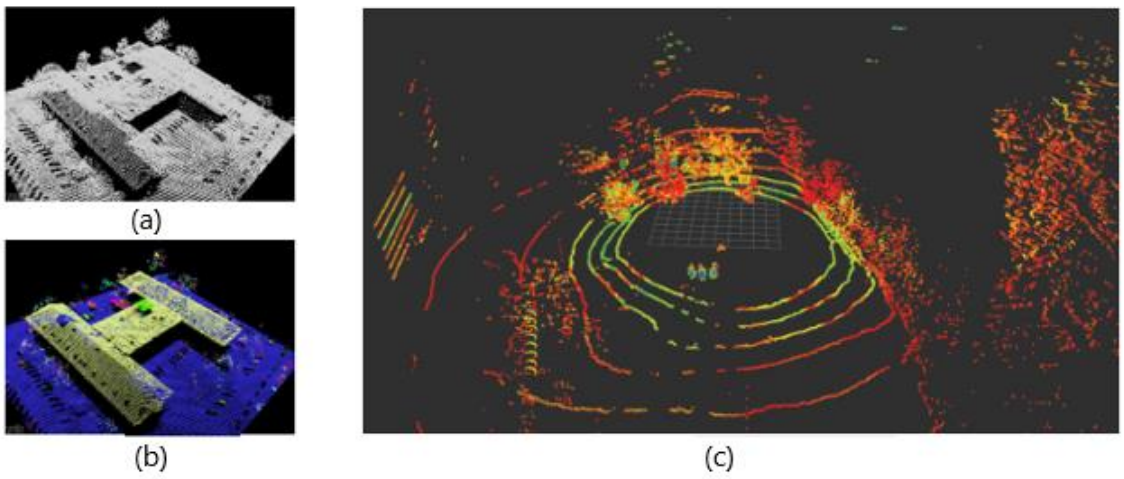

Figura 12: Exemplos de uso do LIDAR. Nas figuras (a) e (b), o LIDAR sendo utilizado para detecção do telhado de uma construção, sendo que em (a) é mostrada a nuvem de pontos original, enquanto que em (b) é mostrada a detecção resultante (Verma et al., 2006, p. 2213–20). **A figura (c) representa um *snapshot* de um LIDAR com ângulo de visão 360º na horizontal** (De Haag et al., 2016, p. 1–11).

Em (Ramasamy et al., 2016, p. 344–58) apresenta-se a modelagem matemática de um LIDAR. As fórmulas apresentadas são utilizadas tanto em um simulador como em um VANT real. Nesse trabalho é apresentado o cálculo da probabilidade de falso alarme, conforme Equação 1 (P_{fa}), e a probabilidade cumulativa de detecção, de acordo com a Equação 2 (P_D). Estas fórmulas permitem calcular a probabilidade de falhas de um LIDAR.

Na Equação 1, B é a largura de banda do receptor, T_{fa} é o tempo médio entre falsos alarmes e η é o máximo intervalo útil. Já na Equação 2, M é o número máximo de possíveis detecções, m é o número mínimo de possíveis detecções e P_d é a probabilidade de detecção.

$$P_{fa} = \frac{1}{B \cdot T_{fa} \cdot \eta}$$

Equação 1: Probabilidade de falso alarme no LIDAR (Ramasamy et al., 2016, p. 344–58)**.**

$$P_D = 1 - \sum_{i=0}^{m} C_M^i P_d^i \left(1 - P_d\right)^{M-i}$$

Equação 2: Probabilidade cumulativa de detecção no LIDAR
(Ramasamy et al., 2016, p. 344–58)**.**

3.2.3. Câmeras

Diferentemente dos demais sensores, as câmeras não foram criadas com o intuito de detecção ou rastreamento de objetos, sendo seu objetivo principal a gravação de imagens de vídeos por meio da captura da luz do ambiente em sua volta. Porém, com

o uso de algoritmos de visão computacional, as câmeras vêm ganhando diversas funções que vão além da tarefa de gravação de imagens de vídeo. Algoritmos como o TLD (*Tracking-Learning-Detection*) (Yang et al., 2011, p. 3823–31) tornam possível a utilização das imagens capturadas pela câmera para rastrear um objeto, podendo-se verificar as posições desse objeto em diferentes *frames* do vídeo, o que permite, assim, a estimativa da direção e velocidade do objeto sendo rastreado.

Com estes algoritmos, as câmeras têm sido popularizadas na área de detecção e rastreamento de objetos, tendo como grande vantagem o fato de geralmente serem sensores pequenos, leves, baratos e com baixo gasto energético, o que as tornam ideais para serem embarcadas em pequenos VANTs, que têm limitações de peso e de consumo de energia. Além disso, esse tipo de equipamento pode ter a capacidade de capturar imagens em diferentes campos ou ângulos de visão[10].

Diferentes formas de captura de dados permitem às câmeras capturar vários comprimentos de onda do espectro de luz. Os comprimentos de onda mais comuns utilizados para a detecção de objetos são os relacionados com a luz visível (com comprimento de onda entre 400nm e 700nm), encontradas em câmeras RGB (padrão de cores *red*, *green*, *blue*), ou então os referentes às frequências de luz infravermelha (com comprimento de onda acima de 700nm), geralmente utilizada por câmeras termais (entre as frequências 9μm a 14μm), pois nesta frequência é possível capturar o calor emitido por um objeto. A Figura 13 mostra as frequências de luz capturadas por diferentes tipos de câmera.

[10] Ângulo de visão: Extensão angular capturada pela câmera.

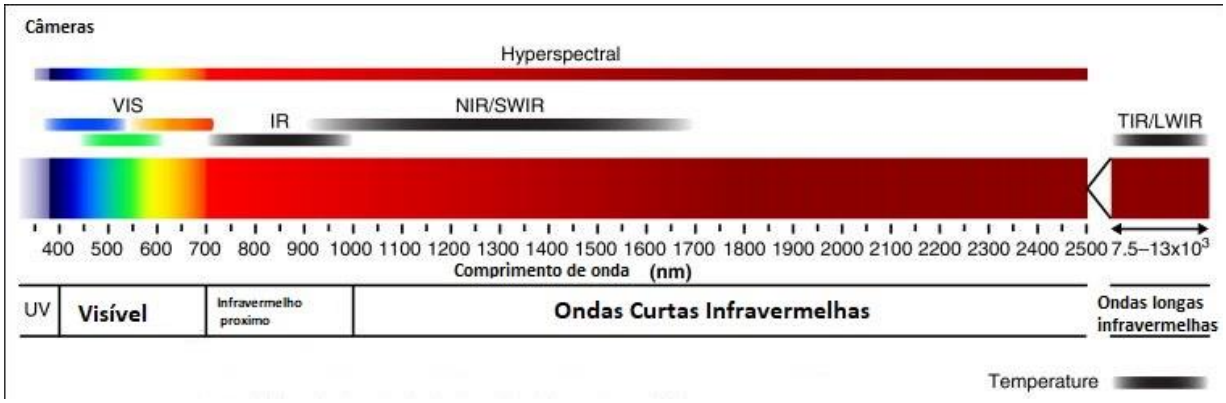

Figura 13: Espectro da luz visível[11]: A parte superior da imagem ilustra as bandas do espectro capturadas por diferentes tipos de câmeras: VIS (visível ou RGB), IR (infravermelha) e NIR/SWIR (infravermelhos de ondas curtas).

Para se fazer uma análise de confiabilidade de câmeras utilizadas como sensor foi tomado por base o trabalho proposto em (Fields and Sands, p. 1–8, 2010), no qual encontra-se a análise da probabilidade de falhas de diversas câmeras, com diferentes níveis de qualidade e custos, e por diferentes períodos de tempo. Nesse trabalho é apresentada a probabilidade de falhas média de câmeras com alta qualidade, da ordem de 2,4% no período de um ano.

Porém, a qualidade na detecção e rastreamento de objetos utilizando-se uma câmera com sensor se deve principalmente à qualidade do algoritmo de visão computacional utilizado pelo sistema na qual ela está inserida. Esta é uma área de pesquisa recente que vem sendo aprimorada com novos algoritmos anualmente, cada um com suas vantagens e desvantagens. Além disso, câmeras com capturas de diferentes espectros de luz podem ter resultados diferentes em cada um desses algoritmos, sendo então necessário avaliar cada par câmera/algoritmo separadamente. Desse modo, para se encontrar a técnica que melhor se adapte ao sistema estudado, devem

[11] Disponível em < https://www.researchgate.net/figure/fig4_272889618>, acessado 11/2018

ser realizados testes com esses diferentes algoritmos para cada tipo de câmera, incluindo os presentes no atual estado da arte em detecção e rastreamento de objetos.

Neste trabalho de pesquisa são realizados testes com diversos algoritmos de visão, apresentados no capítulo 7.

4. Algoritmos e estratégias utilizados na detecção de objetos

Os sensores são elementos fundamentais no processo de resolução de conflitos, pois por meio deles pode-se “sentir” objetos ao redor do veículo aéreo que está sendo controlado. Porém, somente os sensores não são suficientes para identificar um objeto em risco de colisão com a aeronave em questão, pois eles são apenas uma forma de entender o ambiente à volta da aeronave e de transformar esse entendimento em dados capazes de serem processados.

Cabe aos algoritmos a análise destes dados, utilizando técnicas para processar as informações capturadas pelos sensores. Os algoritmos de detecção reconhecem a posição dos objetos que eles capturam, ou utilizam estratégias para fundir os dados de vários sensores em uma detecção única e mais precisa, na qual cada sensor pode cobrir a falha do outro. Além disso, os algoritmos também calculam o risco de colisão da aeronave com outros objetos e planejam uma nova rota que faça com que o veículo controlado possa desviar do objeto em risco de colisão, de modo a interferir o mínimo possível em sua trajetória original.

Neste capítulo são apresentados em detalhes diversos algoritmos capazes de serem utilizados durante o processo de *sense and avoid*, conforme suas funções. Algum destes algoritmos são utilizados nos testes apresentados neste trabalho.

4.1. Algoritmos de detecção

Devido às diferenças entre as tecnologias utilizadas nos diversos tipos de sensores, cada sensor tem sua maneira específica de retornar seus dados, apresentando diferente detalhes do ambiente observado. Desse modo, para cada tipo de sensor devem ser criados tipos específicos de algoritmos. Por este motivo eles são aqui apresentados a partir do sensor no qual são utilizados e, para compreendê-los, são mostrados em detalhes os dados que o sensor retorna e como é o processo de se retirar informação a partir desses dados.

4.1.1. Detecção de objetos em imagem 2D

As câmeras visuais são dispositivos que capturam o espectro de luz visível. Essa luz é capturada pelo sensor e transformada em uma imagem digital de duas dimensões (2D), frequentemente representada por um conjunto de três matrizes, em que cada matriz representa uma banda diferente do espectro de luz visível. Geralmente as bandas utilizadas são a vermelha (*red*), a verde (*green*) e a azul (*blue*), sendo que a fusão delas gera uma imagem colorida, conforme ilustrado na Figura 14. Cada ponto desta matriz guarda a intensidade de luz de uma banda específica (vide Figura 14), chamado de *pixel*, e o número de *pixels* na matriz varia conforme a resolução da imagem (Tekalp, 2015).

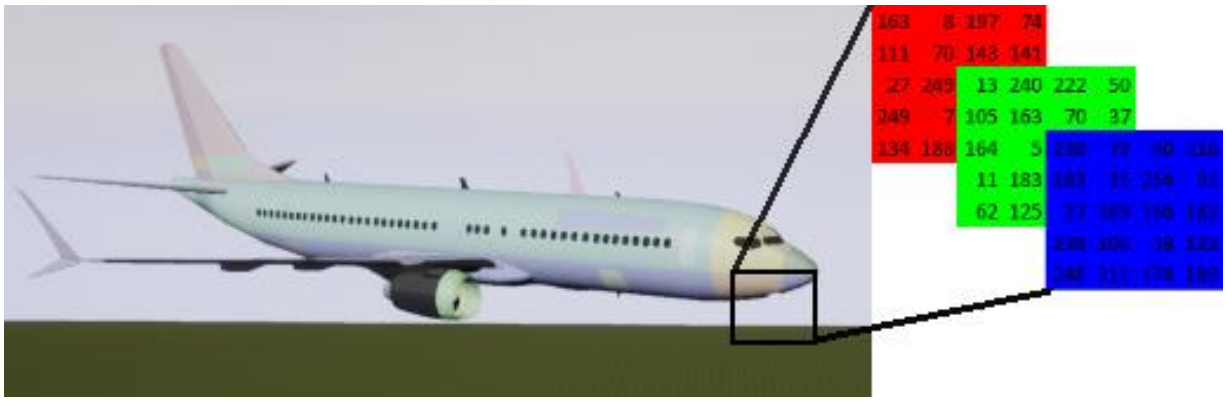

Figura 14: Representação de uma imagem por meio de matrizes de cores [Próprio].

Para possibilitar a detecção de objetos, algumas abordagens analisam separadamente cada imagem. Utilizando-se de algoritmos de detecção, extrai-se características da imagem, retirando o máximo de informações presentes nela, com o objetivo de se reconhecer padrões que diferenciam o objeto alvo dos demais elementos presentes na imagem. A extração de características da imagem pode ocorrer de diversas maneiras e, dentre elas, podem ser citadas:

- Dados sobre a distribuição de cores da imagem, alcançados por meio do uso de histogramas;

- Informações da textura da imagem, como por exemplo, os obtidos com o uso do algoritmo LBP (*Local Binary Pattern*) (Guo et al., 2010, p. 1657–63);
- Caraterísticas dos gradientes da imagem, podendo se utilizar o algoritmo HOG (*Histogram of Oriented Gradients*) ou o algoritmo SIFT (*Scale-Invariant Feature Transform*);
- Dados sobre o relacionamento entre os *pixels* da imagem, alcançados com o uso de filtros convolucionais, como, por exemplo, os presentes nos algoritmos CNN (*Convolutional Neural Network*) (Yang et al., 2011, p. 3823–31).

Após a extração das características da imagem, os algoritmos a classificam como contendo ou não o objeto alvo. Uma abordagem muito utilizada com este objetivo são os algoritmos de aprendizado de máquina. Estes algoritmos “aprendem” a reconhecer padrões por meio de dados, que são utilizados para otimizar parâmetros de uma função de classificação. Dentre os esses tipos de algoritmos, destacam-se:

***Support Vector Machine* (SVM)**

O SVM é um algoritmo de aprendizado supervisionado no qual, a partir de um conjunto de dados pré-rotulados, gera-se um classificador binário linear, podendo ser utilizado para classificar uma imagem como contendo ou não contendo o objeto alvo. Seu processo utiliza os dados de entrada como vetores n-dimensionais e, a partir do rótulo (ou da classe) de cada dado ele otimiza uma função de um hiperplano, de modo que ela separe os dados com a maior distância possível entre as classes. O SVM alcança bons valores de classificação, porém devido à grande complexidade para se calcular o hiperplano, ele pode ser inviável de ser utilizado em VANTs por exigir alto poder computacional (Russell and Norvig, 2010).

Naive Bayes

O *Naive Bayes* é um algoritmo de classificação probabilístico, baseado na aplicação do algoritmo no teorema de *Bayes*. Ele calcula a probabilidade de um determinado conjunto de dados ser rotulado como pertencente ou não a uma classe, a partir de a

probabilidade *a priori* de cada dado individual ser rotulado como pertencente a mesma classe.

Algoritmos baseados em Árvores

Este conjunto de algoritmos cria uma estrutura em árvore, capaz de classificar os dados a partir de uma série de passos. Eles têm como sua grande vantagem o fato de serem rápidos e precisos. Destacam-se nesta categoria:

- **Árvores de Decisão:** Este é o algoritmo base para os outros deste conjunto. Ele se utiliza de métricas matemáticas, como Entropia, para calcular as características que melhor separam as classes dos dados. A partir dessas características é criada uma estrutura hierárquica para se classificar um novo dado (Russell and Norvig, 2010);
- ***Random Forest*:** Neste tipo de algoritmo utiliza-se a ideia de que, para um mesmo conjunto de dados, várias árvores de decisão podem ser criadas, gerando resultados similares. Seu conceito baseia-se na ideia de se criar o máximo de árvores possíveis para classificar os dados, e seu resultado é dado por meio de uma votação dos resultados destas árvores (BREIMAN, 2020, p. 5–32);
- ***Light Gradient Boosting* (*LigthGBM*):** Esta técnica utiliza várias árvores de decisão encadeadas, sendo que a primeira árvore é utilizada para calcular a detecção do objeto, enquanto que as demais são treinadas visando corrigir eventuais erros da primeira árvore. Assim, em cada nova árvore encadeada, ocorre uma melhora na classificação da árvore inicial. Esta técnica foi desenvolvida inicialmente pela Microsoft (Ke et al., 2017, p. 1–9).

Redes Neurais

Redes neurais são algoritmos inspirados no modo como ocorre o aprendizado no cérebro humano, tendo como base o algoritmo *Percetron* (Rosenblatt, 1958, p. 386–408) o qual, simulando um neurônio humano, cria um classificador linear. Unindo-se várias execuções em cadeia do algoritmo *Percentron* é possível criar uma estrutura

similar a uma rede, em um algoritmo chamado de *Percetron* Multicamadas ou de Redes Neurais.

Deep Learning

Nos últimos anos, os algoritmos baseados em redes neurais demonstraram ótimos resultados em problemas de classificação, e isso incentivou um grande aumento nas pesquisas utilizando esses algoritmos, criando-se redes mais complexas e com maior número de camadas, chamadas de *Deep Learning*. Dentre elas, destacam-se algumas classes de algoritmos específicos para o processo de detecção de objetos em imagens:

- **Redes Neurais Convolucionais:** Este é um modelo de redes neurais que foi criado especificamente para trabalhar com problemas envolvendo imagem. Seu princípio tem como base a otimização de filtros de convolução, ou seja, uma classe de filtros aplicados à imagem pode extrair informações das relações entre os *pixels* da imagem. Dentre os vários algoritmos utilizados neste modelo de rede neural vale destacar o ImageNet (Krizhevsky et al., 2012, p. 1097–105) e o Xception (Chollet, 2017, p. 1251–58);
- **YOLO *(You Only Look Once)*:** Esta é uma classe de rede neural otimizada para detecção e rastreamento de objetos em imagem. Sua grande vantagem é o fato de necessitar observar a imagem somente uma vez para identificar a posição, classe e tamanho do objeto. Isso se dá na maneira pela qual este algoritmo retorna seus resultados: enquanto a maioria dos demais algoritmos apenas retorna a classe à qual a imagem pertence, este algoritmo retorna a classe do objeto ao mesmo tempo em que retorna o seu *bounding box* (caixa delimitadora). Além disso, pode retornar até várias detecções de objetos de uma única vez na mesma imagem (Redmon and Farhadi, 2018).

Janelas deslizantes

Para tornar o processo de captura e detecção de informação na imagem mais acurada, estes algoritmos são aplicados separadamente em pequenas amostras da

imagem. Estas técnicas têm duas grandes vantagens. A primeira se dá pelo fato de que uma pequena amostra da imagem contém uma quantidade menor de dados e, assim, permite que o algoritmo de aprendizado se especialize de maneira mais precisa e mais eficiente na classificação desta amostra em comparação com a imagem completa, que contém uma quantidade maior de dados. A segunda vantagem se dá pelo fato de que, ao se classificar uma amostra como aquela que contém o objeto alvo, tem-se também as informações sobre o posicionamento deste objeto na imagem, retirados dos dados de posicionamento da amostra na imagem original.

O algoritmo mais utilizado para a criação destas amostras é o **algoritmo de janelas deslizantes**. Neste algoritmo aplica-se uma janela de tamanho $\hat{n} \times \hat{n}$ em uma imagem de tamanho $n \times n$, em que $\hat{n}$ é menor do que n. Esta janela retira uma amostra da imagem que é então processada por um algoritmo de detecção, separadamente do resto da imagem. Após isso, ela é “deslizada”, em relação à sua última posição, para uma posição deslocada de x nos eixos horizontal e vertical da imagem, em que x é o valor de deslocamento, menor ou igual a $\hat{n}$. O processo é então refeito, até que toda a imagem tenha sido processada em pequenas janelas (Forsyth and Ponce, 2011). Essas janelas podem, eventualmente, conter sobreposições, como ilustrado na Figura 15. Após se processar todas as janelas, o algoritmo terá uma lista de janelas nas quais foram detectados o objeto alvo. Esta lista pode conter janelas que se sobrepõem, ou janelas isoladas. Visando aumentar a precisão da detecção, é comum considerar que um objeto somente é classificado como ‘detectado’ caso ele seja encontrado em um número mínimo de janelas próximas, e este número pode ser chamado de limiar (*threshold*). Para identificar estas janelas próximas é aplicada uma técnica de pós-processamento, utilizando algoritmos de “clusterização”.

A Figura 15 ilustra este processo, no qual é aplicado um algoritmo de detecção de aeronaves utilizando a técnica de janelas deslizantes, que retira pequenas amostras de toda imagem. Nessas amostras, seis janelas são classificadas como pertencentes à uma aeronave, ou seja, foram classificadas como contendo parte do objeto alvo, das quais cinco se sobrepõem, podendo ser considerado um grupo, enquanto que uma outra sexta janela está isolada das demais, e será descartada. Caso seja adotado o

critério de se considerar um mínimo de três janelas próximas para se classificar um objeto como sendo detectado, tem-se então apenas uma detecção, referente ao agrupamento de cinco janelas.

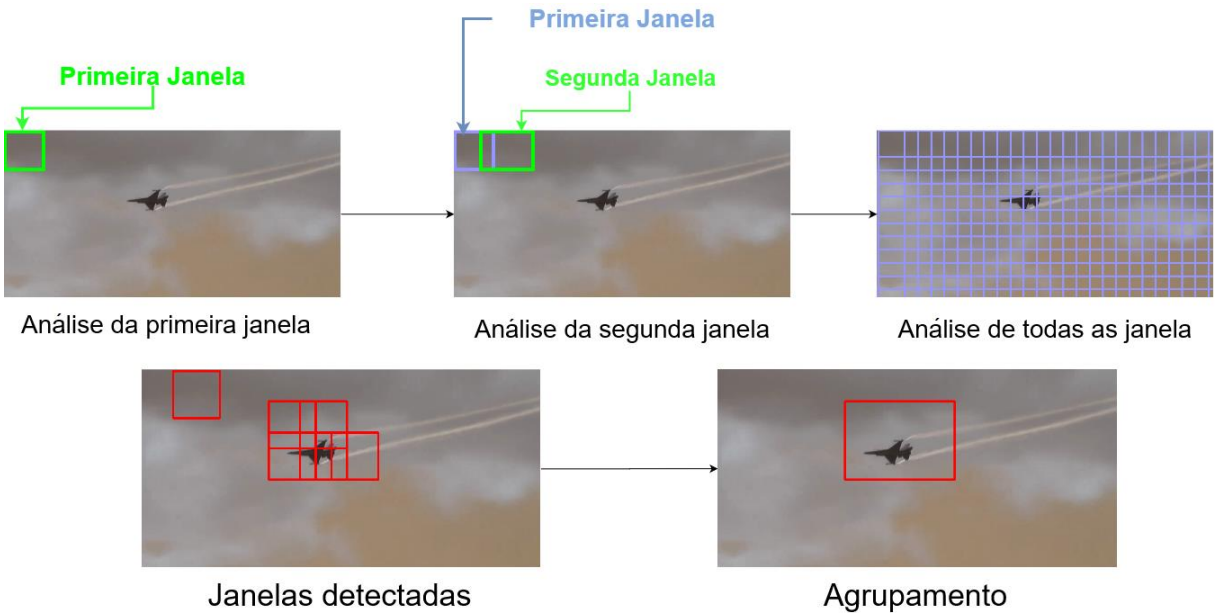

Figura 15: Processo de detecção de um objeto alvo utilizando janelas deslizantes [Próprio].

4.1.2. Detecção de objetos em imagem 3D

As abordagens de detecção apresentadas são eficientes na detecção de um objeto em uma imagem 2D, que é uma projeção em duas dimensões do mundo real (em 3 dimensões – 3D). Apesar desta detecção permitir descobrir a direção do objeto alvo, ela não é capaz de calcular a distância entre o objeto e o observador, pois não contém a dimensão de profundidade.

Para ser possível calcular essa distância é necessário que as imagens utilizadas sejam capazes de gerar dados em 3D. Estes dados podem ser alcançados de duas maneiras: por meio de um sensor que capte imagem em 3D, por exemplo, as câmeras de profundidade, ou utilizando um conjunto de dois ou mais sensores normais, que é uma técnica conhecida como visão binocular, também chamada de visão estéreo.

Câmeras de profundidade

Câmeras de profundidade são sensores capazes de capturar imagem em 3D. Eles podem utilizar diferentes técnicas para capturar a profundidade da imagem, como *lasers* ou luz infravermelha, o que torna este tipo de abordagem mais cara do que as que se utilizam de câmeras tradicionais.

Seus dados frequentemente são retornados como uma matriz, similar à retornada pelas câmeras visuais, porém, neste caso, com apenas uma dimensão, na qual o valor de cada *pixel* significa a distância do objeto capturado em relação a câmera. A imagem da Figura 16 mostra a captura de uma aeronave em uma câmera de profundidade.

Devido ao fato de seus dados terem um formato similar ao apresentado nas câmeras visuais, os mesmos algoritmos de detecção aplicados a elas podem ser aplicados aos dados retornados por câmeras de profundidade, desde de que ambos tenham etapas de treinamentos individuais, para melhor performance do algoritmo.

Figura 16: Aeronave detectada por uma imagem de profundidade. Na imagem, quanto mais claro o *pixel*, mais distante está o objeto capturado. [Próprio]

Visão binocular

A visão binocular, também chamada de visão estéreo, ocorre quando uma imagem é capturada por dois ou mais sensores, permitindo, assim, o cálculo da profundidade da imagem. Este tipo de visão é inspirado na visão humana, na qual a imagem é capturada por intermédio de um par de olhos. Com esta técnica é possível calcular um mapa de profundidade, similar à imagem retornada pela câmera de profundidade, a partir de duas câmeras visuais simples.

Esta técnica utiliza princípios geométricos aplicados ao modelo de formação da imagem. Para entender melhor este princípio é necessário apresentar uma introdução sobre o princípio de formação de uma imagem. Uma imagem é a projeção em duas dimensões de uma cena no mundo real em três dimensões. Um dos modelos matemáticos que melhor descreve essa projeção é o modelo da câmera *pinhole*, no qual a câmera é comparada a uma câmara escura, que permite que a luz passe apenas por um pequeno orifício (*pinhole*), formando uma projeção invertida da imagem no fundo da câmara. Esse processo é representado na Figura 17, sendo a

distância entre o pequeno orifício e a projeção chamada de distância focal, representada pela letra f. (Forsyth and Ponce, 2011) (Hartley and Zisserman, 2003). A imagem projetada tem o formato invertido em relação ao formato real da imagem capturada; pode-se criar uma imagem virtual desta imagem projetada, que tem a mesma orientação do objeto capturado é está posicionada na mesma distância *f* em relação ao *pinhole* da imagem projetada, mas no sentido oposto.

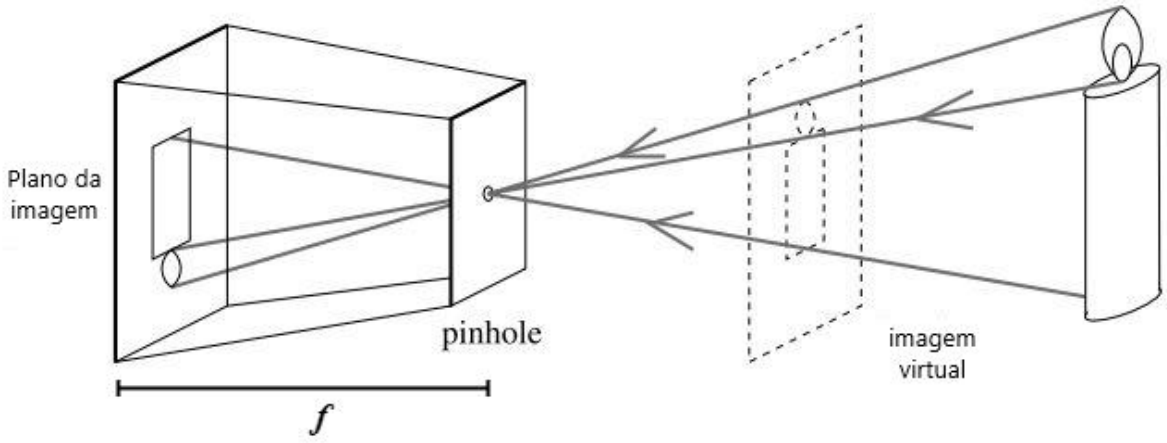

Figura 17: Formulação da imagem pelo modelo de câmera *pinhole*. Adaptado de (Forsyth and Ponce, 2011)

Neste modelo pode-se aplicar fórmulas geométricas e demonstrar matematicamente a formação da imagem. A Figura 18 apresenta a geometria de uma projeção de câmera *pinhole*, na qual o *drone* representa a câmera que está no centro (*0*, *0*, *0*) do seu próprio eixo de coordenadas (*x*, *y*, *z*), o objeto analisado (uma aeronave) está posicionado no ponto *P* = (*X*, *Y*, *Z*) no eixo de coordenadas do mundo real (representado em vermelho), e a imagem virtual projetada está representada na cor amarela, tendo o ponto principal (X_c, Y_c) em seu o centro, e a distância *x* em relação ao *drone*, igual a distância focal da câmera *f*.

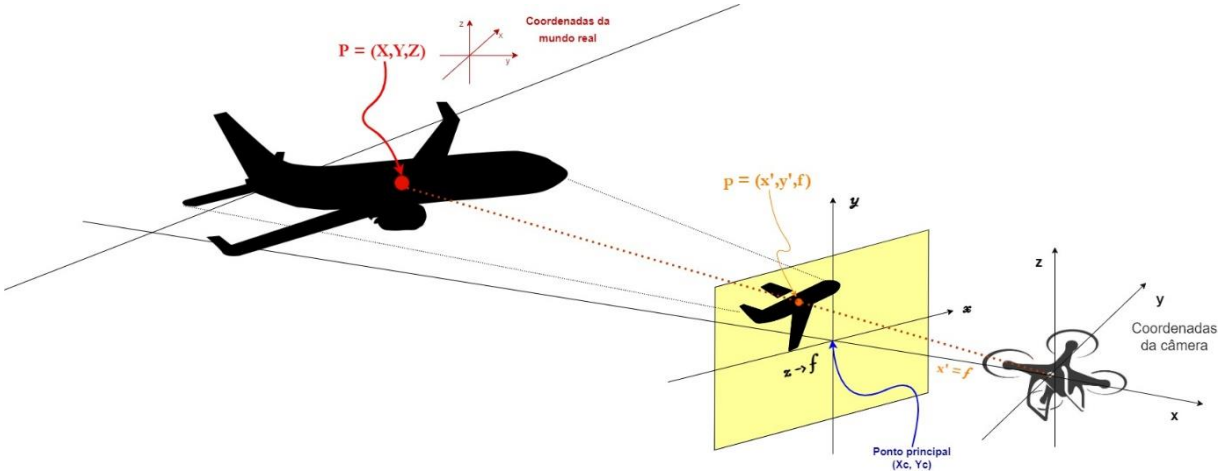

Figura 18: Geometria da formação de imagem em uma câmera *pinhole* [Próprio].

Com esta representação geométrica é possível calcular a posição em duas dimensões que um objeto terá na imagem projetada dado sua posição no mundo real. Para isso é necessário o conhecimento de parâmetros intrínsecos da câmera, que é sua distância focal, o valor do ponto principal (ponto que representa o centro da imagem), o conhecimento dos parâmetros extrínsecos da câmera (sua matriz de rotação e sua posição no eixo de coordenadas do mundo real) e da posição do objeto no mundo real, utilizando a fórmula de modelo linear da câmera, conforme expressão apresentada a seguir (Hartley and Zisserman, 2003).

$$X_{cam} = K\ [\,R \mid T\,]\ X$$

$$\begin{bmatrix} x_{cam} \\ y_{cam} \\ w_{cam} \end{bmatrix} = \begin{bmatrix} fx & 0 & Px \\ 0 & fy & Py \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r1 & r2 & r3 & t1 \\ r4 & r5 & r6 & t2 \\ r7 & r8 & r9 & t3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix}$$

Equação 3: Equações do modelo linear das câmeras.

A primeira equação acima é uma versão simplificada, em que *K* é a matriz intrínseca, *R* é a matriz de rotação do eixo de coordenadas da câmera, *T* é a matriz de translação da câmera e *X* é a posição do objeto no mundo real. A segunda equação é a equação completa, em que fx e fy são os valores da distância focal da câmera nos eixos *x* e *y*, respectivamente, os valores Px e Py são os valores do ponto principal da imagem

(o centro da imagem), $r1$ a $r9$ representam os valores da matriz de rotação da câmera, os valores de $t1$ a $t3$ são os valores da translação da câmera, [*X Y Z*] correspondem à posição do objeto, sendo que a rotação, a translação e a posição são em relação ao eixo de coordenadas do mundo real, e x_{cam}, y_{cam}, w_{cam} é a posição calculada do objeto identificado na imagem projetada.

Apesar de a fórmula calcular a posição do objeto na imagem projetada a partir da posição do objeto no mundo real, não é possível calcular o inverso, ou seja, a posição do objeto no mundo real dada a posição do objeto na imagem projetada. Tal fato ocorre, pois, invertendo-se a fórmula apresentada, tem-se uma equação de reta, sendo possível apenas calcular a direção do objeto no mundo real, não a sua posição. Isso se dá devido ao fato de, ao se observar uma imagem, perdem-se dados de profundidade, dados esses correspondentes às 3 dimensões.

A visão binocular tem o objetivo de contornar este problema, pois, ao se utilizar duas câmeras é possível traçar duas equações de reta, sendo que o encontro delas é a posição estimada do objeto alvo. Este cálculo é chamado de geometria epipolar, como apresentado na Figura 19.

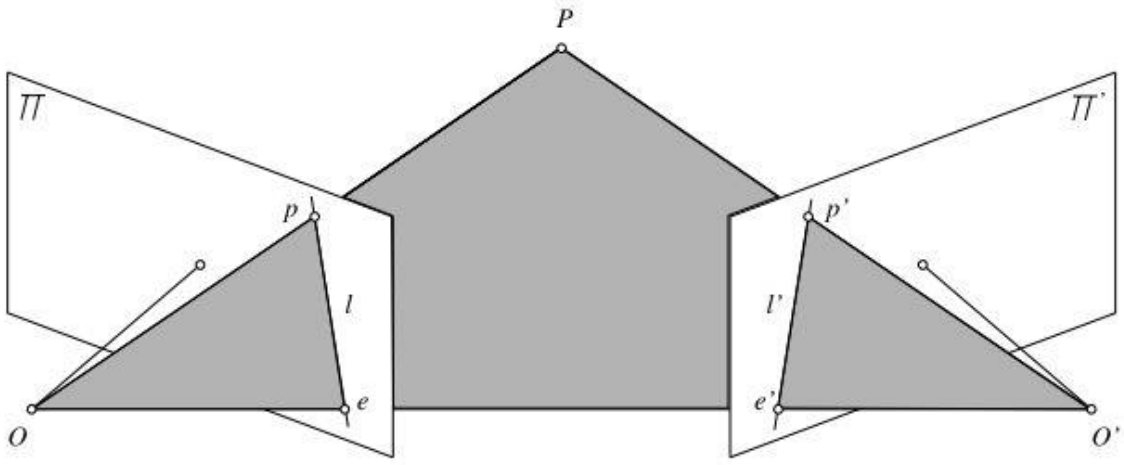

Figura 19: Geometria epipolar, em que *P* é a posição real do objeto, *p'* e *p* as projeções da imagem referentes às duas câmeras, e *O* e *O'* as câmeras da esquerda e da direita (Forsyth and Ponce, 2011)**.**

Com o uso de uma “visão estéreo”, a posição do objeto em três dimensões pode ser calculada com base nas fórmulas dadas a seguir, em que: x_1 e y_1 são os pontos do objeto na imagem à esquerda e x_2 e y_2 são os pontos do objeto na imagem à direita; p_x, e p_y são as coordenadas (x,y) do ponto principal nas imagens à esquerda e à direita (ambas as imagens tem o seu ponto principal no mesmo local); f_x e f_y são as distâncias focais em x e y, respectivamente; b é a disparidade (distância entre as duas câmeras no eixo X). Devido ao fato da câmera deslocar apenas no eixo x temos que $y_1 = y_2$.

$$x = \frac{b\,(x_1 - p_x)}{(x_1 - x_2)} \qquad y = \frac{b\,f_x(y_1 - p_y)}{f_y(x_1 - x_2)} \qquad z = \frac{b\,f_x}{(x_1 - x_2)}$$

Equação 4: Fórmulas para encontrar a posição (coordenadas) x, y, z de um objeto por meio da visão binocular.

4.1.3. Detecção de objetos em vídeo

No processo de identificação de aeronaves, trabalha-se com objetos em movimento que devem ser capturados por meio de vídeos. Os vídeos correspondem a uma captura sequencial de imagens, também chamadas de quadros (*frames*), em que cada imagem corresponde a uma captura em certo instante de tempo, sendo que duas imagens são separadas por um intervalo de tempo Δt. Quanto menor for esse intervalo de tempo, maior será o número de imagens geradas a cada segundo. Assim, quanto mais imagens por segundo, mais fluido se torna o vídeo visualizado. Essa medida geralmente é expressa em quadros por segundo (*frames per second* – *fps*).

Os algoritmos de detecção em imagem descritos observam separadamente cada quadro do vídeo. Desde modo, o funcionamento dos algoritmos de detecção podem ser um processo custoso computacionalmente, não sendo viável de ser utilizado em tempo real, principalmente em vídeos com altas taxas de quadros por segundo. Além disso, com a evolução dos quadros, o objeto observado pode sofrer pequenas alterações em sua aparência ao longo das imagens, quer seja alterando seu formato,

quer seja o seu ângulo de observação. Ainda, o objeto pode sofrer uma oclusão parcial ou total, passando atrás de outro elemento da imagem, o que é um problema conhecido como *drift*.

Porém, outras abordagens analisam o deslocamento de *pixels* ao longo dos quadros. Desse modo, torna-se possível estimar o deslocamento dos objetos entre as cenas, permitindo, então, o cálculo da sua direção e velocidade. Esse é o princípio chamado de fluxo ótico, utilizado para otimizar o processo de detecção e permitir o rastreamento de um objeto em uma sequência de quadros. Esses algoritmos são conhecidos como **algoritmos de rastreamento**. Algumas técnicas também são capazes de adaptar-se à alteração da aparência do objeto ao longo dos quadros, utilizando técnicas de aprendizado em tempo de execução. Dá-se o nome a este tipo de rastreamento de aprendizado *online* (*online tracker*) (Yang et al., 2011, p. 3823–31). Dentre esses algoritmos, destacam-se:

- **MIL:** Um classificador que utiliza um método discriminativo para classificar o objeto alvo do fundo. Para isso ele parte de uma imagem inicial do objeto alvo e retira vários exemplos de imagem ao redor do objeto no próximo quadro, criando “pacotes” de exemplos positivos, fazendo um aprendizado com múltiplas instâncias (*Multiple Instance Learning* – MIL). A grande vantagem desse algoritmo é que apenas uma dessas instâncias necessita conter o objeto alvo para que o rastreamento ocorra corretamente (Babenko and Belongie, 2009);
- **KCF:** o filtro de correlações “kernelizados” (*Kernelized Correlation Filters*) parte do mesmo princípio do MIL, porém ele percebe que durante o aprendizado do MIL são escolhido muitos exemplos sobrepostos; assim, para otimizar o processo de escolha de exemplo, ele utiliza matrizes circundantes calculadas por meio de Transformada de Fourier (Henriques et al., 2012, p. 5–7);
- **TLD:** Estes algoritmo utiliza um princípio de “rastrear, aprender e detectar” (*Tracking Learning Detect* – TLD) e rastreia o objeto em todos os quadros: um detector localiza a posição do objeto alvo, verificando os erros e, caso

necessário, ele atualiza o detector, aprendendo novamente como representar o objeto (Kalal et al., 2012, p. 1409–22);

- **M^3:** Este é um algoritmo recentemente utilizando em pesquisas de detecção de aeronaves (Wu et al., 2017, p. 1–9). Ele utiliza um mapa de saliência para detecção e realiza o processo de rastreamento baseado em múltiplas fases, sendo que a primeira fase se utiliza de um filtro de *Kalman* que é refinado com a utilização de mapas de saliência e variação temporal.

Porém, algoritmos de rastreamento baseados em aprendizado *online* necessitam conhecer a posição inicial do objeto. Desse modo, um algoritmo de rastreamento baseado em aprendizado *online* ainda é dependente de uma detecção prévia do objeto. Nesses modelos pode-se utilizar os algoritmos de detecção previamente citados, além de algoritmos que aproveitam do fluxo de imagens capturadas em vídeo para criar um rastreamento. Esses algoritmos utilizam o princípio de fluxo ótico para identificar objetos salientes[12] na imagem, geralmente extraindo o objeto observado do fundo da cena. Dentre esses algoritmos, destacam-se aqueles baseados na extração de *background*, utilizando modelos gaussianos (MOG) (Stauffer and Grimson, 1999).

Neste trabalho de pesquisa foram realizados testes com diferentes algoritmos de algoritmos de rastreamento e diferentes algoritmos de detecção visual, com o objetivo de se encontrar o algoritmo ou a combinação de algoritmos que melhor se comportem para detecção de objetos por parte de sensores instalados em VANTs. Os testes realizados neste trabalho são apresentados em detalhes no capítulo 7 desta dissertação.

[12] Objetos salientes: Objetos que se destacam na imagem, tornando-se o principal ponto de atenção na imagem considerada (Hong et al., 2015, p. 597–606).

4.1.4. Detecção de objetos utilizando LIDAR

O LIDAR captura dados de um objeto observado em três dimensões, gravando não só a sua altura e a sua largura, mas também a sua distância ao sensor. Seus dados retornam como uma nuvem de pontos (*point cloud*). Porém, diferentemente do que ocorre com as imagens capturadas por câmeras, que são um tipo de dado contínuo, a nuvem de pontos é esparsa e com uma grande variedade de densidade. Cada ponto é uma lista contendo quatro valores (x, y, z, i), sendo que x, y e z são as coordenadas espaciais onde o ponto se encontra e i é a intensidade do sinal desse ponto. Em um processo de captura de n pontos, os dados são salvos como uma matriz, n x 4, em que n é número de pontos e 4 representa os valores (x, y, z, i) para cada ponto. A Figura 20 ilustra como os dados de um LIDAR são processados por um programa.

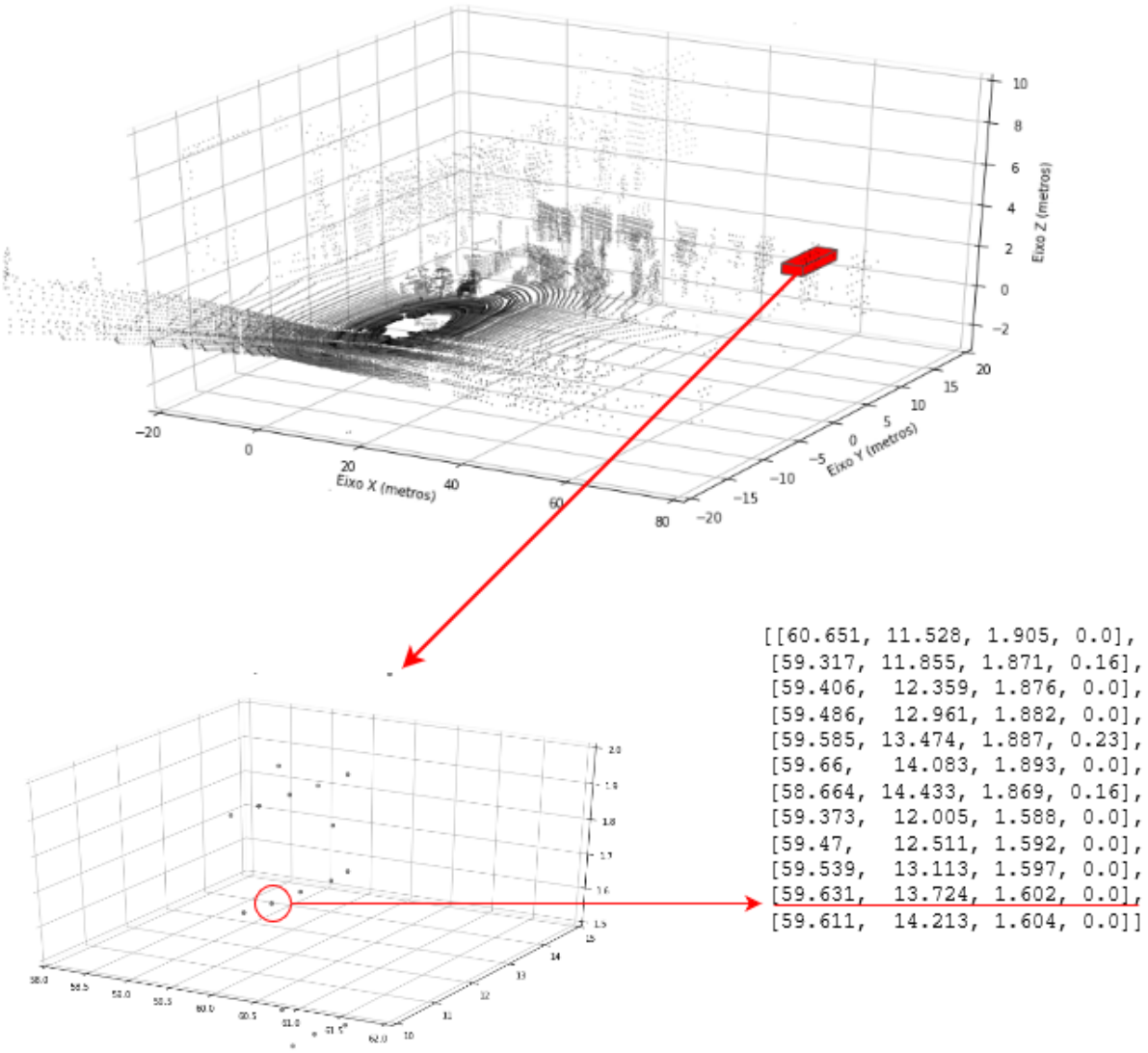

Figura 20: Representação dos dados capturados por um LIDAR conectado em um carro: exemplo extraído da base de dados KITTI (Andreas Geiger, Philip Lenz and Raquel Urtasun, 2012**).**

Na maioria dos trabalhos de detecção de objetos por meio dos dados capturados por um LIDAR, os algoritmos inicialmente discretizam os dados recebidos em uma *grade voxel 3D*, ou seja, o espaço observado é transformado em pequenos cubos, sendo que a partir de cada cubo são extraídas características que representam os pontos de

seu interior. Sobre esta grade de dados são aplicados algoritmos de aprendizado de máquina, que classificam os dados como contendo ou não um objeto. Em algumas pesquisas utilizam-se o conceito de “janela deslizantes 3D” no passo da classificação. A Figura 21 demonstra esse processo.

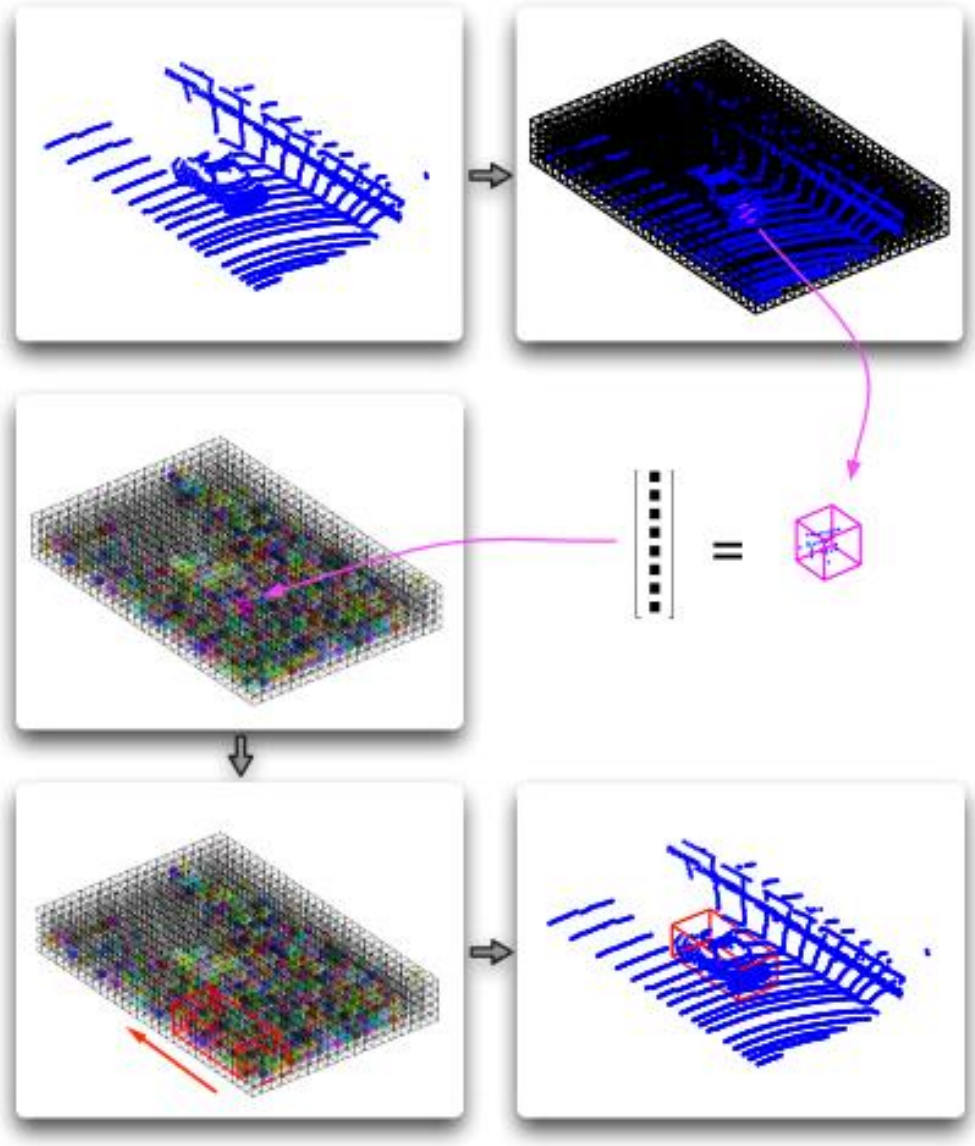

Figura 21: Processo de detecção utilizando dados do LIDAR: a) na 1ª imagem, tem-se o dado bruto; b) na 2ª imagem, ocorre a discretização dos dados; c) a 3ª imagem apresenta a extração de caraterísticas de um *voxel* 3D; d) a 4ª imagem ilustra o conceito de janelas deslizantes 3D percorrendo os dados; e) na 5ª imagem, tem-se o resultado da detecção de um veículo. Imagem retirada de (Wang and Posner, 201**5).**

Dois fatores são cruciais para uma boa performance da detecção de um objeto por parte de um LIDAR: a escolha do algoritmo de extração de características do objeto na imagem e a escolha do algoritmo de aprendizado. Diversos algoritmos são utilizados para estas duas fases e, visando comparar estas técnicas, algumas pesquisas utilizam um conjunto de dados em comum e compartilhados, sendo o mais famoso deles o KITTI. A base KITTI inclui dados capturados por um LIDAR e por câmeras visuais RGB extraídas de um carro se locomovendo por uma cidade. O objetivo é identificar ciclistas, outros carros e também pedestres cruzando as ruas. O *Karlsruhe Institute of Technology* apresenta em seu *site*[13] um *benchmark* que compara os melhores algoritmos de detecção. Dentre esses algoritmos de aprendizado de máquina destacam-se as seguintes abordagens:

- **SVM:** Assim como na detecção em imagem, o SVM é um algoritmo altamente utilizados em detecções com o LIDAR, alcançando bons resultados no conjunto de dados KITTI (Wang and Posner, 2015);
- ***Boosting*:** Técnicas de *Boosting* consistem em aperfeiçoar uma classificação a partir de algoritmos encadeados, normalmente utilizando algoritmos baseados em árvores, como o *LigthGBM*, descrito no subcapítulo anterior. (Xiao et al., 2015, p. 192–98) (Teichman et al., 2011, p. 4034–41);
- **CNN:** Inspirado no uso em imagens, os dados do LIDAR também podem utilizar redes convolucionais como algoritmo de detecção; porém, no LIDAR, os modelos utilizam camadas de convolução 3D. Em vários trabalhos de pesquisa as redes convolucionais trabalham em conjunto com o conceito de janelas deslizantes 3D, o qual divide a captura do objeto em pequenos cubos menores, que são processados individualmente, o que facilita a detecção do objeto. Este modelo encontra-se entre os mais bem sucedidos no *benchmark* criado para o conjunto de dados KITTI (Caltagirone et al., 2017, p. 1019–24) (Shi et al., 2019).

[13] http://www.cvlibs.net/datasets/kitti/, site do Karlsruhe Institute of Technology.

4.1.5. Detecção de objetos de forma cooperativa

Os algoritmos de detecção utilizados nos sensores cooperativos têm uma abordagem diferentes, pois nesses sensores não é necessário extrair de dados brutos obtidos a posição e a direção de aeronaves, pois tais informações são enviadas pelas próprias aeronaves por meio de um protocolo conhecido.

Estes tipos de sensores ainda contam com algumas versões de algoritmos para calcular uma possível situação de colisão com outra aeronave, estimando a posição futura desta outra aeronave e calculando uma possível rota de desvio. Estes algoritmos são apresentados em ordem cronológica do surgimento das tecnologias por eles empregadas.

- **TCAS:** O TCAS contém uma série de algoritmos para identificar um possível risco de colisão de uma aeronave com outra aeronave que se encontra relativamente próxima. O primeiro passo para essa identificação é a coleta de dados da aeronave “intrusa”, por meio de comunicação entre os sensores das duas aeronaves. Porém, conforme o sensor presente na aeronave, as informações recebidas pelo TCAS podem conter mais ou menos detalhes e, por este motivo, é necessário algum algoritmo para estimar algumas destas informações. Para projetar a aeronave no tempo e espaço futuros, o algoritmo utiliza-se de um cálculo de extrapolação linear e, a partir desta métrica, um algoritmo diferente calcula o valor do tempo mínimo para um encontro entre as aeronaves (FAA - Federal Aviation Administration, 2011, p. 1–50);
- **ACAS-X:** O ACAS-X (*Airborne Collision Avoidance System*) é o sistema de detecção de conflito utilizado em projetos avançados de aviação. Nele são implementadas mudanças no algoritmo de detecção original, que passa a corrigir os ruídos presentes no sensor por meio de um modelo probabilístico. Este modelo, inicialmente, aplica um filtro de *Kalman* nos dados de posição, ângulo *bearing* e altitude, e esses dados servem de entrada para um processo de decisão *markoviano* quantitativo, a partir dos quais são calculados os

prováveis estados futuros da aeronave intrusa (Kochenderfer and Chryssanthacopoulos, 2011) (Kochenderfer et al., 2013, p. 17–33).

4.2. Algoritmos de fusão de sensores

Em técnicas e em pesquisas que utilizam mais de um tipo de sensor para o processo de detecção de objetos é necessário se utilizar de algoritmos que processem as respostas fornecidas por todos os sensores, retornando uma única resposta de detecção. Essa técnica é comumente chamada de fusão de sensores e, assim como no processo de detecção utilizando um único sensor, elas podem se utilizar de várias outras técnicas. Dentre essas outras técnicas, destacam-se as seguintes:

- **Medidas de Tendência Central:** Trata-se de uma das técnicas mais simples que pode ser utilizada para fundir dados de detecção de vários sensores, calculando-se o ponto médio entre eles. Para isso é possível se utilizar de fórmulas simples como o cálculo da média ou da mediana dos valores obtidos dos sensores individualmente;
- **Técnicas Hierárquicas:** Outras operações simples que podem ser aplicadas para fusão de sensores é a utilização de hierarquias. Por meio das hierarquias pode-se criar um algoritmo que sempre dá prioridade ao sensor mais confiável, e somente nos momentos de falha de detecção nesse sensor, ou em momentos específicos em que se diminui a sua confiança, serão observados os dados dos demais sensores. Em (Ramasamy et al., 2014, p. 271–76) apresenta-se uma técnica de fusão de sensores por intermédio de um circuito no qual os sensores cooperativos têm maior prioridade sobre os sensores não cooperativos;
- **Lógica *Fuzzy*:** Em técnicas utilizando Lógica *Fuzzy* pode-se trabalhar com cálculos nos quais existam uma “certeza parcial” do dado analisado. Estendendo-se esta técnica para o processo de detecção de objetos, pode-se utilizar o grau de confiança em cada captura feita por determinado sensor, buscando-se encontrar o ponto mais confiável entre todas as capturas (Aliustaoglu et al., 2009, p. 539–46);

- **Filtro de *Kalman*:** O Filtro de *Kalman*, e suas derivações, constitui um dos métodos mais utilizados na fusão de sensores (Kochenderfer and Chryssanthacopoulos, 2011) (Cappello et al., 2015, p. 714–22). Ele é matematicamente ótimo para estimar o erro de uma função ao longo do tempo, calculando os possíveis valores que a função ideal (sem ruídos) poderia alcançar. Ele pode ser aplicado em fusão de sensores, calculando a distribuição do erro de uma detecção e, assim, estimar os possíveis dados de localização que um objeto detectado poderia alcançar. Sobrepondo as localizações estimadas de vários sensores, pode-se encontrar a posição mais provável do objeto procurado.

4.3. Algoritmos de estimativa de rota de objetos

Tendo detectado o objeto próximo da aeronave que está sendo controlada, o próximo passo da técnica de *sense and avoid* é estimar a rota deste objeto e calcular a probabilidade de um possível cenário de colisão com a aeronave controlada. Para alcançar este objetivo é necessária a utilização de algoritmos de estimativa de rota da outra aeronave e confrontar esta estimativa com a rota planejada para a aeronave controlada. Dentre esses algoritmos, destacam-se os seguintes:

- **Cálculos Geométricos:** A partir de múltiplas observações de um objeto pode-se calcular a sua trajetória como uma função de regressão, dado que em grande parte dos casos os objetos apresentam um movimento uniforme. Esse cálculo pode ser simplificado por uma regressão linear, ou em algumas ocasiões, pode-se utilizar uma regressão polinomial;
- **Algoritmos *Bayesianos*:** Devido a interferências externas, como a do vento, e de outras condições meteorológicas adversas, ou por decisões internas da tripulação, uma aeronave pode ter sua rota desviada. Deste modo, pode-se considerar a função da rota como sendo uma função estocástica. Uma abordagem bastante utilizada no planejamento de rota utiliza funções baseadas no teorema de *Bayes* a partir do qual é possível calcular a probabilidade de um

evento ocorrer, baseada em conhecimentos a priori dos eventos relacionados (Furukawa et al., 2006, p. 2521–26)**;**

- **Planejamento Automático:** Nas técnicas de planejamento automático, calcula-se um possível estado futuro de uma aeronave a partir do seu estado atual e das possíveis ações capazes de serem tomadas por esta aeronave. Aplicando-se esta lógica recursivamente, obtém-se uma árvore de possibilidades. Assim, nesta técnica, encontrar um possível estado futuro e a sua probabilidade de ocorrência (por exemplo a probabilidade de um estado de colisão) torna-se um problema de busca em árvores. Os cálculos necessários baseiam-se em técnicas de busca em profundidade ou em heurísticas, como o algoritmo A*;
- **Modelos de Decisão *Markovianos*:** Modelos de decisão *markovianos* são estocásticos, baseados nos conceitos de estados e ações. A partir de um determinado estado s pode se tomar ações a, e, para cada ação a, existe uma função probabilística que leva a um novo estado s'. Isso pode ser descrito por $P(s' | s, a)$. Ou seja, nestes modelos, um estado inicial tem a probabilidade de levar o sistema a um estado futuro por meio de uma ação. Este modelo pode ser utilizado para calcular a probabilidade de um possível estado de colisão, a partir de ações que cada aeronave pode executar, e das probabilidades dessas aeronaves executarem tais ações (Kochenderfer and Chryssanthacopoulos, 2011)**.**

4.4. Algoritmos de resolução de conflito

Após detectado o objeto, ocorre o último processo do *sense and avoid*: o **desvio**. Este processo recebe as posições, direções e velocidades do objeto identificado e da aeronave sendo controlado, além do local onde ocorrerá o possível cenário de colisão entre a aeronave controlada e o objeto identificado. Com esses dados, o algoritmo de desvio calcula uma nova rota para a aeronave controlada que evitará essa colisão, ao

mesmo tempo que interferirá o mínimo possível no plano de voo inicial dessa aeronave.

Entre as principais abordagens utilizadas para criação do algoritmo de resolução de conflito destacam-se:

- **Algoritmo baseado em estados:** Os principais modelos utilizados atualmente são baseados em estados previamente analisados. Estes algoritmos seguem lógicas determinísticas e estáticas, previamente examinadas por um especialista. Por exemplo, no algoritmo utilizado pelo TCAS, apesar de se utilizar de diversas variáveis, a altura tem um papel crucial, funciona em um cenário de colisão, e o replanejamento da rota apenas utiliza manobras verticais, priorizando que as aeronaves continuem no sentido que estão. Assim aeronaves que apresentem um nível mais alto de altura tendem a aumentar sua altura, enquanto que as aeronaves em um nível mais baixo de altura tendem a diminuir sua altura. Outro exemplo de algoritmo baseado em estados é uma técnica de desvio horizontal, na qual ao se deparar com uma situação de conflito, o algoritmo sugere que a aeronave faça um voo lateralmente no sentido divergente ao da aeronave intrusa, como ilustrado na Figura 22;

Figura 22: Ilustração da técnica de resolução de conflito entre aeronaves: a) Situação de conflito; b) Plano de resolução de conflito; c) Reestabelecimento da rota original.

- **Modelos de Decisão *Markovianos*:** Assim como os modelos de decisão *markovianos* podem ser úteis na estimativa de probabilidade de um cenário de conflito, como descrito em 4.3, eles também podem ser utilizados no processo de resolução desse mesmo conflito, buscando a sequência de ações que gerem a maior probabilidade de se chegar a um estado seguro, no qual não ocorra o cenário de conflito;
- **Aprendizado por Demonstração:** Em modelos de aprendizado por demonstração, o algoritmo recebe dados de voos de especialistas. Nestes voos incluem-se situações de colisão as quais o piloto tenha resolvido por meio de sua habilidade. Observando esses dados, o algoritmo “aprende” padrões de voos que podem ser utilizados na resolução de conflito em um cenário futuro (Matsumoto et al., 2015, p. 2771–79).

5. Arquitetura do sistema *sense and avoid* proposta

Visando criar um *framework* capaz de se utilizar dos diversos sensores em conjunto e possibilitando a experimentação de algoritmos diversos de detecção e/ou de desvio de modo simples, foi necessária a criação de uma arquitetura modular objetivando orientar a construção do sistema, organizada segundo alguns princípios de padrões de projeto.

Para que a criação da arquitetura do sistema *sense and avoid* (*framework*) se desse da maneira mais eficiente, seu processo foi divido em três etapas. Na primeira etapa foi realizado um levantamento de requisitos, identificando-se quais as principais características necessárias para o *framework*, além de possíveis problemas que deveriam ser resolvidos por essa arquitetura. Na segunda etapa foi criada uma visão da arquitetura em alto nível, com o objetivo de reconhecer as fases do processo, suas interações, além de possíveis protocolos de comunicação necessários para a interface entre essas fases. Na terceira e última etapa foi criada uma modelagem da arquitetura aprofundada, com um diagrama de classe simplificado, com o objetivo de facilitar o processo de implementação da arquitetura. Cada uma dessas etapas é apresentada nos próximos três subcapítulos.

5.1. Análise de requisitos

O primeiro passo para construção do *framework* é analisar as suas necessidades e caraterísticas, ou seja, seus requisitos. Durante esse processo foram investigadas quais funcionalidades são essenciais para o sistema e como elas podem ser executas, facilitando assim o processo de criação da arquitetura e posteriormente de implementação do *framework*.

Visando alcançar o objetivo principal do *framework* proposto neste trabalho, (desenvolver um *framework* modular de *sense and avoid* para VANTs, que permita a trocas de sensores e algoritmos), e com a intenção que este *framework* possa ser

utilizado de uma maneira simples e intuitiva, foram elencados alguns requisitos essenciais:

- **Modularidade:** O principal requisito deste *framework* é a sua modularidade. Um *framework* modular é dividido em fases que trabalham de forma independente, comunicando-se a partir de interfaces ou protocolos. Ao se utilizar a modularidade no projeto proposto, divide-se o *framework* em pequenas fases, que trabalham de forma a não precisar conhecer a implementação das demais, apenas se comunicando com elas por intermédio de uma interface padronizada. Isso possibilitará as trocas ou alterações de cada fase de forma independente, o que dará a possibilidade de personalizações no seu modo de funcionamento por parte do usuário do *framework;*
- **Possibilidade de troca de algoritmos:** Um caso especial do uso da modularidade do sistema está na necessidade de personalizações dos algoritmos utilizados pelo *framework* - nos algoritmos de detecção de um possível conflito, na técnica de fusão de sensores ou no cálculo de uma rota de desvio. A implementação deste requisito permite que o *framework* seja utilizado para testes de diferentes técnicas, visando-se encontrar a melhor dentre elas. Este requisito também impede que o *framework* se torne obsoleto caso um novo algoritmo seja criado, permitindo, neste caso, que este algoritmo seja incluído ao sistema proposto;
- **Flexibilidade a diferentes tipos de sensores:** Como apresentado no capítulo 3 deste trabalho, a detecção de um objeto pode se dar de várias formas, utilizando-se de diversos tipos de sensores. Um *framework* completo deve ser robusto a estas diferenças, possibilitando o uso de diferentes modelos de sensores e permitindo trocas de modelos. Este requisito impacta diretamente a detecção do conflito, pois os sensores presentes nos veículos aéreos são fundamentais neste processo. Porém, como demonstrado no capítulo anterior, sensores diferentes utilizam tecnologias diferentes e retornam suas leituras de modos distintos. Assim, para que o *framework* proposto neste

trabalho trabalhe de maneira a cobrir o máximo de sensores possível faz-se necessária a criação de um protocolo genérico de comunicação entre os sensores e as fases posteriores do sistema, de modo que se considere suas diferentes maneiras de envio e leitura de dados;

- **Capacidade de utilizar quantidade variada de sensores:** Além de permitir o uso de diferentes modelos de sensores, a arquitetura também deve permitir a mudança no número sensores de maneira simples e eficiente, possibilitando ao usuário do sistema adicioná-los ou removê-los quando julgar necessário, o que possibilita a criação de diversas modelagens para teste, auxiliando na busca pela melhor técnica de detecção. Possibilitar que esta troca aconteça em tempo de execução concede funcionalidades extra ao *framework*, pois desta maneira é possível criar um sistema robusto a falhas, que se adapte quando houver um defeito em um sensor, reajustando o sistema de detecção e desvio de modo a não ser dependente deste dispositivo defeituoso, que seria automaticamente isolado e impediria que a falha se propagasse para as fases seguintes do *framework*;
- **Independência da plataforma aeronáutica considerada:** Outra caraterística importante para o *framework* é ser independente do tipo de plataforma aeronáutica, possibilitando a sua utilização em diferentes tipos de veículos, ou mesmo em um simulador, sem serem necessárias mudanças em seu algoritmo interno. Este requisito pode ser alcançado com o emprego de um protocolo de comunicação distinto, amplamente conhecido e utilizado. Comunicar-se por intermédio de um protocolo padronizado faz com que o *framework* não seja criado para uma aeronave ou simulador específicos, podendo se comunicar com qualquer aeronave, sistema ou veículo que permita o seu controle a partir deste *framework*;
- **Interface com o usuário simples:** Conceber uma interface do *framework* com o usuário simples permite que outros pesquisadores possam utilizá-lo para futuras pesquisas, e que trabalhos futuros possam se aproveitar dele.

Todos esses requisitos podem ser alcançados com uma boa modelagem da arquitetura, aplicando-se conceitos de engenharia de *software* e criando-se cada fase do sistema de *sense and avoid* como blocos coesos, segundo o princípio de responsabilidade única (E Gamma, R Helm, R Johnson, 1994), em que cada bloco tem apenas uma determinada função, comunicando-se com os demais blocos por meio de uma interface específica. Tendo em vistas estas práticas, foi criada a arquitetura *sense and avoid* proposta neste trabalho, e que passa agora a ser apresentada nos subcapítulos seguintes.

5.2. Arquitetura em alto nível do *sense and avoid*

Visando a modularidade, a arquitetura proposta neste trabalho divide o sistema *sense and avoid* em quatro fases distintas: o **sensoriamento**, a **detecção**, o **desvio** e o **controle**. Cada uma dessas quatro fases é responsável por um subsistema do *framework*. Todas essas fases comunicam-se com a aeronave por intermédio de uma interface de comunicação comum.

Na Figura 23 há uma representação de alto nível da arquitetura do sistema *sense and avoid*, destacando a interação e a comunicação de cada fase. Nela o *framework* representado está envolto por linha tracejada e cada bloco interno representa um subsistema, enquanto que a aeronave (aeronave controlada) e o usuário são representados por blocos externos que se comunicam com o *framework*.

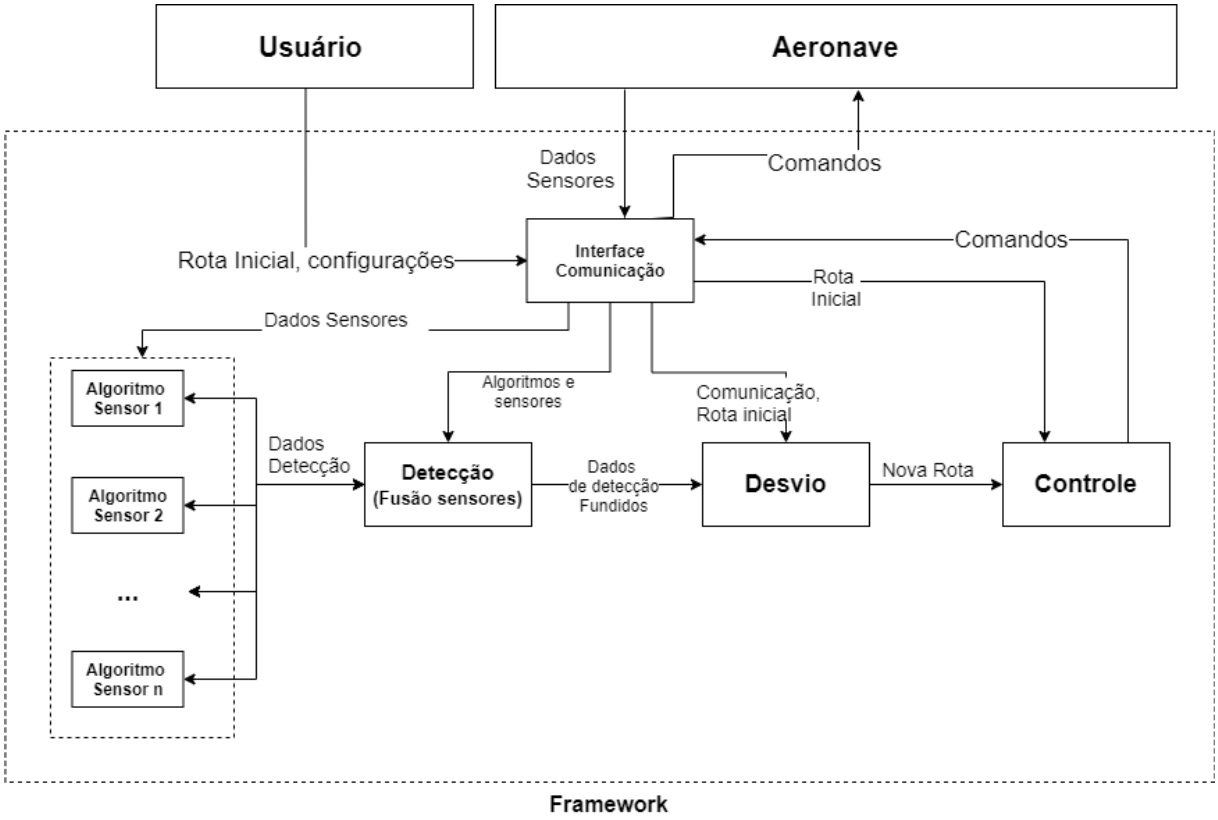

Figura 23: Fluxo do sistema *Sense and avoid.*

Os subsistemas do *framework* trabalham de maneira paralela e independente, sendo que cada um é executando em um *thread* específico (um subprocesso do programa), comunicando-se, quando for necessário, com o próximo subsistema por meio de uma interface, recebendo os dados necessário para a execução de suas tarefas e enviando os dados necessários para a próxima fase. Dessa forma, encapsulam-se suas implementações na forma de caixas pretas.

Em uma visão geral, o sistema executa inicialmente o bloco “Interface Comunicação”, que é responsável por receber os dados da aeronave (como as informações de sensores) e as configurações do usuário, repassando-os para as fases seguintes do sistema. Este bloco também tem como função iniciar as fases “Detecção” e “Desvio”, assim como passar a rota inicial para iniciar o controle (nova rota concebida para a aeronave). Em seguida, a fase “Detecção” inicia os algoritmos de sensoriamento com base na configuração dada pelo usuário do sistema, não havendo limite máximo de

algoritmos de detecção, porém deve haver pelo menos um. Estes algoritmos são responsáveis por fazer um sensoriamento contínuo em busca de objetos ou aeronaves intrusas, com possíveis riscos de colisão com a aeronave controlada, enviando constantemente relatórios sobre essa busca de volta para a fase “Detecção”. Ao receber os relatórios de vários sensores, a fase “Detecção” aplica um algoritmo de fusão de sensores, visando verificar se existe algum risco de colisão com a outra aeronave (aeronave “intrusa”). Caso o risco seja confirmado, este bloco envia uma mensagem para o bloco “Desvio”. Com a mensagem de um possível risco de colisão disponível, o algoritmo de desvio traça uma nova rota que seja capaz de corrigir o conflito eminente, e esta nova rota é enviada à fase “Controle”, que dará prioridade à nova rota em detrimento à rota inicialmente recebida pela “Interface Comunicação”. O “Controle”, por sua vez, é responsável por calcular quais manobras são necessárias para que a aeronave cumpra a rota enviada. Esses comandos são enviados à “Interface Comunicação” que, por sua vez, os enviará para aeronave controlada, segundo o protocolo *MavLink*[14] (*Micro Air Vehicle Link*), amplamente utilizado em aeronaves de pequeno porte.

Os subsistemas responsáveis pelos algoritmos de sensoriamento, pela fusão de sensores e pelo de desvio podem ser alterados (intercambiados) independentemente, desde que respeitem a interface de comunicação com os demais blocos do sistema. Cada um desses subsistemas é detalhado a seguir.

5.2.1. Interface de comunicação

O subsistema “Interface Comunicação” visa criar uma maneira de comunicação com *framework* simples e transparente para o usuário ou para a aeronave controlada, ocultando a complexidade dos subsistemas internos, a partir de uma interface

[14] *MavLink*: Protocolo de comunicação com VANTs, utilizado para enviar comandos de controle ao veículo ou receber dados de seus sensores e estados. Este protocolo está presente nos principais sistemas de controle (Koubaa et al., 2019, p. 87658–80).

unificada, semelhante à proposta no padrão de projeto apresentando por Gamma, denominado Façade (E Gamma, R Helm, R Johnson, 1994). Ela será o meio de comunicação com todo o *framework* e é responsável por repassar os dados recebidos por agentes externos (usuários, aeronaves e sensores) para os subsistemas internos.

Este bloco é o ponto de partida do sistema, iniciando todos os demais processos do sistema *sense and avoid*. Ao iniciá-lo, o usuário deve passar três conjuntos de dados: a) os dados de comunicação com a aeronave controlada; b) a rota inicial da aeronave controlada; e c) os dados sobre a modelagem pretendida para a arquitetura de *sense and avoid*.

Os **dados de comunicação com a aeronave** são as informações necessárias para que o sistema consiga controlá-la e ler os seus sensores, ou seja, são os dados necessário para se criar um canal de comunicação entre o veículo e o *framework*. Estes dados variam conforme a implementação utilizada para a interface de comunicação. Nos testes foi utilizado o protocolo TCP/IP. Assim, os dados de comunicação são o endereço IP e a porta da aeronave controlada.

A **rota inicial da aeronave controlada** é a rota planejada para o veículo, que será o caminho que a aeronave fará caso não encontre nenhum obstáculo no seu caminho. Essa rota é passada como um conjunto de pontos-alvos (*way points*), que consistem em pontos em 3 dimensões por onde a aeronave deverá passar.

A **modelagem pretendida para a arquitetura de *sense and avoid*** contém os dados sobre os sensores e algoritmos que são utilizados pela aeronave controlada. Ela contém três grupos de informações: o primeiro grupo contém os sensores utilizados para detecção, bem como o algoritmo utilizado em cada sensor (Ex: LIDAR: *Deep Neural Network*; câmeras: *Random Forest*); o segundo grupo contém o algoritmo de fusão de sensores; e o último grupo, o algoritmo de desvio.

Após iniciado o sistema e modelado o *framework*, o subsistema “Interface Comunicação” continuará ativo, como um canal de comunicação contínuo com a aeronave, recebendo os dados de seus sensores e repassando-os para os

subsistemas do *framework* que irão consumir estes dados, ao mesmo tempo que envia de volta à aeronave os dados de controle, responsável por pilotar e executar as manobras na aeronave.

5.2.2. Controle

O subsistema “Controle” é responsável por enviar comandos de movimento do veículo (aeronave). Ele recebe uma rota inicial com os pontos pelos quais o VANT deveria passar. Caso uma ameaça seja detectada, o controle tem como funcionalidade atualizar a rota do VANT, finalizando automaticamente a rota em curso e iniciando a nova rota recebida pelo algoritmo de desvio, agora livre do risco de colisão com o objeto intruso.

5.2.3. Algoritmos de sensoriamento

A fase de “Sensoriamento” do *framework* é responsável por se comunicar com os sensores (blocos Algoritmo Sensor 1, Algoritmo Sensor 2, ... e Algoritmo Sensor n), recebendo seus dados e os utilizando em algoritmos para identificar a possível presença de intrusos.

Nesta fase pode-se alterar o número de blocos que representam os sensores ou algoritmos utilizados. Para isso o *framework* foi criado de modo que esta fase tenha um número indeterminado de subsistemas. Deste modo, o *framework* pode ter o número de blocos responsáveis pelos algoritmos de detecção tão grande quando o usuário do *framework* julgar necessário, sendo obrigatório ter ao menos um bloco de algoritmo de detecção implementado.

Porém, cada sensor tem uma forma particular de reconhecer um objeto intruso, utilizando algoritmos próprios, com acurácias e confiabilidades diferentes. Em alguns casos, um mesmo sensor pode ter associado a si diversos algoritmos para a detecção de objetos intrusos, como, por exemplo as câmeras, para as quais existem inúmeros algoritmos de detecção por vídeo. Um sistema completo e seguro de detecção utiliza vários sensores e algoritmos diferentes ao mesmo tempo.

Visando que se torne possível a existência desse número flexível dos algoritmos de detecção, neste trabalho é sugerido um modelo genérico de comunicação entre os blocos de sensoriamento e a fase de detecção (responsável pelo gerenciamento dos diversos algoritmos). Ele foi criado visando englobar o máximo possível de técnicas, permitindo que cada uma possa ser um processo sendo executado computacionalmente em paralelo (como um *thread*).

Para isso é fundamental que todos os blocos tenham um formato padrão e sigam o modelo de comunicação concebido. Para este *framework* foi criada uma interface a qual todos os algoritmos de sensoriamento devem seguir como modelo. Desse modo, os algoritmos terão um ciclo de vida similar, sendo iniciados pela fase de detecção, e recebem dele os dados das aeronaves, assim como também são capazes de enviar ao bloco de detecção uma mensagem padronizada informando sobre um possível risco de colisão encontrado.

As mensagens trocadas seguem o protocolo de comunicação criado, devendo ser enviadas em intervalos constantes de tempo, contendo quatro grupos de informações (***status* de detecção**, **posição da aeronave**, **posição relativa do intruso** e **posição absoluta do intruso**) que são detalhadas a seguir:

- **Status de detecção:** Este é um valor *booleano* (verdadeiro ou falso) que informa se um objeto (aeronave intrusa) foi, ou não, encontrado pelo algoritmo;
- **Posição da aeronave:** Alguns sensores não são responsáveis por identificar aeronaves intrusas, e sim por identificar a posição atual do veículo controlado. Estes podem retornar dados constituindo-se de uma lista contento três valores inteiros, representando o ponto atual no espaço em que a aeronave se encontra; seus valores serão descritos por x_{meu}, y_{meu} e z_{meu} do veículo, porém nem sempre os sensores retornam os valores neste padrão. Assim, este bloco tem por funcionalidade converter os dados recebidos por este tipo de sensor para o padrão proposto neste trabalho. Esta leitura será retornada por sensores como os inerciais e pelo GPS, enquanto que os demais sensores devem retornar o campo de posição da aeronave como “nulo”;

- **Posição relativa do intruso:** A maioria dos sensores de detecção calcula a posição de objeto detectado em relação à posição do veículo que está sendo controlado. Este é o caso de sensores como o LIDAR e câmeras, dentre outros. Assim como o dado de posição da aeronave, a posição relativa do intruso também é representada por um vetor contendo três valores descritos por x_{rel}, y_{rel} e z_{rel}. Porém, neste caso, representando a distância relativa do objeto para a aeronave controlada, ou seja, é retornado um "vetor" entre o intruso e o veículo controlado;
- **Posição absoluta do intruso:** Por fim, alguns sensores fornecem os valores absolutos da posição do objeto observado. Este é o caso dos sensores cooperativos. Neste caso, também é retornada uma lista com três valores, denominados por x_{int}, y_{int} e z_{int}, que representam os valores globais da posição do intruso.

Assim, cada bloco retorna uma lista com 10 valores (*detect*, x_{meu}, y_{meu}, z_{meu}, x_{rel}, y_{rel}, z_{rel}, x_{int}, y_{int}, z_{int}). Porém, deve-se destacar que nenhum sensor consegue obter ou fornecer todos esses dados simultaneamente. Desse modo, os dados que não forem possíveis de serem encontrados por estes sensores são retornados como "nulo". O mesmo acontece quando nenhum intruso é encontrado, retornando-se, desta forma, a detecção como "falsa" e todos os demais valores como "nulo".

5.2.4. Detecção (fusão de sensores)

O subsistema "Detecção (Fusão sensores)" irá coordenar os algoritmos de sensoriamento. Este bloco é responsável por iniciar os algoritmos e por receber deles se ocorreu ou não a detecção de um objeto intruso em risco de colisão com o veículo controlado. Caso vários algoritmos identifiquem o objeto intruso simultaneamente, o subsistema de detecção irá analisar as diferentes informações visando encontrar o real ponto de detecção do veículo controlado com o objeto intruso.

Ao iniciar o *framework*, o usuário do sistema deve selecionar os sensores e algoritmos que serão utilizados. Isso é feito enviando-se uma lista com os nomes ou classes dos

algoritmos que se pretende utilizar, porém o usuário também tem a opção de implementar seu próprio algoritmo, desde que este algoritmo siga o protocolo descrito em 5.2.3. Esta lista de algoritmos será repassada como entrada para o bloco “Detecção” que inicia os algoritmos como *threads*.

Após iniciar os algoritmos, o bloco “Detecção” entra em modo de espera até que receba uma resposta de detecção de objetos pelos algoritmos. Porém, como foi descrito anteriormente, apesar de todos os algoritmos utilizarem o mesmo protocolo para retornar seus dados, cada algoritmo pode enviar dados diferentes sobre a detecção. Assim, o bloco “Detecção” também é responsável pela fusão dos dados dos sensores, com a intenção de reunir todos os dados como se fossem um só.

Quando dois algoritmos retornam que um risco de colisão foi encontrado, porém em posições diferentes, é utilizada uma técnica aplicada para a fusão de informações, podendo ser, por exemplo, a média simples dos valores, a média ponderada com base na confiança no dado de cada sensor, ou um cálculo personalizado pelo usuário. O bloco de detecção também é responsável por observar o movimento do objeto alvo ao longo do tempo, para, com isso, calcular a sua direção e a sua velocidade.

Após a fusão dos sensores, e o cálculo da direção e velocidade do objeto detectado (caso houver mais de uma detecção em um período curto de tempo), o bloco de Detecção retorna os dados obtidos por meio de um protocolo similar ao dos algoritmos, porém com alguns acréscimos. Neste protocolo são retornados dados sobre as duas aeronaves envolvidas, a aeronave controlada e a aeronave intrusa. Para cada aeronave são retornadas dois grupos de valores: um grupo representa a posição da aeronave, enquanto que o outro grupo representa o versor de direção da aeronave. Dentre os dados da aeronave intrusa, são retornados os dados da posição relativa entre a aeronave intrusa e a aeronave controlada, referenciadas como (x_{int},y_{int},z_{int}), e os dados do versor de direção, referenciado como (x'_{int},y'_{int},z'_{int}). Ainda, para a aeronave controlada, são enviados os dados de sua posição absoluta (x_{meu},y_{meu},z_{meu}) e os dados de seu versor de direção (x'_{meu},y'_{meu},z'_{meu}). Para ambas as

aeronaves é enviada, também, a sua velocidade (em m/s). A Figura 24 apresenta essas informações para uma aeronave.

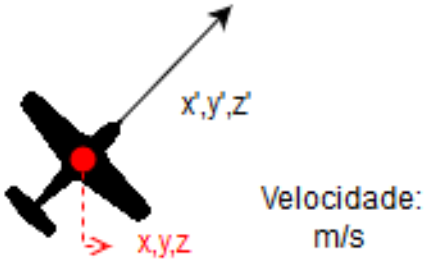

Figura 24: Ilustração dos valores retornados por um algoritmo de detecção presente em um sensor.

Assim, a mensagem final enviada para o bloco de Desvio contém as seguintes informações:

$$(x_{meu}, y_{meu}, z_{meu}, x'_{meu}, y'_{meu}, z'_{meu}, velocidade_{meu}, x_{int}, y_{int}, z_{int}, x'_{int}, y'_{int}, z'_{int}, velocidade_{int})$$

representando, respectivamente, as coordenadas (x,y,z) da aeronave sendo controlada, seus valores de vetor de direção (x',y',z'), além de sua velocidade; da mesma forma, essas informações referentes ao objeto intruso também são passadas.

5.2.5. Algoritmo de desvio

Por fim, tem-se o bloco “Desvio”, que é responsável por ler os dados recebidos do algoritmo de detecção e, assim, traçar uma rota de fuga da ameaça de colisão, retornando para o bloco “Controle” uma rota alternativa para a aeronave controlada, livre de riscos de colisão com o objeto intruso.

Este bloco também está em constante comunicação com os sensores passivos (p.ex., o ADS-B), permitindo uma troca de informações entre as aeronaves para a criação de

uma rota de fuga sincronizada por ambas, impedindo que ambas as aeronaves tracem a mesma rota e gerem um novo cenário de colisão.

Neste bloco também é possível alterar os algoritmos, permitindo testes com diferentes técnicas de desvio, para que assim seja possível compará-las do ponto de vista de eficácia e custo computacional.

5.3. Arquitetura detalhada do *Sense and Avoid*

Para que sejam alcançados os objetivos apresentados na análise de requisitos para a Arquitetura *Sense and Avoid* é necessário implementá-la segundo alguns padrões de projetos. Nesta seção é abordada a maneira como foi organizado o *design* da arquitetura, com o foco em se alcançar esses objetivos. Na Figura 25 é apresentando um diagrama de classes em alto nível da Arquitetura *Sense and Avoid* proposta, exibindo as relações entre as classes, sem entrar em detalhes de suas variáveis e métodos. As figuras subsequentes apresentam em detalhes alguns de seus pontos estratégicos.

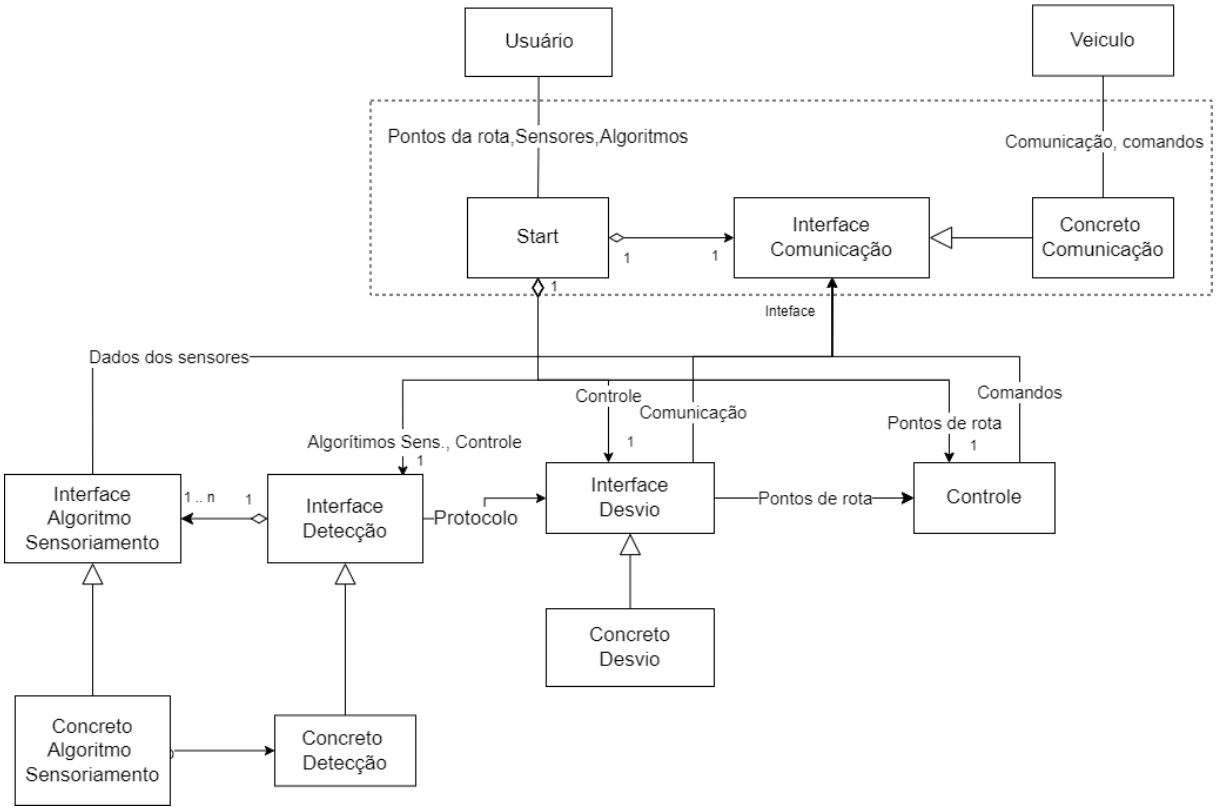

Figura 25: Relação entre classes do sistema *sense and avoid.*

Na Figura 25, as setas com pontas triangulares cheias representam uma associação entre as classes, ou seja, quando uma classe utiliza um objeto da outra, enquanto que as setas com os losangos vazios significam agregação, que é um caso específico de associação em que a classe agregada faz parte da classe agregadora e, assim, ambas compartilham o mesmo ciclo de vida. Por fim, as setas com pontas triangulares e sem preenchimento significam herança, o que ocorre quando uma classe herdada é uma extensão da outra, tendo as variáveis e métodos em comum.

Os princípios de padrões de *desing* utilizados seguem (ou são inspirados) no livro *Desing Pattern* (E Gamma, R Helm, R Johnson, 1994). Entre os padrões utilizados estão: interfaces para tornar a arquitetura modular; o padrão *Facade* para fornecer uma interface simples de comunicação entre o *framework*, o usuário e o veículo; e uma versão inspirada nos padrões *Observer e Strategy* para permitir que um número variável de objetos possa trabalhar em conjunto com um gerenciador. Nos

subcapítulos a seguir entra-se em detalhes sobre cada padrão utilizado e os motivos de suas escolhas.

5.3.1. Interfaces

Para que o sistema *sense and avoid* se tornasse modular, cada fase de seu processo teve de ser concebida de forma independente, trabalhando de modo a não necessitar reconhecer a implementação das demais fases, apenas se comunicando com elas por intermédio de uma interface fixa de comunicação.

Para permitir este requisito no *framework* proposto, as suas principais classes foram modeladas como interfaces, ou seja, elas são apenas uma estrutura a ser seguida para implementação da classe concreta. Deste modo, o sistema não fica ligado a uma implementação de código, mas apenas à modelagem de suas classes, permitindo trocas no código de cada classe, sem a necessidade de trocar o código fonte das demais classes.

Um exemplo de interface está ilustrado na Figura 26, que é um recorte da Figura 25. Nela exibe-se a relação entre as classes responsáveis da fase de desvio do *framework*, o qual foi implementada tendo em vista apenas a classe *Interface Desvio*. Na sua interface está toda a comunicação com as demais classes, porém esta é uma classe puramente abstrata, ou seja, não apresenta implementações de código, apenas as assinaturas de funções necessárias para o funcionamento da fase. Enquanto isso a classe *Concreto Desvio* é uma herança da interface e nela está contida a implementação do código de todas as funções presentes na interface.

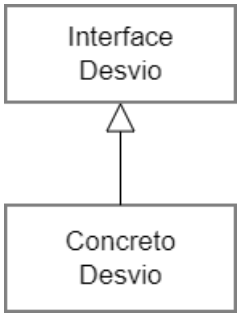

Figura 26: Exemplo de Interface do sistema *sense and avoid*.

Este padrão de *design* possibilita a troca de algoritmos de maneira simples, apenas alterando-se a implementação da classe concreta. Assim, com a utilização deste padrão de *design* tem-se a implementação dos requisitos de **modularidade**, **troca de algoritmo** e de **independente de aeronave**.

Este padrão de *design* foi utilizado nas fases “Desvio”, “Detecção” (Fusão de sensores), “Sensoriamento” e “Interface Comunicação”.

5.3.2. Façade

Para permitir uma interface simples de comunicação entre o usuário e o veículo foi utilizado o padrão Façade. Este padrão fornece uma interface unificada para um conjunto de interface nos subsistemas (E Gamma, R Helm, R Johnson, 1994). Para isso ele criar uma única classe com o papel de comunicar e gerenciar as demais classes. O cliente do sistema comunica-se com esta classe gerenciadora, que por sua vez repassa seus pedidos às demais. Desta forma, o Façade isola o cliente dos subsistemas.

No *framework* proposto neste trabalho existem dois clientes distintos, com objetivos e interações específicas. O usuário, responsável por iniciar o sistema e enviar suas configurações, descrevendo qual será a rota inicial, também tem a possibilidade de

personalizar o sistema, escolhendo quais os algoritmos serão utilizados a partir dos sensores disponíveis no veículo; e o veículo, que executará a rota enviada pelo controle do *framework* e irá, constantemente, enviar os dados de seus sensores para os algoritmos de detecção.

Deste modo serão necessárias duas interfaces distintas de comunicação com o *framework*, ou seja, duas classes do tipo Façade. No diagrama da Figura 25 elas são representas pela classe *Start* e *Interface Comunicação*. Cada uma destas classes comunica-se com os subsistemas *Detecção*, *Desvio* e *Controle* por meio de suas interfaces. Na Figura 27 apresenta-se a interação do usuário com a classe *Start* (um exemplo do padrão Façade), e esta, por sua vez, com as interfaces de outros subsistemas.

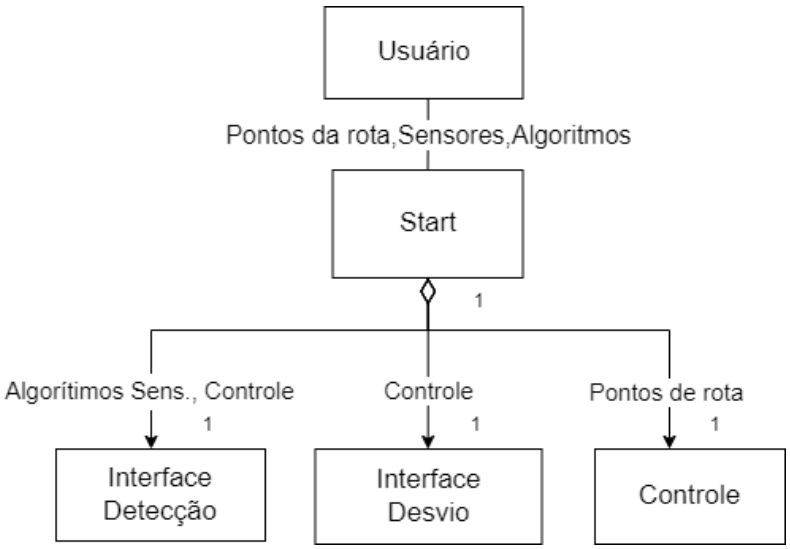

Figura 27: Exemplo do padrão Façade na arquitetura do sistema *sense and avoid.*

5.3.3. *Observer* invertido

A comunicação dos sensores com a detecção é uma das fases mais complexas do sistema *sense and avoid* implementado, pois um dos seus requisitos é permitir uma quantidade variada de sensores. Para que a arquitetura proposta seja robusta a essa personalização, preferencialmente permitindo a troca de sensores também em tempo de execução, é necessário a criação de um *design* específico. Existem dois padrões

de *design* clássicos que resolvem em parte este requisito, o *Strategy* e o *Observer*, porém nenhum deles resolve este requisito totalmente.

O *Strategy* (E Gamma, R Helm, R Johnson, 1994) tem como principal foco encapsular um algoritmo, permitindo que o sistema defina uma família de algoritmos similares que possam ser alterados em tempo de execução, algo similar ao que deve ocorrer no *framework* proposto neste trabalho. Porém, este padrão não permite que vários algoritmos trabalhem de forma independente ao mesmo tempo e com o mesmo conjunto de dados, nem cria uma maneira de comunicação com uma classe gerenciadora.

Por outro lado, o *Observer* (E Gamma, R Helm, R Johnson, 1994) é um padrão que define uma dependência de *um para muitos* entre objetos. Nele uma classe principal, chamada de *subject*, pode conter um número variado de objetos dependentes, chamados de *observer*. Este padrão visa facilitar a atualização do estado dos *observer* caso ocorra uma mudança na classe *subject*. Porém, no *framework* aqui proposto o caminho é inverso, ou seja, existe uma dependência de *muitos para um*, pois nele há somente um objeto da classe detecção, que é dependente de dados vindo de diversos objetos da classe de sensoriamento – tem-se uma detecção dependente de vários sensores.

Para resolver estes impasses no *framework* de *sense and avoid* foi criado um padrão alternativo, que unifica o *Observer* e a *Strategy*, funcionando com um padrão de dependência de *muitos para um*, no qual pode-se alterar o algoritmo em tempo de execução. Nele a classe observadora é a *Detecção*, e esta por sua vez contém uma lista de classes fornecedoras de dados, que são os objetos da classe algoritmo de sensoriamento. Os algoritmos, ao encontrarem um novo objeto em risco de colisão, enviam um sinal à classe *detecção* com os dados do objeto observado, que por sua vez processa esses dados em um algoritmo de fusão de sensores. Para que os objetos da classe de sensoriamento sejam capazes de enviar um sinal à classe de detecção, a classe de sensoriamento contém uma referência do objeto da classe detecção, podendo chamar a função de atualização da detecção. Na Figura 28 existe

um exemplo da relação entre as classes *Detecção* e *Sensoriamento*, exibindo a organização do padrão proposto.

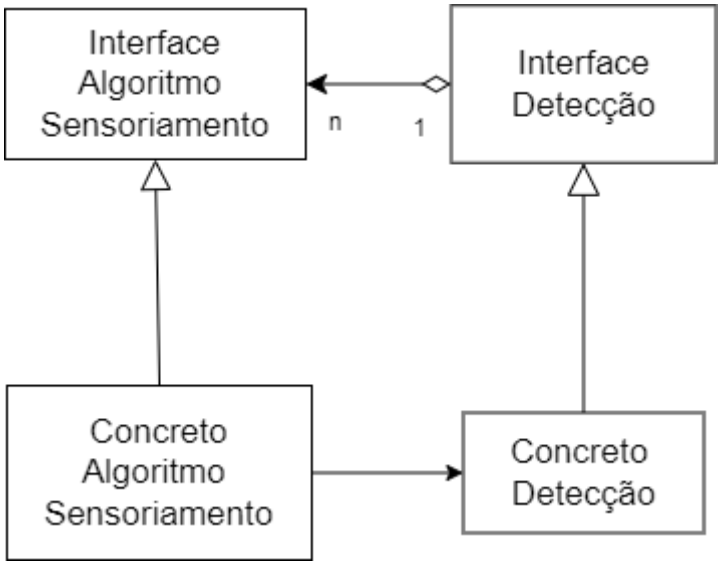

Figura 28: *Observer* invertido na arquitetura do *sense and avoid*.

5.4. Exemplo de execução do *framework* de *sense and avoid*

Visando facilitar a compreensão dos passos executados pelo sistema de *sense and avoid*, apresenta-se agora um exemplo de sua execução, explicitando como cada fase desempenha suas funções e interage com as demais fases do sistema.

A Figura 29 exibe um exemplo de todo o ciclo do sistema de *sense and avoid* proposto neste trabalho.

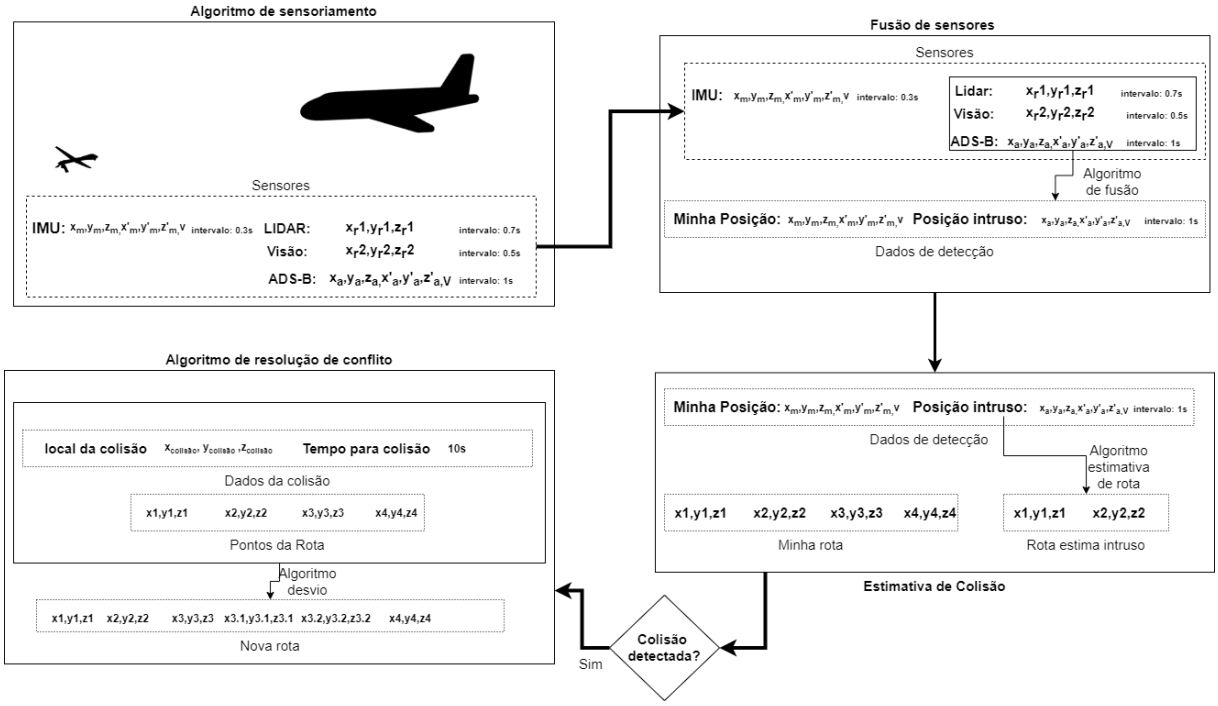

Figura 29: Exemplo de execução do sistema de *sense and avoid.*

No exemplo apresentado na Figura 29, o VANT embarca quatro diferentes tipos de sensores (quadro “Algoritmo de sensoriamento”), sendo que: a) um sensor, o IMU (*Inertial Measurement Unit*, ou Unidade de Medidas Inerciais) que são os dados capturados por sensores inerciais (acelerômetro e giroscópio) e representam a localização do próprio VANT, sua direção e sua velocidade (x_m,y_m,z_m,x'_m, y'_m,z'_m,v); b) outro sensor corresponde a um sensor de detecção cooperativa, o ADS-B, que recebe os dados da posição, direção e velocidade absoluta da aeronave intrusa (x_a,y_a,z_a,x'_a,y'_a,z'_a,v); c) os dois demais sensores são sensores de detecção não cooperativa, o LIDAR e a Visão (representando uma câmera), um de detecção ativa e o outro de detecção passiva, sendo que ambos capturam apenas a posição do objeto intruso relativa à posição do VANT (x_r,y_r,z_r). Também pode-se notar que cada sensor tem um intervalo de atualização diferente, variando entre 0,3 segundos (no IMU) a 1 segundo (no ADS-B).

Ao se detectar uma aeronave, na fase de sensoriamento, todos estes sensores enviam dados para a fase de fusão de sensores (quadro “Fusão de sensores”). Nesta

fase, um algoritmo é utilizado para aperfeiçoar os dados de detecção, retirando a redundância dos dados recebidos dos diversos sensores, transformando-os em apenas um dado de detecção mais acurado, contendo a posição, a direção e a velocidade da aeronave controlada (VANT), bem como a posição e a direção relativas do objeto intruso e sua velocidade. Um passo alternativo pode ser o enriquecimento dos dados de sensores não cooperativos, utilizando múltiplas detecções para calcular sua direção e velocidade.

Os dados de detecção unificados e aprimorados são enviados para a fase de estimativa de colisão (quadro “Estimativa de Colisão”), que os utiliza para estimar uma rota futura da aeronave intrusa, e une estes dados com a rota planejada para a aeronave controlada (VANT), objetivando encontrar um possível risco de colisão entre as duas aeronaves. Caso essa aeronave intrusa seja encontrada, estima-se a localização e o tempo para que a colisão aconteça.

Encontrado um ponto de colisão futura, e o tempo para que esta colisão ocorra, a última fase passa a atuar, que é a responsável por traçar uma nova rota (quadro “Algoritmo de resolução de conflito”). Esta nova rota deverá desviar a aeronave controlada o mínimo possível da sua rota inicial proposta, de preferência não alterando os pontos significativos da rota (“*way points*”) propostos inicialmente, apenas acrescentando um desvio, como exibido na Figura 30, na qual a nova rota tem apenas dois novos pontos, sem alterar os três antigos.

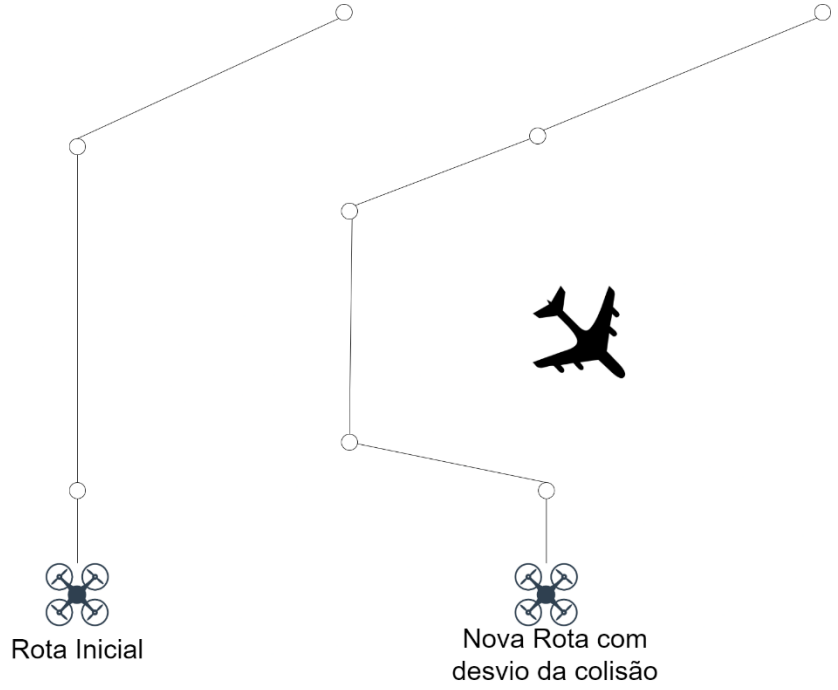

Figura 30: Exemplo de replanejamento de rota com dois novos pontos.

6. Testes e simulações

Objetivando avaliar a eficácia, eficiência e modularidade da arquitetura *sense and avoid* proposta, torna-se necessária a realização de testes, utilizando diferentes algoritmos de sensoriamento, de detecção de objetos (aeronaves intrusas) e de desvio de trajetória. Porém, para que esses testes ocorram de modo seguro, sem haver colisão entre aeronaves, eles devem ser realizados em simuladores antes de serem implementados em VANTs reais.

Desta forma, a criação de um simulador de voo para veículos autônomos é imprescindível para os testes da arquitetura *sense and avoid* proposta neste trabalho de pesquisa. Deve-se destacar que essas simulações de ações realizadas por veículos aéreos autônomos já são alvo de pesquisas há alguns anos, existindo vários simuladores em uso, tanto em nível comercial quanto em nível acadêmico.

Visando uma simulação a mais fidedigna possível da realidade é necessária a escolha do simulador que apresente as melhores características para testes de *sense and avoid* com múltiplos sensores. Isso posto, esse simulador deverá atender a alguns requisitos:

Física realista: Para que se consiga uma simulação realista é essencial que o simulador tenha capacidade de facilmente incluir cálculos físicos do modelo da dinâmica e cinemática do VANT, impendido, assim, que o veículo simulado faça manobras impossíveis de serem realizadas no mundo real.

Simulação de VANTs: Para a realização de testes que envolvam a possível colisão de aeronaves, além da dinâmica envolvida nas aeronaves e objetos é necessária uma representação física do corpo do veículo, que pode variar desde um pequeno VANT até a grandes aeronaves autônomas. Desde modo, o simulador dever ser capaz de replicar colisões entre um veículo aéreo controlado com outros objetos.

Simulação de sensores: As simulações dos comportamentos dos sensores são essenciais para os testes do *framework* proposto neste trabalho, considerando suas

diferentes fases. Para a fase de sensoriamento é necessário que seja possível simular diferentes tipos de sensores para detecção, sejam eles cooperativos ou não, e na fase de desvio é importante também a simulação de sensores inerciais.

Simulação do ambiente: Colisões de um VANT podem ocorrer não apenas com outras aeronaves, mas também com construções ou montanhas. Além disso, o ambiente pode ser um fator de grande impacto na qualidade de detecção. Assim, a possibilidade de modelá-lo é uma grande vantagem para os testes a serem realizados em simuladores.

Visualização gráfica realista: Alguns sensores necessitam de requisitos especiais para uma simulação eficaz. Um exemplo desses tipos de sensores são as câmeras, pois seus algoritmos são ligados a visualização do ambiente em sua volta, sendo necessárias visualizações gráficas que simulem o cenário e o ambiente da forma mais real possível.

Código aberto: Para que o simulador possa ser utilizando por um maior número de pessoas é importante que exista a possibilidade de edição e utilização do seu código, sendo possível implementar as alterações que serão necessárias para a criação do *framework* proposto neste trabalho. Estas são as características desejáveis de *softwares* de código aberto (*OpenSource*).

Simulação de comunicação: Para que os testes se tornem mais realistas é necessário também simular a comunicação envolvida. Neste caso, deve-se enviar e ler dados do simulador de forma similar à encontrada em VANTs reais. Por exemplo, em veículos pequenos, como *drones* quadrimotor, essa comunicação normalmente se dá por meio de um protocolo TCP/IP, denominado de MavLink, sendo interessante a implementação desde protocolo para testes de simulação da comunicação com estes veículos.

Para se encontrar um simulador aderente a esses requisitos foi realizada uma revisão na literatura dos principais simuladores robóticos, focando-se principalmente em simuladores para veículos aéreos autônomos que apresentassem ambiente gráficos

realista em três dimensões. Os simuladores foram analisados segundo os requisitos apresentados anteriormente.

6.1. Estado da arte em simulação empregadas em VANTs

Dentre os simuladores pesquisados, o Gazebo (Koenig and Howard, 2004, p. 2149–54) está entre o mais populares no meio robótico. Este simulador, um dos pioneiros na criação de ambientes realistas em 3D para interações com robôs, foi desenvolvido com uma representação fiel da dinâmica e da cinemática relacionadas com um robô, contando com um projeto modular que permite a criação de diversos sensores e mecanismos, como braços robóticos. Por ser um simulador de código aberto, tem grande utilização, sendo alvo de criação de vários *plug-ins* e funcionalidades incrementais.

Porém, como o objetivo do Gazebo não tem o foco na simulação de VANTs, torna-se necessário modelar a física e o corpo de aeronaves para simulá-las. Esse foi o foco do trabalho desenvolvido por Sendobry et al. em (Sendobry et al., 2012), no qual apresenta-se uma abordagem de simulação de VANTs do tipo quadrimotor, utilizando o Gazebo. Esse trabalho acrescenta ao simulador os cálculos físicos necessários para a dinâmica de voo de um VANT, bem como o sensoriamento e a representação do corpo deste tipo de aeronave. Porém, como o Gazebo conta com uma modelagem gráfica de certa forma ultrapassada, por não fazer representações visuais do ambiente e da aeronave realísticas, pode influenciar e até mesmo prejudicar testes que envolvam visão computacional.

Para cobrir a limitação visual encontrada no Gazebo, pesquisas mais recentes têm apresentado simuladores que se utilizam de tecnologias criadas para *engine de*

game[15], permitindo, assim, gráficos modernos e realistas, além de um modo simples de se criar cenários complexos. O Unreal, uma *engine de games* desenvolvida pela Epic Games, é utilizado em diversos trabalhos de simulação, por contar com modelagens gráficas no estado da arte atual. Dentre esses trabalhos, destaca-se (Savva et al., 2017, p. 1–14) no qual é descrito o MINOS (*Multimodal Indoor Simulator for Navigation in Complex Environments*), um simulador criado para testes de navegação e inteligência artificial em robôs, utilizando técnicas multissensoriais. A modelagem proposta nesse simulador permite uma fácil criação de múltiplos cenários de testes, com diversos níveis de complexidade, e por contar com um cenário realista, este simulador permite testes de visão computacional. Savva et al., em seu trabalho utiliza o MINOS para analisar quais conjuntos de sensores ou técnicas de inteligência artificial trazem melhores resultados em problemas de navegação. Ainda, para esse simulador, são implementados quatro diferentes tipos de sensoriamento: por visão, por profundidade, por toque (colisão com outros objetos) e por medidas de navegação (aceleração, velocidade, distância e direção). Porém, como esse simulador tem o foco apenas em robôs simples e para movimentação *indoor*, não conta com uma implementação de físicas de voo para aeronaves ou uma modelagem de ambientes externos.

O UE4Sim, desenvolvido por Mueller et al. (Mueller et al., 2017) é um outro trabalho de simulação utilizando a *Unreal Engine*. Ele utiliza também técnicas do estado da arte para “renderização” e criação de ambientes, construindo um cenário realista e com facilidade de alterações. Para este simulador os autores modelaram o corpo e a física de VANTs quadrimotores e carros, com o intuito de testar algoritmos de detecção e rastreamento por intermédio de visão computacional. Nesse trabalho foi apresentado um *benchmarking* de algoritmos do estado da arte em rastreamento

[15] Conjunto de programas e bibliotecas utilizados para simplificar a criação de jogos, que incluem, entre outros, um motor gráfico, responsável pela “renderizar” o ambiente 3D, e um motor de física responsável por simular a parte física do jogo.

(*tracking*), demonstrando bons resultados em testes utilizando *deep learning*. Porém, seu foco situa-se unicamente na detecção por meio de câmeras, deixando de lado demais sensores, como radares, sensores de profundidade e sensores cooperativos. Ainda, o desenvolvimento do UE4Sim é do tipo código aberto, sendo possível alterar seu código conforme as necessidades de novas pesquisas.

Por fim, o AirSim – *Aerial Informatics and Robotics Simulation* (Shah et al., 2017, p. 1–14) é um outro simulador do tipo código aberto que está sendo desenvolvido pela Microsoft para teste de inteligência artificial e navegação em VANTs e em carros autônomos. O AirSim tem também a sua essência criada com base no *Unreal Engine*, apresentando boa qualidade de gráficos e uma grande possibilidade de edição do ambiente. Este simulador conta com uma ótima simulação física de dinâmica e cinemática para quadrimotores, e tem como vantagens uma API para controle de VANT por meio do protocolo MavLink (*Micro Air Vehicle Link*), similar à utilizada em VANTs reais, permitindo que se utilize os mesmos comandos empregados em aeronaves reais. Em sua programação, o AirSim contém diversos tipos de sensores simulados, como câmeras visuais e de profundidade, além de sensores inerciais (acelerômetro, giroscópio, barômetro, magnetômetro e GPS). Apesar de ainda estar em fase de desenvolvimento, este simulador conta com uma comunidade de pesquisa ativa, com vários desenvolvedores criando novas funcionalidades constantemente.

6.2. Comparação entre simuladores

Para se comparar os simuladores estudados, tais simuladores foram avaliados conforme os requisitos apresentados no início deste capítulo. Na Tabela 2 encontra-se uma comparação entre os diversos simuladores citados, considerando as referências descritas.

	Física Realista	Simulação VANTs	Simulação Sensores	Simulação Ambiente	Gráficos Realistas	Código Aberto	Simulação Comunicação
Gazebo	X		X	X		X	
Sendobry	X	X	X	X		X	
MIMO	X		X		X		
UE4Sim	X	X		X	X	X	
AirSim	X	X	X	X	X	X	X

Tabela 2: Comparação dos simuladores nas referências consultadas

Também foram analisados os requisitos, separadamente, verificando seu comportamento entre os simuladores.

Física realista: Por ser um requisito crucial para uma simulação, a física está presente em todos simuladores analisados. O Gazebo e Sendobry têm a mesma origem e, deste modo, seus sistemas de simulação da física do ambiente são semelhantes, enquanto o MIMO, UE4Sim e AirSim são criados com base no Unreal, e utilizam o sistema de simulação desta *engine*.

Simulação de VANTs: Apesar de todos os simuladores terem um ambiente simples para criação de veículos e modelagem de sua física, neste trabalho o foco principal é a simulação de VANTs. Assim, simuladores que já contenham uma modelagem pronta de VANTs têm grandes vantagens. Dentre os simuladores analisados somente o Gazebo e o MIMO não têm uma modelagem previa de VANT, enquanto nos demais ele é representado por um veículo quadrimotor. Porém, atualmente nenhum dos simuladores apresenta diferentes modelos de veículos implementados.

Simulação de sensores: Nenhum desses simuladores tem todos os sensores implementados, porém alguns simuladores já apresentam modelos de sensores inerciais e câmeras visuais e de profundidade. O AirSim apresenta a maior variedade de sensores, contando mais recentemente com uma implementação do LIDAR.

Simulação de ambiente e Gráficos realistas: Apesar de o Gazebo possuir uma ferramenta simples de edição de ambiente, suas visualizações gráficas não são tão realísticas, enquanto isso os simuladores que têm compatibilidade com *engine de*

games (como a Unreal), apresentam gráficos no estado da arte de “renderização” e também contam com um sistema de edição de ambiente simples e completo.

Código aberto: Todos os simuladores apresentados foram utilizados em pesquisas científicas e têm parte de sua documentação aberta para acesso. Porém, não são todos esses simuladores que têm seus códigos liberados gratuitamente, e em alguns casos, como no exemplo do UE4Sim, apesar de estar disponível para pesquisas, este acesso tem que passar por avaliação interna do fabricante.

Simulação de comunicação: Neste quesito, o AirSim é o único simulador que a apresenta uma maneira de comunicação entre VANTs implementada, contendo uma interface compatível com o MavLink, que é um protocolo utilizando em vários VANTs domésticos, para comunicação e controle.

Analisando estes requisitos o AirSim mostrou-se o simulador mais completo, considerando os requisitos destacados e, dessa maneira, foi o simulador escolhido para a realização dos testes do *framework* proposto neste trabalho de pesquisa. Assim, este simulador é apresentado em detalhes a seguir.

6.3. AirSim

O AirSim (*Aerial Informatics and Robotics Simulation*) (Shah et al., 2018, p. 621–35) é um projeto de código aberto (*OpenSource*) da Microsoft para simulação de VANTs e carros. Ele foi construído com a intenção de ser uma ferramenta capaz de criar dados realistas para teste de inteligência artificial em veículos aéreos, provendo uma interface gráfica fidedigna e suporte a simulações *hardware-in-loop*, além de um protocolo de comunicação similar ao protocolo MavLink utilizado em VANTs reais e pelo controle Pixhawk, a controladora de voo mais utilizada em *drones*.

A construção do Airsim se dá sobre a Unreal, um motor de jogo no qual é possível construir cenários e personagens com gráficos de qualidade, programar a lógica de *games* e a interação entre os personagens e objetos dos cenários. O AirSim funciona

como um *plugin* do Unreal, beneficiando-se, assim, de todas as utilidades deste motor de jogo.

Por meio do simulador AirSim, um VANT é representado como sendo um personagem do cenário, no formato de um *drone* quadrimotor, tendo a sua física, a sua interação com o cenário e os seus sensores programados de forma que interajam com o ambiente desenhando no *Unreal*. O controle do veículo é dado por meio de um programa externo, chamado de *Companion Computer* (Computador Companheiro), ou por intermédio de um controle remoto. Um *plugin*, por sua vez, oferece uma interface de comunicação que permite ler dados dos sensores do veículo, como os vídeos de sua câmera, sua posição GPS, dentre outros, além de enviar comandos de movimentação ou ações no veículo.

A arquitetura do AirSim (Shah et al., 2017, p. 1–14) é modular, podendo ser dividida em sete grandes blocos: 1. Modelagem do veículo; 2. Modelagem do ambiente; 3. Modelagem física da dinâmica do veículo; 4. Modelagem dos sensores; 5. Representação do ambiente (*Engine Rendering*); 6. Controle; e 7. Camada da API. Essa arquitetura interage com o Computador Companheiro. A camada Renderização[16] das simulações do Ambiente (*Engine Rendering*) é criada pelo *Unreal Engine*, enquanto que as demais são criadas pelo *plugin* AirSim. Cada um desses blocos é apresentado em detalhes como subcapítulos na sequência. A Figura 31 ilustra a arquitetura do AirSim.

[16] Renderização é o método pelo qual se obtém o produto final de um processamento digital.

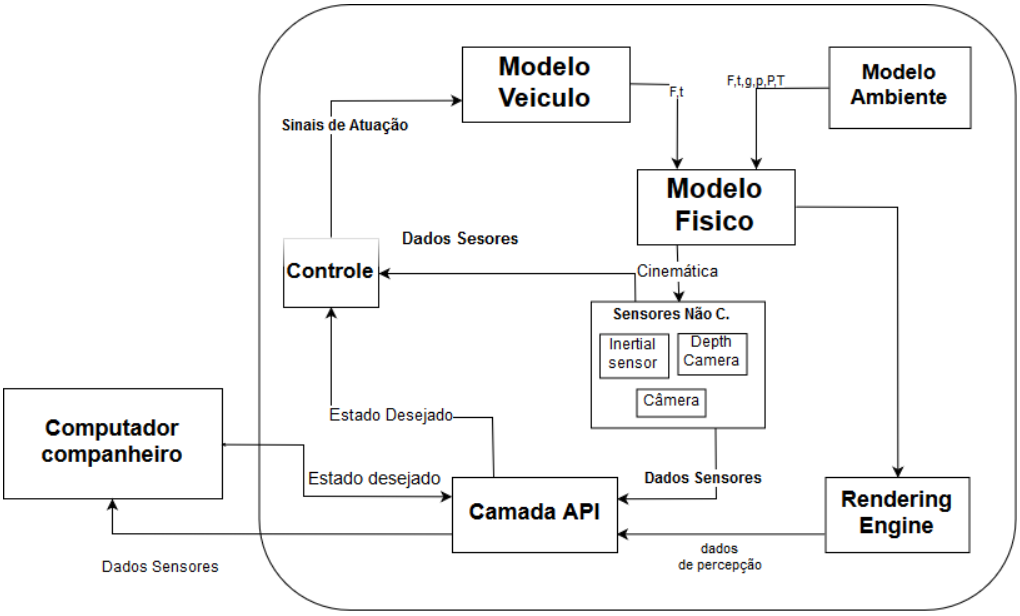

Figura 31: Arquitetura do AirSim. Adaptada de (Mueller et al., 2017, p. 1–14)**.**

6.3.1. Modelagem do veículo

O bloco “Modelo Veículo” é a camada responsável pelos cálculos físicos necessários para modelagem e dinâmica do veículo. No AirSim, o VANT é representado como um corpo rígido, contendo 4 atuadores de força e torque, que são os seus motores. Sua modelagem inclui valores de massa, inércia, coeficientes para arrasto linear e angular, além de valores de fricção. Um veículo é formado pelo conjunto de quatro vértices posicionados em suas extremidades, em que cada um tem sua entrada de controle u_i que impulsiona os vértices, gerando torque e força, conforme mostrado na Figura 32. Seu movimento é calculado por meio da Equação 5.

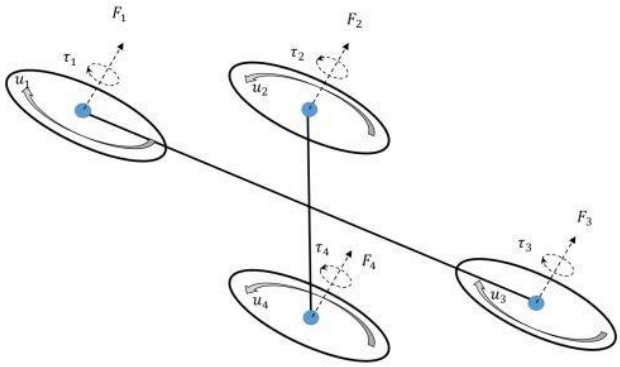

Figura 32: Modelagem física de um VANT no AirSim (Mueller et al., 2017, p. 1–14)**.**

$$F_i = C_\tau \rho \omega_{max}^2 D^4 \mu_i \qquad e \qquad \tau_i = \frac{1}{2\pi} C_{pow} \rho \omega_{max}^2 D^5 \mu_i$$

Equação 5: Fórmulas de tração e força das hélices de um VANT no AirSim.

Nessas fórmulas, o C_T e C_{pow} são, respectivamente, os coeficientes de impulso e força baseados nas características físicas do propulsor, ρ é a densidade do ar, D é o diâmetro do propulsor e ω_{max} é a velocidade angular máxima em rotações por minuto.

6.3.2. Modelagem do ambiente

Para uma modelagem da dinâmica do voo de um VANT é necessário calcular os fenômenos físicos aos quais o veículo está exposto. O bloco “Modelo Ambiente” corresponde à camada de modelagem do ambiente e é responsável por isso, possuindo modelos acurados capazes de executar operações em tempo real. A seguir seguem detalhes a respeito do modelo físico utilizado pelo AirSim.

Gravidade: Para o cálculo da gravidade é utilizada a fórmula da gravidade de Newton, apresentada a seguir, na Equação 6. Desse modo, tem-se a gravidade atrelada à altura do veículo e à circunferência da Terra.

$$g = g_0 \cdot \frac{R_e^2}{(R_e + h)^2} \approx g_0 \cdot \left(1 - 2\frac{h}{R_e}\right)$$

Equação 6: Fórmula de simulação da gravidade

em que *R* é o raio da Terra e g_0 é a constante gravitacional medida na superfície da Terra.

Campo magnético: As modelagens mais fiéis do campo magnético são complexas, incapazes de serem processadas em tempo real. Desse modo, no AirSim optou-se por um modelo mais simples, porém que apresenta bons resultados. Para isso foi utilizado um modelo que assume que a Terra é uma perfeita esfera dipolar.

Pressão e densidade do ar: O cálculo da pressão *P* e da temperatura *T* do ar são dados com referência na altura dos veículos, utilizando para isso uma tabela com base nas diferentes camadas da atmosfera. Assim, a pressão é calculada a partir da fórmula dada de modelos atmosféricos[17]. Ainda, a densidade do ar é calculada a partir da fórmula $\rho = \frac{P}{R.T}$ (em que *R* é a constante do gás).

6.3.3. Modelagem física da dinâmica do veículo

A modelagem física do veículo é responsável por calcular o próximo estado da cinemática do VANT, expressa por 6 valores: posição, orientação, aceleração linear, velocidade linear, aceleração angular e velocidade angular. Estes valores também são conhecidos como 6-Dof (*6 degress-of-Freedom*). Seus valores de saída são utilizados tanto para desenhar o novo estado do VANT no ambiente gráfico, quanto para servir de entrada para os sensores simulados no AirSim.

---

[17] Modelos atmosféricos: disponível em http://www.braeunig.us/space/atmmodel.htm, acessado em 11/2018

Este bloco calcula cada estado de cinemática do VANT a partir dos dados de força e torque recebidos da modelagem física do VANT, e dos valores da gravidade, campo magnético, pressão, temperatura e densidade do ar recebidos da modelagem física do ambiente. Estes valores são utilizados para calcular o arrasto angular e linear, a aceleração, a integração e as possíveis colisão do VANT. Os resultados dessas fórmulas são usados tanto para o cálculo do novo estado do veículo, quanto como entrada de dados para os sensores.

6.3.4. Modelagem dos sensores

O AirSim conta com sensores programados para o veículo, permitindo uma boa representação do voo de um VANT, sendo que todos eles estão detalhados a seguir. Porém, para uma simulação mais realista do espaço aéreo são necessárias simulações de sensores cooperativos, como ADS-B e TCAS, ambos amplamente utilizados na aviação. Estes sensores foram acrescentados ao AirSim como parte do presente trabalho de pesquisa, e são detalhados no próximo capítulo.

Barômetro: O barômetro utiliza como entrada a pressão do ar, calculada no modelo implementado para a modelagem do ambiente, e calcula seu desvio utilizando um processo gaussiano *markoviano*.

Giroscópio e acelerômetro: Estes sensores constituem o IMU (Unidade de Medidas Inerciais), um dos sensores mais comuns utilizados em VANTS. Seus valores são calculados por meio dos dados de aceleração retirados do modelo físico do VANT, acrescentando-se o ruído branco.

Magnetômetro: O cálculo do magnetômetro é feito a partir dos dados calculados pela fórmula usada pelo campo magnético, acrescentando-se o ruído branco.

GPS: O GPS simula a latência (em torno 200ms) e a menor taxa de atualização (em torno de 50Hz), além do erro acumulativo horizontalmente e verticalmente.

Câmeras: As imagens de câmeras providas pelo AirSim são geradas por meio do Unreal, este ambiente simula o ambiente, a partir de algoritmos de modelagem 3D. O AirSim disponibiliza duas câmeras visuais posicionadas lado-a-lado na parte frontal do VANT. Além disso, ele também disponibiliza uma visão de profundidade e uma visão utilizando segmentação dos objetos na imagem.

LIDAR: O LIDAR é o sensor incluído recentemente no AirSim, e é simulado criando-se uma nuvem de pontos, que são criadas a partir de simulações de pulsos de laser. Ao habilitar o uso do LIDAR, o usuário pode personalizar dados sobre o ângulo do seu campo de visão, o número de canais, o número de pontos enviados por segundo e o número de rotações por segundo.

6.3.5. Representação do ambiente (*Engine Rendering*)

A renderização é responsável por criar a parte gráfica do simulador, essencial para a criação das imagens capturadas pela câmera. Esta camada é implementada pelo motor gráfico do *Unreal Engine.* Para isso, ela utiliza-se da localização do VANT calculada pela camada da modelagem física.

6.3.6. Controle

O bloco “Controle” é responsável por receber comandos do bloco API e enviar intenções de movimento ao VANT. Em outras palavras, ele recebe comandos da camada API e os traduz para ações de alteração de estado no modelo físico do veículo.

Esta camada emula uma controladora PixHawk[18], podendo ser substituída pela mesma para utilização em veículos reais. Sua função é, a partir de dados do estado

[18] PixHawk é uma controladora e piloto automático para VANTs, presente nos veículos de diversas empresas. Suas especificações podem ser encontradas no site oficial pelo link (https://px4.io/).

atual do veículo, por meio dos seus sensores e de um estado futuro desejado, calcular qual a alteração necessária em cada motor do VANT para que seja possível atingir o novo estado com estabilidade.

6.3.7. Camada da API

O bloco “Camada API” permite a simulação da comunicação com o VANT. Com ela é possível conectar-se com o veículo, enviar comandos e receber dados lidos dos sensores, como o estado do veículo ou imagens.

Os dados dos sensores são recebidos da camada de modelagem dos mesmos e enviados continuamente para o “Computador Companheiro”. Este, por sua vez, é capaz de enviar comandos com pedidos de mudança de estado, que podem ser de movimentação, de acionamento ou de finalização de algum sistema no veículo. Esses comandos são enviados à camada de Controle para que possam ser aplicados no VANT.

A comunicação entre a “Camada API” e o “Computador Companheiro” se dá por meio de um protocolo TCP/IP, enviando comandos MavLink a partir dos quais é possível se conectar com veículo, enviando e recebendo dados por intermédio de mensagens “fortemente tipadas”.

6.3.8. Computador Companheiro

O “Computador Companheiro” é um agente externo ao AirSim com a capacidade de controlar o veículo, recebendo os dados dos sensores e enviando quais são as intenções de movimento necessária para que o veículo atinja a posição final desejada.

Este bloco pode ser visto como o cérebro do veículo, pois nele se encontra toda a lógica do voo, seja por intermédio de algoritmos de voo autônomo ou fornecida por um usuário externo ou por controle manual de um usuário. Ainda, nesse bloco, podem ser processados os ainda dados dos sensores em busca de um objeto em risco de

colisão com o VANT, e também pode ser calculada a rota de fuga da colisão por parte do veículo sendo controlado.

O “Computador Companheiro” comunica-se com a “Camada API” por meio de um protocolo TCP/IP. Deste modo, a comunicação é invariante da linguagem utilizada, ou do sistema presente no computador. No *link*[19] do AirSim são apresentados alguns exemplos utilizando a linguagem C++ e Python. Ambas as linguagens foram utilizadas em testes realizados no presente trabalho.

[19] Disponível em https://github.com/Microsoft/AirSim, acessado 11/2018

7. Estratégias de testes

Após a realização da modelagem da arquitetura do *framework* proposta, e tendo-se escolhido o simulador a ser utilizado, é necessária a criação de uma estratégia de testes que permita analisar o comportamento do *framework* diante dos requisitos apresentados no subcapítulo 5.1.

Para tanto foi criando um sistema que implementa o *framework* apresentado. O sistema foi programado seguindo o diagrama apresentado no subcapítulo 5.3 (arquitetura detalhada do *sense and avoid*) e utilizando a linguagem Python[20]. Foi implementada, ainda, uma interface de comunicação entre o *framework* e o AirSim, permitindo o seu uso para simulações de voo com um *drone*.

Por fim, foram organizados cenários de testes que permitem analisar todos os requisitos desejados: a modularidade do sistema, a possibilidade de troca de algoritmos, a capacidade de uso de diversos sensores, a independência da aeronave considerada e a simplicidade da interface com o usuário.

7.1. Arquitetura dos testes

Para a realização de testes é necessário que o *framework* se comunique de forma simples e eficiente com o simulador. Assim, foi necessário que as arquiteturas do simulador AirSim (apresentada no subcapítulo 6.3) e do *framework* (apresentada no subcapítulo 5.3) passassem a constituir um único sistema.

Neste novo sistema, o *framework* assume o papel do “Computador Companheiro” dentro da arquitetura do AirSim (vide item 6.3.8), pois ele será responsável por criar a lógica de voo da aeronave simulada, enviando os comandos de controle da aeronave. Por consequência, tornou-se necessária a criação de uma classe específica de

[20] O código está disponível em git@github.com:viniciusmdalmeida/Mestrado.git

comunicação, denominada “Concreta Comunicação”, a qual herda as funções abstratas da “Interface Comunicação”, adaptando-as para as peculiaridades da comunicação com AirSim, conforme Figura 33.

A arquitetura do sistema concebido está representada pela Figura 33, que pode ser dividida em três componentes: O **Usuário**, o **Framework** e o **Simulador (AirSim)**. O **Usuário** tem o papel de piloto remoto e é responsável por enviar para a aeronave simulada a rota a ser executada, bem como a escolha dos sensores e dos algoritmos utilizados para o sistema *sense and avoid*. O **Framework** é responsável pela implementação do sistema de *sense and avoid* proposto neste trabalho, utilizando para isso os dados dos sensores e dos algoritmos passados pelo usuário para criar uma versão personalizada do *framework*. O simulador **AirSim** é o componente responsável pela simulação, modelando o cenário do teste e as aeronaves, com suas mecânicas e cinemáticas, e recebendo do *framework* os dados de controle do VANT para transformá-los em ações para as aeronaves.

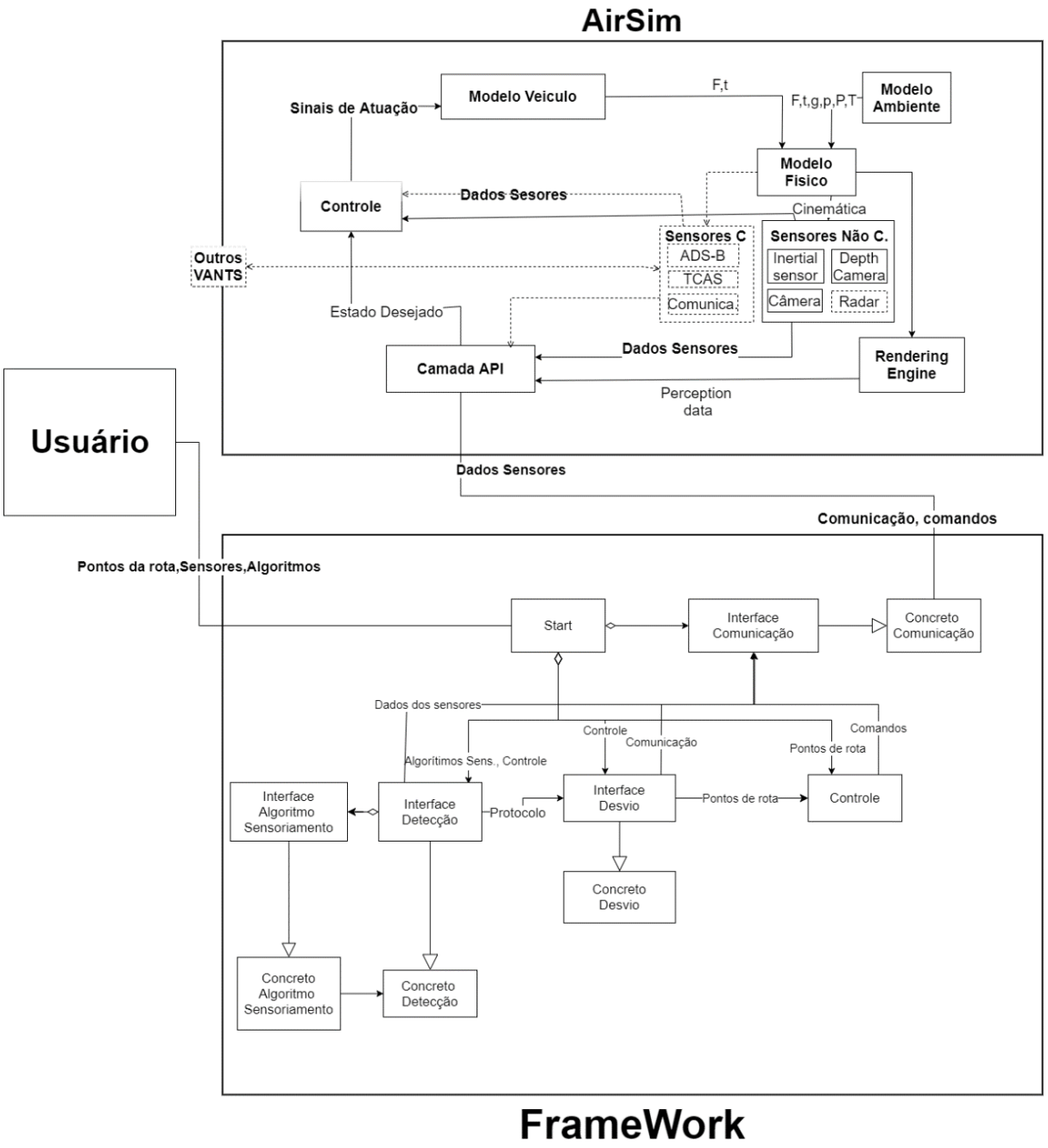

Figura 33: Arquitetura do sistema de testes do *sense and avoid*: O bloco à esquerda apresenta o Usuário interagindo com a arquitetura do *Framework* (bloco abaixo) - esta arquitetura está detalhada no capítulo 5. Por sua vez o *Framework* interage trocando dados com o AirSim (bloco acima) - sua arquitetura é detalhada no capítulo 6 [Próprio].

Deve ser salientado que os sensores cooperativos não estão representados na arquitetura original do AirSim. Porém, para uma simulação com diversidade de modelo

de detecção (cooperativa e não cooperativa) foi necessário que o veículo tenha sensores que permitam comunicação. Assim, para a realização dos testes aqui apresentados foram acrescentados ao AirSim códigos que simulam o sistema ADS-B.

7.2. Cenários de testes

Os testes foram conduzidos criando-se uma simulação em um ambiente 3D utilizando a *Unreal Engine*, em sua versão 4.22. No ambiente simulado, existem sempre apenas dois objetos: o veículo controlado, no caso um **VANT quadrimotor**, e o veículo em rota de colisão, um **avião** modelo Boeing 737.

Os cenários foram modelados de modo que ambas as aeronaves tenham uma rota retilínea e em todas as execuções gerem uma colisão frontal entre o VANT e o avião. As rotas de ambas as aeronaves se cruzam com um **ângulo máximo de 40º**.

Para se calcular uma rota de colisão foi criada uma função que recebe quatro parâmetros: o **ponto de colisão**, o **ângulo** θ entre suas rotas, o **tempo para colisão** (*t*) e **velocidade** (*v*) da aeronave. A partir destes parâmetros, esta função calcula o ponto inicial do avião e sua direção. Conforme ilustrado na Figura 34.

Para cada teste de colisão foram gerados 9 cenários de testes distintos, nos quais se altera o ângulo θ de incidência entre a rota do VANT e a da aeronave, variando entre 40º e -40º, de 10º em 10º, gerando cenários com ângulos de incidência de -40º, -30º, -20º, -10º, 0º, 10º, 20º, 30º e 40º. A partir do ângulo de incidência estipulado é calculada a posição e direção inicial da aeronave, que permite que a colisão entre a aeronave e o VANT ocorra exatamente no ponto de impacto pré-determinado. Em cada cenário foram repetidos os testes 3 vezes, alterando-se os valores de posição inicial do veículo controlado, bem como acrescentando-se pequenas variações no movimento do veículo, seguindo uma distribuição normal. Assim, para cada conjunto de algoritmos, foram realizados 27 testes.

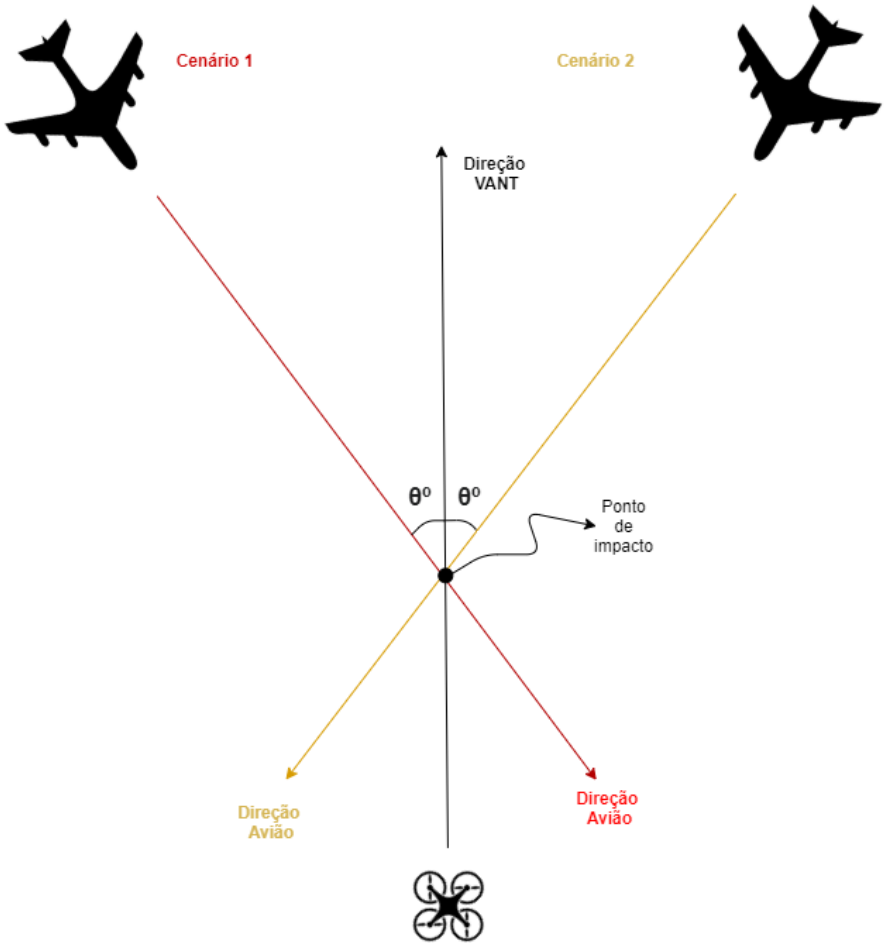

Figura 34: Representação de dois cenários de teste, demonstrando a rota de colisão entre as aeronaves e os ângulos de incidência -θ e θ [Próprio].

A imagem apresentada na Figura 35 ilustra um instante de uma simulação na qual uma aeronave está em rota de colisão com um VANT (*drone* quadrimotor). Nesta mesma imagem é possível observar o algoritmo de detecção visual em funcionamento (quadro dentro da figura, com retângulo indicando a detecção de uma aeronave).

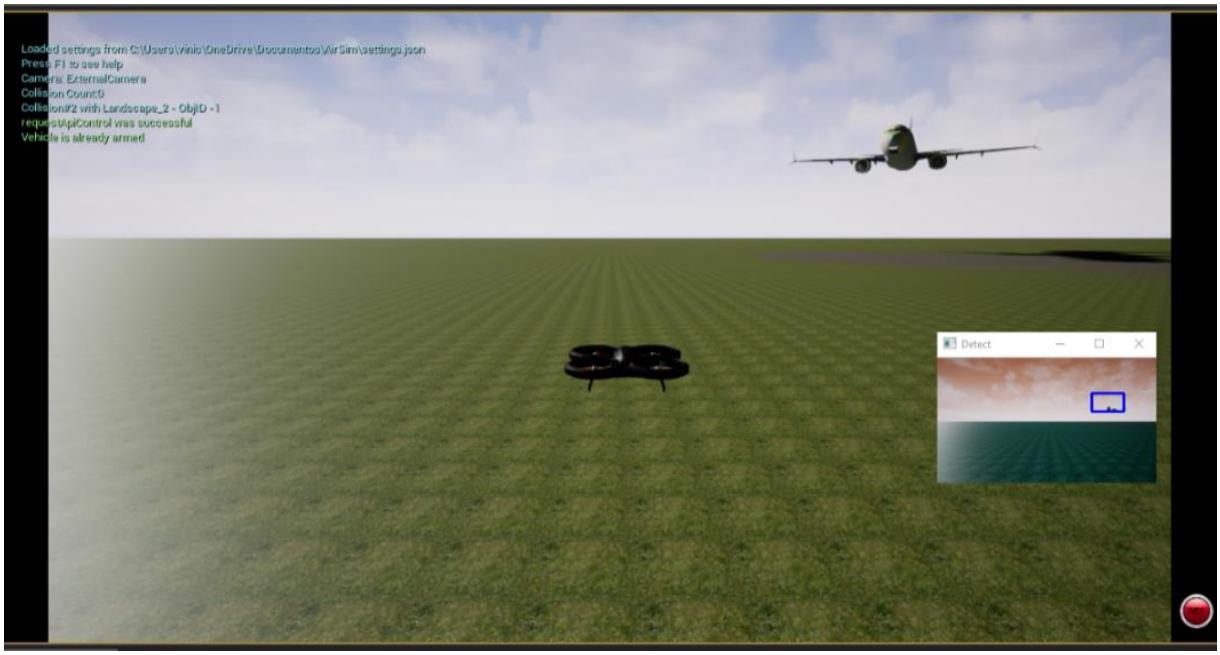

Figura 35: Ilustração do ambiente em funcionamento da simulação via *framework*. No quadro à direita da imagem mostra-se a detecção visual da aeronave por parte do VANT [Próprio].

Visando avaliar a capacidade de modularização do *framework* e de sua adaptação para diferentes sensores, bem como a possibilidade de utilização de vários algoritmos dentro do mesmo, foram desenvolvidos diferentes algoritmos para os testes, sejam eles de detecção, fusão de sensores ou replanejamento de rotas. Para avaliá-los foram realizadas três baterias de testes, cada uma referente a uma das fases do *sense and avoid.* Em cada fase, ao menos dois algoritmos diferentes foram utilizados para se testar a capacidade de intercâmbio de diferentes algoritmos pelo *framework*.

As baterias de testes foram dividas segundo a fase analisada do sistema. São elas: a **Detecção**, a **Fusão** de algoritmos de sensores e o **Desvio**, realizadas nesta ordem. Os testes foram executados de forma hierárquica, de forma que apenas os algoritmos com melhores performances na fase anterior foram analisados na fase seguinte. Todas as baterias de testes foram realizadas com o mesmo cenário apresentado no subcapítulo anterior. Porém, cada bateria contou com formatos diferentes de análise de performances.

7.2.1. Fase de detecção

Os testes dos algoritmos de detecção têm como objeto reconhecer qual conjunto sensor/algoritmo apresenta maior acurácia ao encontrar uma aeronave. Foram testados três tipos de sensores: **ADS-B**, **câmera visual** e **câmera de profundidade**.

Algoritmos

A detecção baseada no sensor ADS-B foi testada apenas com um algoritmo, baseado em regras geométrica. Enquanto isto, na detecção baseada em câmeras (visual ou de profundidade), foram testados diversos algoritmos, sendo cinco deles baseados em aprendizado de máquina e outros quatro baseados em algoritmos de rastreamento. Devido ao fato de ambas as câmeras apresentarem uma detecção baseada em visão computacional, é possível aplicar um mesmo algoritmo em ambas, porém os resultados da detecção se tornam diferentes em cada tipo de câmera.

Vale ressaltar que a câmera visual, quando utilizada individualmente, tem a capacidade de fazer uma detecção apenas em duas dimensões, calculando somente a posição vertical e horizontal da aeronave. Assim, não é capaz identificar a distância (profundidade) do objeto detectado, que é um dado crucial para o sistema *sense and avoid*. Devido a esta limitação nos testes envolvendo a câmera visual, na detecção com este tipo de sensor sempre será utilizada a visão binocular, utilizando duas câmeras de mesmo tipo, deslocadas horizontalmente por 25 cm.

Também é aplicada uma versão de detecção utilizando os sensores visuais em conjunto, ou seja, a câmera visual junto com a câmera de profundidade. Neste modelo, a detecção do algoritmo se dá por meio da câmera visual, pois este tipo de câmera possui melhor resolução, enquanto que a câmera de profundidade é utilizada como sensor auxiliar, apenas para o cálculo da distância do VANT à outra aeronave, quando esta aeronave tiver sido previamente detectada pela câmera visual. Este processo ocorre replicando-se a detecção da câmera visual na imagem da câmera de profundidade, e calculando-se o ponto da aeronave detectada de menor distância no

interior da área detectada (o *bound box representado pela caixa vermelha)*, conforme ilustrado na Figura 36.

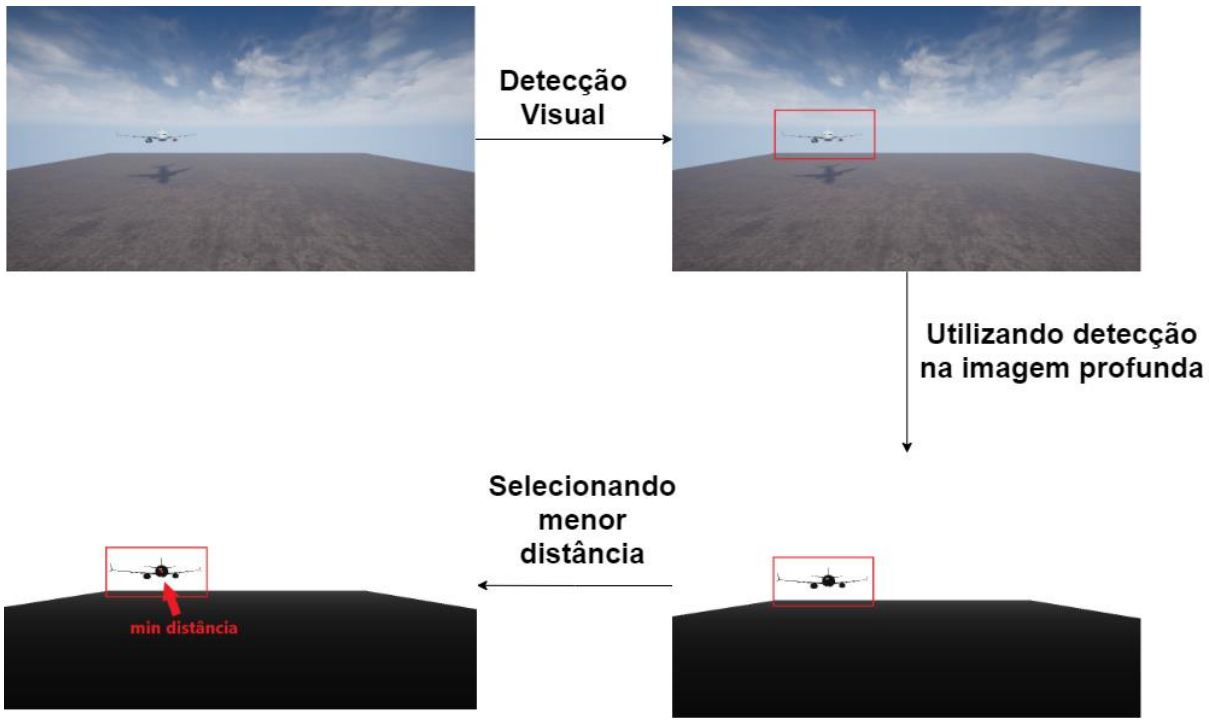

Figura 36: Detecção visual com auxílio de câmera de profundidade para calcular a distância do VANT à outra aeronave. Na figura pode ser visto o processo de encontrar um objeto na imagem de detecção visual, e projetar essa detecção para a imagem de detecção de profundidade onde será calculado o ponto de menor distância [Próprio].

Os algoritmos testados estão listados a seguir:

- **Câmera Visual e Câmera de Profundidade**
  - Para a técnica de aprendizado de máquina, foram utilizados os seguintes algoritmos:
    - SVM
    - *Random Forest*
    - *Ligthgbm*
    - *Percetron* multicamadas
    - *Naive Bayes*

    - Como algoritmos de rastreamento utilizou-se:
        - MIL
        - KFC
        - TLD
        - *Boosting*
- **ADS-B**
    - Para o ADS-B foi utilizada:
        - Técnica de detecção cooperativa

Entre os algoritmos de detecção visual estão dois tipos distintos de algoritmos: os algoritmos de **aprendizado de máquina** e os algoritmos de **rastreamento**. Os algoritmos de aprendizado de máquina fazem detecções a cada *frame* capturado. Desta maneira, eles são algoritmos que demandam um maior processamento e são mais inconstantes, podendo alternar entre detecções corretas e detecções erradas a cada *frame*. Por outro lado, os algoritmos de rastreamento utilizam detecções anteriores como base para as detecções futuras, demandando assim menor poder de processamento, e tendem a ser mais assertivos nas suas respostas, por utilizarem dados de *frames* anteriores para a próxima detecção; porém, nesses algoritmos é necessária uma detecção *a priori*, ou seja, uma primeira detecção vinda de outro algoritmo, podendo advir dos próprios algoritmos de aprendizado de máquina. Devido a esta limitação dos algoritmos de rastreamento, estes sempre foram testados em conjunto com um algoritmo de aprendizado de máquina, funcionando como um algoritmo auxiliar.

Avaliação

A avaliação da fase de detecção foi baseada no erro entre a posição detectada da aeronave e sua posição real. A cada detecção foi calculada a distância euclidiana, em metros, entre as duas posições (a posição detectada e a posição real). A partir dessa distância foi verificado se o objeto foi encontrado corretamente ou não. Foi considerada uma detecção correta aquela em que a distância entre a posição real e a detectada é menor do que 100 metros, enquanto que para distâncias maiores do que

100 metros foram consideras detecções erradas. A partir deste cálculo, foram obtidas quatro métricas de avaliação para cada algoritmo, que são:

- **Média do erro:** Também denotada por μ, a média do erro é calculada por meio da média da distância entre a posição da aeronave detectada e a posição real da aeronave, em todos os *frames*. Porém, no cálculo só se leva em consideração detecções consideradas corretas, ou seja, detecções nas quais essa distância é menor do que 100 metros. Dessa forma, a média do erro indica a precisão do algoritmo.
- **Taxa de verdadeiro positivo:** Representam a taxa de detecções consideradas corretas considerando todas as detecções feitas pelo algoritmo, ou seja, a porcentagem de detecções no qual o erro é menor que 100 metros. Taxas altas de verdadeiros positivos significam que o algoritmo tem grande cobertura de detecção correta na maioria dos *frames*. Essa taxa é representada em termos de porcentagem.
- **Taxa de falso positivo:** Representam a porcentagem de detecções consideradas inválidas, ou seja, aquelas que apresentam distância maior do que 100 metros entre as posições real e detectada da aeronave. Taxas de falso positivo altas sinalizam que o algoritmo tem a tendência de detectar aeronaves em locais onde elas não estão. Nestes casos, houve uma detecção, porém, essas detecções estão erradas.
- **Taxa de falso negativo:** Assim como os falsos positivos, a taxa de falsos negativos também calcula uma porcentagem de erro, porém neste caso o erro se dá por não haver uma detecção onde existe uma aeronave. Dessa forma, essa taxa calcula a porcentagem de *frames* contendo aeronaves nos quais não foi identificada a aeronave pelo algoritmo. Algoritmos com altas taxas de falso negativo podem significar *overfit*, que ocorre quando o algoritmo está tão otimizado para um determinado conjunto de dados que não reconhece dados de mesma classe fora do conjunto de treinamento.

7.2.2. Fase de fusão de sensores

Após uma análise individual do melhor algoritmo encontrado em cada sensor, foi feita uma análise comparativa entre quais grupos de sensores apresentam melhores resultados caso sejam utilizados em conjuntos, e qual é o melhor algoritmo de fusão destes dados.

Algoritmos

Para os testes foram analisadas algumas combinações de sensores. Os casos marcados como Misto utilizam, simultaneamente, câmera de visão e câmera de profundidade, conforme listado a seguir:

- Visão binocular + Câmera de profundidade (**BvCp**)
- Misto Câmeras + Câmera de profundidade **(CmCp)**
- Misto Câmeras + ADS-B **(CmAd)**
- Visão binocular + ADS-B (**BvAd**)
- Visão binocular + Câmera de profundidade + ADS-B (**BvCpAd**)
- Visão binocular + Câmera de profundidade + Misto Câmeras + ADS-B (**BvCpCmAd**)

Em todos os testes também foram utilizados os sensores inerciais e os sensores de colisão. Porém, o foco desses dois tipos de sensores não é a detecção do objeto em risco de colisão com o VANT, e sim a medida do movimento do VANT ou a ocorrência de uma colisão do VANT com a aeronave “intrusa”. A Figura 37 representa a estrutura final do *framework* ao utilizar câmeras visuais e ADS-B como sensores.

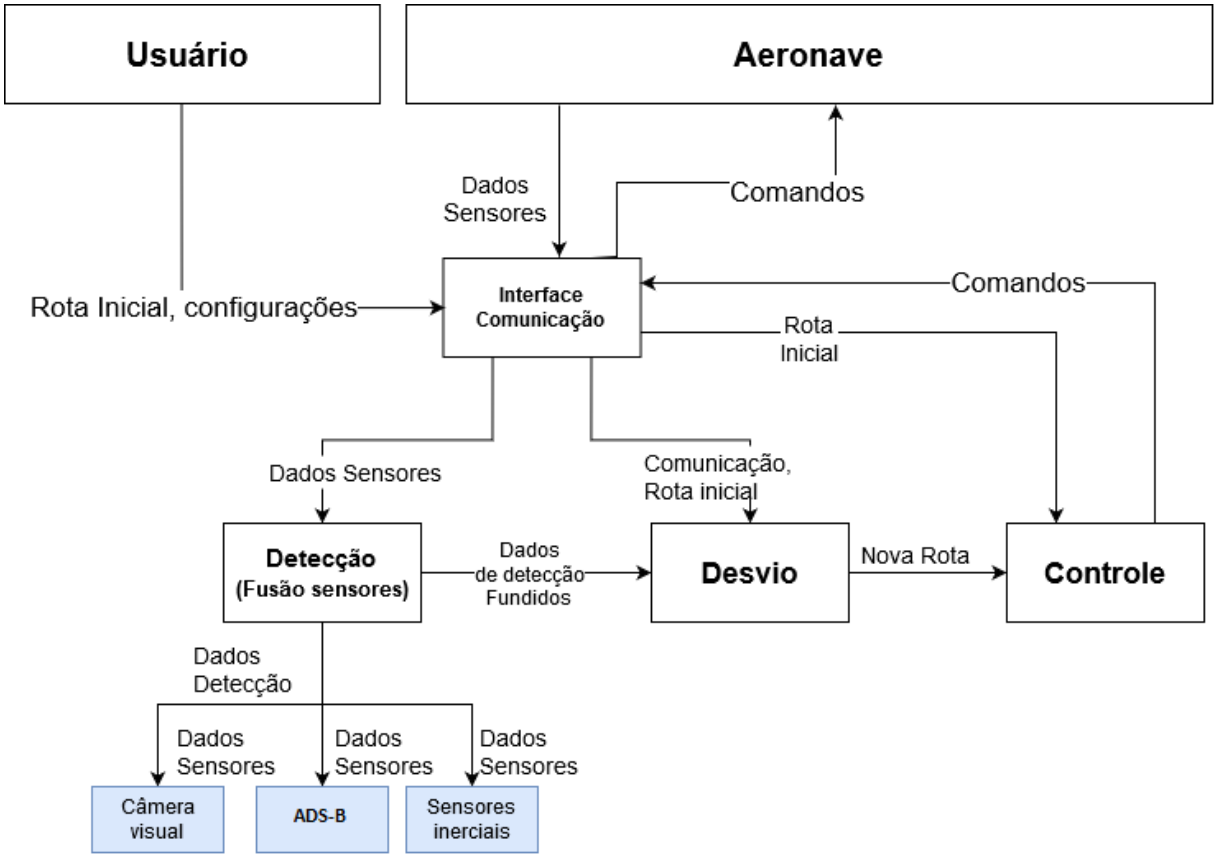

Figura 37: Arquitetura do ambiente *sense and avoid* utilizada nos testes [Próprio].

Avaliação

Dois algoritmos diferentes foram utilizados para os testes de fusão de sensores. São eles:

- **Média simples:** Nesse cenário, caso dois sensores identifiquem a aeronave no mesmo instante, a posição final será calculada a partir da média da posição encontrada por cada sensor.
- **Média ponderada:** Assim como no caso da média simples, a posição final é calculada a partir da média da posição encontrada por cada sensor, porém, neste caso, a média será balanceada com base na precisão (taxa de erro médio) de cada sensor utilizado. Assim, a detecção feita por um sensor com grande precisão terá peso maior do que a detecção efetuada por um sensor menos preciso.

As métricas de análise de performance utilizadas para avaliar os algoritmos de fusão de sensores foram as mesmas utilizadas para avaliar a detecção: média de erro, taxa de verdadeiro positivo, taxa de falso positivo e taxa de falso negativo.

7.2.3. Fase de desvio

O desvio é a última fase avaliada, sendo que nela são avaliados, em conjunto, os algoritmos de estimativa de rota da aeronave e os algoritmos de resolução de conflito. Sua métrica de avaliação será diferente dos demais testes, pois nesta fase será avaliada a porcentagem de cenários de teste em que a colisão iminente entre as aeronaves foi resolvida. Ou seja, em um caso no qual o resultado seja 100%, os algoritmos utilizados foram capazes de resolver todos os cenários de colisão e, assim, não ocorreu nenhum impacto entre a aeronave e o VANT controlado, enquanto que um resultado de 0% indicaria que todos os testes realizados levaram a um cenário no qual as aeronaves colidiram.

Apenas os algoritmos que representaram melhores resultados nos testes de detecção foram analisados na fase de desvio, porém foi utilizado um algoritmo para cada combinação de sensor possível, incluindo um algoritmo para cada sensor trabalhando individualmente.

Dois algoritmos de resolução de conflito diferentes foram aplicados nesta fase. Apesar de se utilizar uma estratégia similar entre ambos, eles diferem pela direção utilizada para o desvio. São eles:

- **Desvio horizontal:** Neste modelo, ao encontrar um cenário de colisão, o VANT controlado desvia horizontalmente no sentido oposto à aeronave intrusa até o cenário ser corrigido.
- **Desvio vertical:** Este modelo é semelhante ao que realiza o desvio lateral, porém neste caso os desvios são verticais e, assim, o VANT poderá mover-se

para baixo caso a aeronave intrusa esteja acima dele, ou caso contrário, para cima, caso a aeronave intrusa esteja abaixo dele.

7.3. Captura de dados de treinos

Grande parte dos algoritmos de detecção é baseada em técnicas de aprendizado de máquina. Estes algoritmos funcionam de forma a otimizar os parâmetros de uma função a partir de análises de dados coletados previamente, podendo se dizer que eles “aprendem” uma função a partir de dados.

Para “ensinar” estes algoritmos a reconhecer uma aeronave intrusa, é necessária a coleta de amostras de dados contendo as imagens de voos da aeronave intrusa capturadas pelo VANT. Nestas imagens as posições da aeronave intrusa foram detectadas manualmente para servir como base do treinamento do algoritmo.

A coleta ocorreu utilizando vídeos feitos a partir de voos aleatórios de ambas as aeronaves (que não gerassem necessariamente uma colisão). A partir das imagens gravadas a aeronave foi rotulada manualmente, e de cada imagem foram tiradas duas amostras, uma amostra que contém a aeronave, que foi classificada como contendo o objeto alvo *(target)*, e outra amostra que contém apenas o fundo da imagem, e que foi classificada como imagem de fundo *(background)*. A Figura 38 ilustra esse processo.

Devido ao fato de a aeronave estar presente somente em um pequeno espaço da imagem, em algumas imagens foram retiradas uma quantidade maior de janelas rotuladas como *background* do que como *target*. Este número desproporcional de imagens pode gerar um aprendizado tendencioso aos algoritmos. Visando resolver esse problema, após a rotulação das amostras, os dados retornados são balanceados de forma que ambas os rótulos tenham a mesma quantidade de janelas rotuladas, retirando-se aleatoriamente janelas do rótulo que se apresente em maiores quantidades.

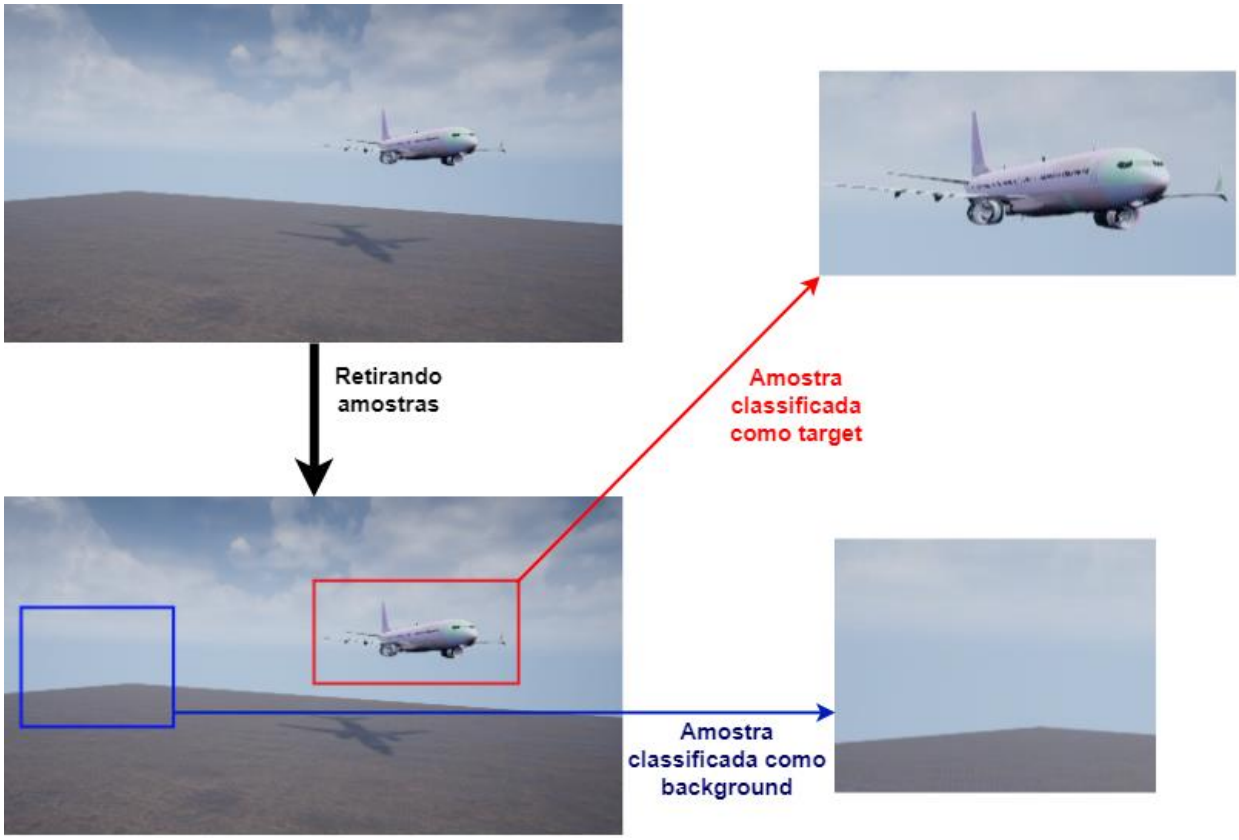

Figura 38: Captura de dados de treino. As imagens mostram uma captura retirada da câmera. Nota-se a retirada de duas amostras, que são classificadas como contendo uma aeronave ou pertencente ao fundo (*background*) [Próprio].

8. Resultados

Para se verificar a funcionalidade do sistema apresentado, foram realizados testes do *framework* proposto neste trabalho, com base nas estratégias apresentadas no capítulo 7. A implementação do *framework* ocorreu na linguagem Python, enquanto que o ambiente de testes e simulações foi desenvolvido no *Unreal* e programado em C++, mesma linguagem utilizada na programação do simulador AirSim. Os testes foram realizados em um *notebook* com processador Intel i5 de *clock* 2.3GHz, com 8GB de memória RAM e sistema operacional Windows 10.

Nos testes foram programados os algoritmos apresentados no capítulo 4, implementando-se todas as diferentes técnicas de visão computacional apresentadas. Para melhor eficiência de execução dos algoritmos, foram utilizadas bibliotecas de amplo uso na área científica, enquanto que para a manipulação de imagens, a principal biblioteca utilizada foi a *numpy*. Para criação de algoritmos de visão computacional utilizou-se a biblioteca *opencv*, enquanto que para algoritmos de aprendizado de máquina as bibliotecas utilizadas foram a *keras* e a *sklearn*.

Para a detecção de objetos em risco de colisão com o VANT controlado foram utilizados três tipos diferentes de sensores: as câmeras visuais, as câmeras de profundidades e o ADS-B. A simulação do sensor ADS-B foi programada a partir de uma implementação própria por meio de sistema de troca de informações entre o *Unreal* (utilizando a linguagem C++) e o *framework* Python. Já para as simulações das câmeras foi utilizado o sistema programado no AirSim. Em parceria com os sensores utilizados para a detecção, também foram utilizados os sensores e os dispositivos de colisão disponíveis no AirSim: câmeras visuais, câmera de profundidade, IMU (Unidades de Medidas Inerciais) e sensor de colisão.

As simulações foram divididas em três fases, sendo que cada uma destas etapas do processo de *sense and avoid* tiveram seus detalhes apresentados no capítulo 4.

8.1. Fase de Detecção

A detecção foi a primeira fase a ser testada. Nela foi implementado um teste de algoritmo/sensor de detecção, considerando que seu foco principal está na eficácia do algoritmo em reconhecer a posição da aeronave intrusa.

Os testes consideraram voos em rota de colisão, entre o VANT controlado e uma aeronave. Nestes voos foram ativados os algoritmos de detecção de cada sensor, e seus resultados foram salvos, para posteriormente serem comparados com o resultado real da posição da aeronave no momento da detecção.

Três sensores foram utilizados: câmera visual, câmera de distância e ADS-B. Para o sensor ADS-B utilizou-se apenas um algoritmo, enquanto que para os dois tipos de câmeras empregou-se um mesmo conjunto de algoritmos, divididos entre detecção utilizando apenas o aprendizado de máquina e detecção mesclando o aprendizado de máquina e algoritmos de rastreamento. Devido ao grande número de algoritmos de visão computacional utilizados, estes foram analisados separadamente.

8.1.1. Algoritmos de visão computacional

Para criação dos algoritmos de detecção por visão computacional, baseados em aprendizado de máquina, é necessário um treinamento. Este treinamento ocorreu com base nas imagens capturadas em voos feitos previamente, no qual a aeronave foi detectada manualmente.

Cinco algoritmos de aprendizado de máquina foram utilizados para os testes neste trabalho. São eles: SVM, *Radom Forest*, *LigthGBM*, Rede Neural (*Perceptron* Multicamada) e *Naive Bayes*. Para avaliar a qualidade do aprendizado de seu treinamento foi utilizada a técnica de validação cruzada (*cross-validation*), utilizando cinco grupos (*folds*). Esta técnica consiste em dividir os dados de treinamento em cinco *folds*, sendo um grupo utilizado para teste, enquanto que os demais foram utilizados para treino. O processo é aplicado cinco vezes, alterando-se o grupo

utilizado como teste e, em cada aplicação do processo é calculada a acurácia[21] (outras métricas podem ser utilizadas, como a precisão e o *recall*) da classificação no conjunto de teste. Ao final é analisada a acurácia média entre as cinco execuções. A Figura 39 exemplifica a utilização desta técnica em um conjunto de dados utilizando 3 *folds*.

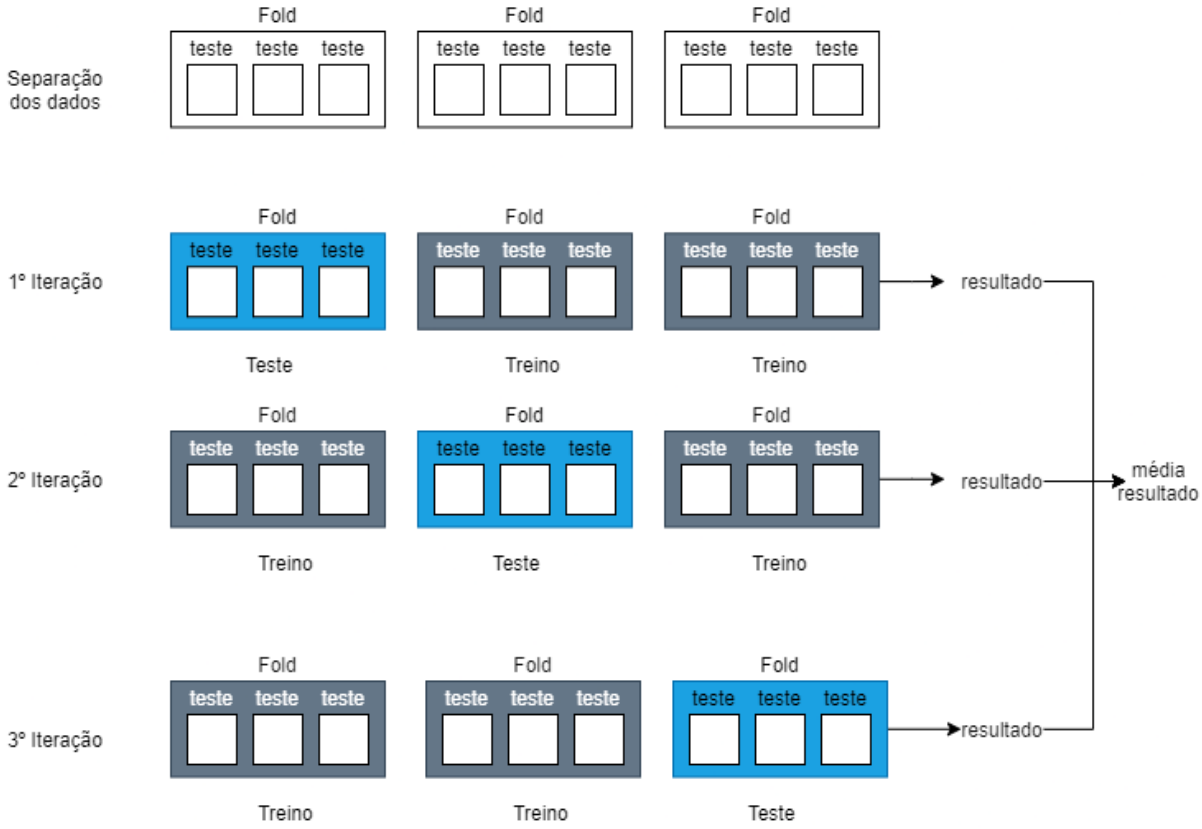

Figura 39: Exemplo de *validação cruzada* com 3 *folds* aplicado em um conjunto de 9 testes.

A Tabela 3 apresenta os valores de acurácia média retornados pelos testes de "*cross*-validade" dos algoritmos de detecção. Essa tabela está separada conforme o tipo de sensor, distinguindo a câmera visual e a câmera de profundidade.

---

[21] A acurácia é calculada pela razão do número de dados classificados corretamente pelo número total de dados, com a seguinte formula: $\frac{vp+vn}{vp+vn+fp+fn}$, em que vp é número de verdadeiros positivos, vn é número de verdadeiros negativos, fp é número de falsos positivos e fn é número de falsos negativos.

Algoritmos	Câmera Visual	Câmera de Profundidade
SVM	90%	98%
Random Forest	90%	96%
LigthGBM	93%	97%
Rede Neural	91%	92%
Xcetion	99%	99%

Tabela 3: Acurácia dos algoritmos de detecção visual no treinamento.

Após a fase de treinamento, os algoritmos foram utilizados para a detecção da posição da aeronave nos cenários de teste descritos no capítulo 7. Neste cálculo foram criadas três combinações diferentes de câmeras: a visão binocular, utilizando duas câmeras visuais como descrito no subcapítulo 4.1.2; a câmera de profundidade, funcionando de maneira isoladamente; e câmera visual em conjunto com a câmera de profundidade. Nos três conjuntos foram testados algoritmos de aprendizado supervisionado e algoritmos de aprendizado *online* (ou rastreamento), sendo que ambos os tipos de algoritmos foram analisados em testes separados.

No primeiro grupo de teste foi analisada a detecção utilizando apenas algoritmos de aprendizado supervisionado, que são: SVM, *Random Forest*, *LigthGBM*, *Naive Bayes* e Redes Neurais. Para a análise dos resultados foram calculados o erro médio (μ) em metros, a porcentagem de verdadeiros positivos (%VP), a porcentagem de falsos positivos (%FP) e a porcentagem de falsos negativos (%FN), segundo descrito no subcapítulo 7.2.1. Seus resultados estão apresentados na Tabela 10.

Algoritmos	Visão Binocular				Misto de Câmeras				Câmera de Profundidade			
DETECÇÃO	**μ(m)**	**%VP**	**%FP**	**%FN**	**μ(m)**	**%VP**	**%FP**	**%FN**	**μ(m)**	**%VP**	**%FP**	**%FN**
SVM	41,6	34%	64%	2%	54,0	100%	0%	0%	42,5	93%	7%	0%
Random Forest	28,8	**47%**	53%	0%	52,0	**100%**	0%	0%	**34.3**	**100%**	0%	0%
LigthGBM	32,6	31%	56%	13%	41,1	86%	6%	8%	49,1	2%	98%	0%
Naive Bayes	**28,2**	20%	26%	54%	**36,3**	56%	3%	41%	**37,4**	87%	13%	0%
Rede Neural	34,4	33%	64%	0%	48,1	95%	5%	0%	43,2	97%	3%	0%

Tabela 4: Média de erro dos algoritmos de detecção entre os algoritmos de aprendizado supervisionado: os valores em negrito representam os melhores valores para um algoritmo segundo a métrica analisada.

No segundo grupo de teste foram analisados os algoritmos de rastreamento, que são: KFC, MIL, TLD e *Boosting*. Porém, este grupo de algoritmos necessita de uma detecção prévia, considerando que eles apenas são capazes de rastrear um objeto inicialmente detectado. Por este motivo eles foram testados em conjunto com os algoritmos de aprendizado supervisionado. Neste modelo de detecção em conjunto, o algoritmo de aprendizado supervisionado foi responsável pela primeira detecção e, em seguida, o algoritmo de rastreamento foi utilizado nos demais *frames*. Caso ocorram cinco *frames* seguidos sem uma detecção do rastreamento, o algoritmo de aprendizado supervisionado é novamente utilizado, reiniciando o processo.

Para testar os algoritmos de rastreamento foram utilizados apenas os algoritmos de detecção que apresentaram melhor precisão na detecção (menor valores de média de erro, representado na tabela por μ) e maior cobertura (maiores valores percentuais de verdadeiro positivos, representado na tabela por %VP), que são o *Random Forest* (RF) e *Naive Bayes* (*Naive*). Para os sensores de visão binocular, sensores do tipo misto de câmeras e sensor usando apenas câmera de profundidade, os melhores resultados foram obtidos por meio dos algoritmos *Random Forest* e *Naive Bayes*.

Algoritmos		Visão Binocular				Misto de Câmeras				Câmera de Profundidade			
RASTREAM.	**DETEC.**	**μ(m)**	**%VP**	**%FP**	**%FN**	**μ(m)**	**%VP**	**%FP**	**%FN**	**μ(m)**	**%VP**	**%FP**	**%FN**
KFC	**RF**	56.8	55%	38%	7%	37,1	22%	78%	0%	36,5	**100%**	0%	0%
MIL	**RF**	55.7	30%	70%	0%	47.3	19%	81%	0%	37,6	89%	11%	0%
TLD	**RF**	59.8	**70%**	30%	0%	46.2	**50%**	50%	0%	34,3	79%	21%	0%
Boosting	**RF**	62.6	20%	80%	0%	33.9	35%	65%	0%	31,7	65%	35%	0%
KFC	***Naive***	42.9	43%	25%	32%	41,2	27%	13%	60%	**25,7**	96%	4%	0%
MIL	***Naive***	**26.0**	17%	43%	40%	**32.2**	12%	53%	35%	35,3	96%	4%	0%
TLD	***Naive***	36.8	45%	14%	41%	46,2	16%	8%	76%	40,7	94%	6%	0%
Boosting	***Naive***	23,7	15%	50%	35%	36.6	18%	45%	37%	44,0	98%	2%	0%

Tabela 5: Resultado dos testes de algoritmos de rastreamento em conjunto com algoritmos de detecção: os valores em negrito apresentam os melhores resultados de cobertura e precisão para cada grupo de sensores.

8.1.2. ADS-B

Após os testes com os algoritmos de detecção visual, simulou-se o dispositivo ADS-B. Os resultados obtidos foram analisados segundo as mesmas métricas utilizadas para as câmeras visuais, calculando-se o erro médio (μ) em metros, a porcentagem de verdadeiros positivos (%VP), a porcentagem de falsos positivos (%FP) e a porcentagem de falsos negativos (%FN), obtendo-se os resultados presentes na Tabela 6.

Algoritmo	**μ (M)**	**%VP**	**%FP**	**%FN**
ADS-B	0,69	99,6%	0%	0,4%

Tabela 6: Resultados de detecção de aeronave utilizando ADS-B.

8.1.3. Análises e comparações entre sensores

Observados os resultados obtidos, é possível realizar-se uma análise comparativa da capacidade de detecção de cada sensor. A maior diferença observada se dá entre o

ADS-B e os demais sensores. O ADS-B tem uma detecção muito precisa e com grande cobertura, e isso se deve ao fato de ele ser um sensor cooperativo. Como neste modelo não é necessária a utilização de algoritmos complexos para a detecção de outra aeronave, mas sim apenas a leitura de dados enviados pela outra aeronave, e o cálculo de suas distâncias, isso torna o processo de detecção mais simples. Por outro lado, este modelo depende do fato de que ambas as aeronaves estejam equipadas com algum modelo do sensor ADS-B, podendo ser ambas com o modelo de ADS-B de entrada e saída, ou apenas uma com um modelo de saída e outra com o modelo de entrada.

Diferentemente do ADS-B, os sensores de visão computacional têm sua detecção baseada exclusivamente nos algoritmos utilizados. Assim, a análise desta detecção é feita comparando-se estes algoritmos. Observando-se os resultados é possível perceber que existe um *tradeoff* entre a precisão e a cobertura obtida. Algoritmos com detecção precisa (com um erro médio baixo), como o *Naive Bayes* aplicado à câmera visual, têm uma baixa cobertura (baixa porcentagem de verdadeiro positivo), enquanto que os algoritmos com alta cobertura, como o SVM, têm uma precisão de detecção mais baixa.

Era esperado que este modelo de detecção combinando algoritmos de detecção e rastreamento permitisse a criação de algoritmos com maior cobertura (maiores taxas de verdadeiro positivos – VP), pois o algoritmo de rastreamento permite uma continuidade de uma detecção correta prévia, com um processamento menor e eficaz quando ocorre apenas pequenas variações do objeto na imagem.

A hipótese se demonstrou real nos testes feitos em câmera de profundidade aplicando a detecção por meio do algoritmo *Naive Bayes*. Porém, utilizando a visão binocular, a hipótese se provou real apenas aplicando o algoritmo de rastreamento TLD, apresentando testes em conjunto com os algoritmos *Random Forest* e *Naive Bayes*, que apresentaram melhorias nos resultados. Por outro lado, utilizando-se a detecção com misto de câmera (câmera visual trabalhando em conjunto com a câmera de

profundidade), a hipótese se demonstrou contrária, apresentando menores valores percentuais de verdadeiro positivo, porém uma maior precisão na detecção.

Por fim, analisando-se os resultados obtidos nas detecções por visão computacional, foram selecionados seis algoritmos: os que apresentaram maior precisão e os que apresentaram maior cobertura para cada sensor. Utilizando a visão binocular, os algoritmos com maior precisão foram o rastreamento usando MIL em conjunto com a detecção utilizando *Naive Bayes*, enquanto que para uma maior cobertura foram o TLD utilizando o *Random Forest*. Para a detecção utilizando o misto de câmeras, os algoritmos com maior precisão para o rastreamento foram os que utilizam o algoritmo MIL aliado à detecção utilizando *Naive Bayes*, e o que apresentou maior cobertura foi o *Random Forest*. Para detecção utilizando apenas câmeras de profundidade, o algoritmo com melhor precisão foi o KFC trabalhando em conjunto com a detecção por meio do *Naive Bayes*; já para uma melhor cobertura, houve um empate entre dois algoritmos: o *Random Forest* e o *KFC* aliado ao *Random Forest*. Para o teste de fusão de sensores foi selecionado o algoritmo *Random Forest*, que apresenta o melhor conjunto de cobertura e precisão entre os dois.

8.2. Fase de Fusão de Sensores

O teste de fusão de sensores foi aplicado em duas diferentes combinações de sensores: a) Visão Binocular + Misto de câmeras; e b) Visão Binocular + Misto de câmeras + ADS-B. Os resultados obtidos para a fusão dos sensores foram por meio do uso da média simples e da média ponderada pela taxa de cobertura do algoritmo (VP – Verdadeiro Positivo). Porém, foram utilizados dois diferentes algoritmos para cada sensor, baseados em câmeras, sendo eles os que apresentaram maior precisão e cobertura para cada sensor. Estes algoritmos estão descritos no capítulo anterior.

Ao final, foram simulados oito cenários de testes, listados a seguir:

1) **2 Câmeras:**
   - Visão binocular: rastreamento utilizando MIL + detecção utilizando *Naive Bayes*;
   - Misto câmeras: rastreamento utilizando MIL + detecção utilizando *Naive Bayes*;
2) **2 Câmeras:**
   - Visão binocular: rastreamento utilizando MIL + detecção utilizando *Naive Bayes*;
   - Misto câmeras: *Random Forest*
3) **2 Câmeras:**
   - Visão binocular: rastreamento utilizando TLD + detecção utilizando Random Forest;
   - Misto câmeras: rastreamento utilizando MIL + detecção utilizando *Naive Bayes*;
4) **2 Câmeras:**
   - Visão binocular: rastreamento utilizando TLD + detecção utilizando Random Forest;
   - Misto câmeras: *Random Forest;*
5) **2 Câmeras:**
   - Visão binocular: rastreamento utilizando MIL + detecção utilizando *Naive*
   - Câmeras de profundidade: *Random Forest;*
6) **2 Câmeras + ADS-B:**
   - Visão binocular: rastreamento utilizando MIL + detecção utilizando *Naive Bayes*;
   - Misto câmeras: rastreamento utilizando MIL + detecção utilizando *Naive Bayes*;
   - ADS-B.

7) **2 Câmeras + ADS-B:**
   - Visão binocular: rastreamento utilizando TLD + detecção utilizando *Random Forest*;
   - Misto câmeras: *Random Forest*;
   - ADS-B.
8) **3 Câmeras + ADS-B:**
   - Visão binocular: rastreamento utilizando TLD + detecção utilizando *Random Forest*;
   - Misto câmeras: *Random Forest*;
   - Câmeras de profundidade: *Random Forest;*
   - ADS-B.

Foram testadas duas técnicas diferentes de fusão: a fusão por média simples e a fusão por média ponderada. A Tabela 13 apresenta as combinações segundo a numeração utilizada anteriormente.

Técnica de fusão	MÉDIA SIMPLES				MÉDIA PONDERADA			
COMBINAÇÃO	μ(m)	%VP	%FP	%FN	μ(m)	%VP	%FP	%FN
1	34,9	100%	0%	0%	36,3	16%	82%	0%
2	23,6	100%	0%	0%	18,5	31%	70%	0%
3	51,3	44%	56%	0%	26,3	43%	57%	0%
4	48,9	80%	20%	0%	42,0	44%	56%	0%
5	30,5	50%	50%	0%	13,1	20%	80%	0%
6	18,3	98%	2%	0%	19,0	82%	18%	0%
7	18,8	100%	0%	0%	16,7	98%	2%	0%
8	16,4	99%	1%	0%	16,7	91%	9%	0%

Tabela 7: Métricas dos algoritmos de fusão de sensores

8.3. Fase de Desvio

Por fim, os melhores algoritmos de detecção foram testados na fase de desvio. Para esta fase, a métrica de análise utilizada foi diferente. Analisou-se a porcentagem de

voos nos quais ocorreriam colisões e, assim, foi analisado o grupo algoritmos com maior capacidade de desviar de cenários iminentes de colisão entre duas aeronaves.

Foram analisados cinco grupos de algoritmos diferentes, dois quais: a) três grupos utilizam apenas um sensor de detecção: câmeras visuais, câmeras de profundidade ou ADS-B; b) um grupo utiliza dois sensores em conjunto: visão binocular e câmeras visuais aliadas às de profundidade; e c) um grupo utiliza todos os sensores em conjunto.

O algoritmo utilizado para a câmera visual foi o *Random Forest*, enquanto que para a câmera de profundidade foi utilizado o KFC em fusão com o SVM.

Algoritmos	**Porcentagem de Desvios Corretos (Horizontal)**	**Porcentagem de Desvios Corretos (Vertical)**
Visual	86%	100%
Profundidade	100%	100%
ADS-B	100%	100%
Visual + Profundidade	100%	100%
ADS-B + Visual + Profundidade	100%	100%

Tabela 8: Análise do algoritmo de desvio. Valor mostrado representa a porcentagem de casos que o algoritmo de desvio funcionou corretamente, ou seja, não ouve colisão.

8.4. Conclusões a respeito dos testes realizados

As simulações demonstraram que foi possível criar algoritmos capazes de evitar colisões em todos os cenários simulados. Os sensores cooperativos demonstraram melhor capacidade de detecção de outra aeronave do que os demais sensores simulados, enquanto que os algoritmos de detecção por visual binocular demonstram um custo-benefício entre precisão e cobertura do algoritmo (porcentagem de verdadeiro positivo). Assim, a escolha de qual algoritmo utilizar pode se dar a partir da métrica a ser priorizada.

Os testes também demonstraram a capacidade de o *framework* trabalhar com diferentes sensores e algoritmos, o que caracteriza a sua modularidade e o torna capaz de ser utilizados em diferentes testes. Com isso, o *framework* pode se tornar uma ferramenta útil para pesquisas futuras, além de ser flexível para o uso em cenários reais.

9. Conclusões

Com a popularização do uso de VANTs vem ocorrendo um crescimento do número de aeronaves no espaço aéreo e, com isso, pode haver um aumento do risco de colisão entre estes veículos com outras aeronaves.

Para controlar esse risco torna-se necessária a criação de técnicas de prevenção de colisão entre aeronaves. Entre estas técnicas destaca-se o *sense and avoid*, um modelo de prevenção de colisão baseado em duas etapas principais, a etapa de detecção, na qual o veículo utiliza-se um sensor ou um conjunto de sensores para detectar uma possível aeronave em risco de colisão, e a etapa de desvio, na qual é aplicada uma técnica de inteligência artificial para planejar uma nova rota para a aeronave controlada visando evitar os riscos de colisão.

Porém, cada etapa do *sense and avoid* pode ser executa de diversas maneiras. A detecção pode variar conforme o sensor utilizado (câmera, LIDAR, radar, comunicação, etc.) ou até mesmo com uma fusão de sensores (câmera visual + câmeras de profundidade, câmera + ADS-B), enquanto que a forma de desvio pode variar conforme o algoritmo utilizado (conjuntos de regras, modelos geométricos, modelos probabilísticos, aprendizado de máquina). Esta enorme variedade de técnicas e tecnologias pode dificultar uma análise e integração entre as elas.

Neste trabalho foi proposto um *framework* genérico e modular para o modelo *sense and avoid*, capaz de se utilizar de diferentes tipos de sensores e técnicas de detecção em conjunto com algoritmos de resolução de conflito. Foi apresentada a arquitetura do *framework*, bem como sua capacidade de personalização.

Para a avaliação do *framework* proposto, ele foi implementado e utilizado junto a um simulador de voos para VANTs (AirSim) e, também, foram implementados diferentes algoritmos de detecção e desvio, a fim de se verificar a capacidade do *framework* trabalhar com essas diferentes técnicas.

9.1. Contribuições

9.1.1. *Framework* para *sense and avoid*

A principal contribuição deste trabalho foi o *framework* de *sense and avoid*. Este *framework* apresentou como características principais a modularidade, a possibilidade de troca de algoritmos, a flexibilidade a diferentes tipos de sensores, a capacidade de utilizar quantidade variada de sensores e ao fato de ser independente da aeronave considerada.

Estas características fazem com que este *framework* seja capaz de trabalhar com as diversas técnicas de *sense and avoid* propostas nos estudos detalhados apresentados no capítulo 2. Com isso, o *framework* possibilita que diferentes técnicas trabalhem em conjunto sobre um mesmo sistema, facilitando a utilização de múltiplos sensores de detecção ou o uso de diferentes algoritmos de desvio, em um mesmo veículo, aumentando a capacidade de redundância, sem necessariamente aumentar a complexidade do sistema. Outra vantagem do *framework* de *sense and avoid* é possibilitar que múltiplos veículos trabalhem com o sistema de *sense and avoid*, independentemente de seus sensores ou capacidades de processamento. Por fim, o *framework* também permite uma análise entre as diferentes técnicas de detecção, facilitando que pesquisas futuras possam avaliar o melhor grupo de sensores e algoritmos para otimizar as técnicas de *sense and avoid.*

9.1.2. Análise dos algoritmos de detecção

Para a avaliação do *framework* de *sense and avoid* foram realizados simulações e testes para as fases de detecção e de desvio entre um VANT e uma outra aeronave, utilizando diferentes algoritmos e sensores. Assim, este trabalho também apresentou uma avaliação de algumas das técnicas atuais de detecção, verificando quais delas são capazes de resolver com melhor precisão os possíveis cenários de colisão entre aeronaves.

Com estes testes é possível descobrir o grau de confiança de cada de cada técnica de detecção, verificando qual pode ser a mais adequada para cada cenário, levando em conta eventuais restrições, como sensores disponíveis, capacidade de processamento ou limitações da aeronave.

9.2. Trabalhos futuros

Algumas atividades não contempladas neste trabalho, em virtude do escopo desta pesquisa, mas poderiam ser úteis para uma melhor análise do *framework* ou das estruturas de *sense and avoid*. Elas são aqui elencadas e podem ser incorporadas em possíveis trabalhos futuros:

- **Implementação da detecção por LIDAR:** O LIDAR é um tipo de sensor que vem ganhando destaque nas pesquisas científicas, envolvendo detecção de objetos em um ambiente 3D. Sua utilização como sensor de detecção foi proposta neste trabalho, porém não foi realizada uma implementação exaustiva de todos os tipos de sensores, pois o objetivo desta pesquisa é propor um *framework* para o *sense and avoid*, e não avaliar todos os tipos e combinações de sensores. Logo, os testes com o uso do LIDAR podem ser realizados em pesquisas futuras;
- **Utilização de diferentes algoritmos de desvio:** O foco principal deste trabalho de pesquisa está na proposição de um *framework* para o *sense and avoid*. Dessa forma, os testes realizados tiveram um escopo restrito, por isso não foram utilizadas técnicas muito avançadas de desvio e de replanejamento de rota. Porém, o *framework* demonstrou-se capaz de permitir a alteração das técnicas de desvio e de replanejamento de rota de maneira simples. Desta forma, um outro possível trabalho futuro pode ter foco nessas técnicas, avaliando-se qual seria a melhor técnica de desvio de rota para ser utilizada dentro deste contexto;
- **Implementação do filtro de *Kalman*:** O filtro de *Kalman* é adequado para aumentar a acurácia de um dos algoritmos de rastreamento ou para a

otimizar a fusão de sensores. Sua utilização nos passos de detecção poderia trazer bons resultados ao processo de *sense and avoid*;

- **Testes com dados de veículos reais:** As análises atuais do *framework* foram realizadas utilizando-se apenas um simulador, o AirSim. Porém, o *framework* foi criado de modo que as trocas do tipo de aeronave ou do ambiente de teste se dessem de maneira simples (apenas acrescentando uma classe no programa). Assim, um bom teste para o *framework* seria seu uso no cenário real, considerando um VANT físico.

9.3. Considerações finais

Este trabalho de pesquisa apresentou um *framework* genérico e modular para a técnica de *sense and avoid* aplicada a VANTs. O *framework* foi concebido pensando-se de maneira que fosse simples a sua adaptação para utilizar diferentes tipos de algoritmos de detecção e de desvio, que poderiam ser utilizados por qualquer aeronave equipada com diferentes sensores. Para validar esta pesquisa foram criados cenários de colisão entre um *drone* quadrimotor e um avião, nos quais foram testadas a capacidade de detecção e de resolução de conflito, considerando-se diferentes sensores e algoritmos.

Com esses testes o *framework* se demonstrou versátil para cobrir diferentes técnicas, além de ser modular, conforme as especificações estabelecidas para ele. O trabalho se torna disponível para que possa ser utilizado em pesquisas futuras de *sense and avoid*, podendo seus detalhes de implementação serem acessados por meio do *link*: https://github.com/viniciusmdalmeida/Mestrado.

REFERÊNCIAS

Aliustaoglu, Cuneyt, et al. “Tool Wear Condition Monitoring Using a Sensor Fusion Model Based on Fuzzy Inference System.” *Mechanical Systems and Signal Processing*, vol. 23, no. 2, 2009, pp. 539–46, doi:10.1016/j.ymssp.2008.02.010.

Babenko, Boris, and Serge Belongie. *Visual Tracking with Online Multiple Instance Learning*. 2009.

Bharati, Sushil Pratap, et al. “Real-Time Obstacle Detection and Tracking for Sense-and-Avoid Mechanism in UAVs.” *IEEE Transactions on Intelligent Vehicles*, vol. 3, no. 2, 2018, pp. 1–1, doi:10.1109/TIV.2018.2804166.

BREIMAN, LEO. “Random Forests.” *Machine Learning*, vol. 45, 2020, pp. 5–32, doi:10.1007/978-3-030-62008-0_35.

Caltagirone, Luca, et al. “Fast LIDAR-Based Road Detection Using Fully Convolutional Neural Networks.” *IEEE Intelligent Vehicles Symposium, Proceedings*, no. Iv, IEEE, 2017, pp. 1019–24, doi:10.1109/IVS.2017.7995848.

Cappello, Francesco, et al. “Low-Cost Sensors Based Multi-Sensor Data Fusion Techniques for RPAS Navigation and Guidance.” *2015 International Conference on Unmanned Aircraft Systems, ICUAS 2015*, 2015, pp. 714–22, doi:10.1109/ICUAS.2015.7152354.

Carrio, Adrian, et al. “Obstacle Detection System for Small UAVs Using ADS-B and Thermal Imaging.” *Journal of Intelligent & Robotic Systems*, no. May 2014, 2017, doi:10.1007/s10846-017-0529-2.

Chollet, François. “Xception: Deep Learning with Depthwise Separable Convolutions.” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 1251–58, doi:10.4271/2014-01-0975.

De Haag, Maarten Uijt, et al. “Flight-Test Evaluation of Small Form-Factor LiDAR and Radar Sensors for SUAS Detect-and-Avoid Applications.” *AIAA/IEEE Digital Avionics Systems Conference - Proceedings*, vol. 2016-Decem, no. July, IEEE, 2016, pp. 1–11, doi:10.1109/DASC.2016.7778108.

DECEA, Departamento de controle do espaço aéreo. *REGRAS DO AR*. 2016.

E Gamma, R Helm, R Johnson, J. Vlissides. *Design Patterns: Abstraction and Reuse of Object-Oriented Design*. Springer, 1994.

Eurocontrol. *ACAS Guide Airborne Collision Avoidance*. no. December, 2017, https://www.eurocontrol.int/sites/default/files/content/documents/nm/safety/ACAS/safety-acas-II-guide.pdf.

FAA - Federal Aviation Administration. *FAA Aerospace Forecast*. 2021.

FAA- Federal Aviation Administration. *FAA Aerospace Forecast 2018-2038*. 2018, p. 101.

FAA- Federal Aviation Administration. *FAA Aerospace Forecast 2019-2039*. 2019, pp. 1–107.

FAA- Federal Aviation Administration. *Introduction to TCAS II - Version 7.1*. 2011, pp. 1–50, doi:10.1017/CBO9781107415324.004.

Aweiss, Arwa, et al. *Unmanned Aircraft System (UAS) Traffic Management (UTM)*. 2018.

FAA- Federal Aviation Administration. “Unmanned Aircraft Systems Traffic Management: UTM.” *Unmanned Aircraft Systems Traffic Management: UTM*, 2021, pp. 1–317, doi:10.1201/9781003124689.

Fasano, Giancarmine, et al. “Morphological Filtering and Target Tracking for Vision-Based UAS Sense and Avoid.” *2014 International Conference on Unmanned*

Aircraft Systems, ICUAS 2014 - Conference Proceedings, 2014, pp. 430–40, doi:10.1109/ICUAS.2014.6842283.

Fasano, Giancarmine, et al.. “Multi-Sensor-Based Fully Autonomous Non-Cooperative Collision Avoidance System for Unmanned Air Vehicles“Multi-Sensor-Based Fully Autonomous Non-Cooperative Collision Avoidance System for Unmanned Air Vehicles.” *Journal of Aerospace Computing, Information, and Communication*, vol. 5, no. 10, 2008, pp. 338–60, doi:10.2514/1.35145.

Fields, Josh, and Austin Sands. *Digital Camera Failure Rates : Introduction : Digital Camera Reliability*. 2010, pp. 1–8.

Forsyth, David, and Jean Ponce. “Computer Vision: A Modern Approach.” *Prentice Hall*, vol. 59, 2011.

Fu, Changhong, et al. “Robust Real-Time Vision-Based Aircraft Tracking from Unmanned Aerial Vehicles.” *Proceedings - IEEE International Conference on Robotics and Automation*, 2014, pp. 5441–46, doi:10.1109/ICRA.2014.6907659.

Furukawa, Tomonari, et al. “Recursive Bayesian Search-and-Tracking Using Coordinated UAVs for Lost Targets.” *Proceedings - IEEE International Conference on Robotics and Automation*, vol. 2006, no. May, 2006, pp. 2521–26, doi:10.1109/ROBOT.2006.1642081.

Guo, Zhenhua, et al. *A Completed Modeling of Local Binary Pattern Operator for Texture Classification*. Vol. 19, no. 6, 2010, pp. 1657–63.

Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge university press, 2003.

Henriques, F., et al. *Exploiting the Circulant Structure of Tracking-by-Detection with Kernels*. 2012, pp. 5–7.

Hong, Seunghoon, et al. “Online Tracking by Learning Discriminative Saliency Map

with Convolutional Neural Network.” *32nd International Conference on Machine Learning, ICML 2015*, vol. 1, 2015, pp. 597–606.

ICAO. *GUIDE ON TECHNICAL AND OPERATIONAL CONSIDERATIONS FOR THE IMPLEMENTATION OF ADS-B IN THE SAM REGION*. no. May, 2013, pp. 1–61.

---. “Unmanned Aircraft Systems.” *Cir 328 AN/190*, vol. 23, no. 2, 2009, doi:10.1016/B978-0-12-374518-7.00016-X.

Jung, Dongwon. *On-Line Path Generation for Unmanned Aerial Vehicles Using B-Spline Path Templates*. Vol. 36, no. 6, 2013, doi:10.2514/1.60780.

Kalal, Zdenek, et al. “Tracking-Learning-Detection.” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 7, 2012, pp. 1409–22, doi:10.1109/TPAMI.2011.239.

Kapoor, Rohan, et al. “Acoustic Sensors for Air and Surface Navigation Applications.” *Sensors (Switzerland)*, vol. 18, no. 2, 2018, pp. 1–25, doi:10.3390/s18020499.

Ke, Guolin, et al. *LightGBM : A Highly Efficient Gradient Boosting Decision Tree*. no. Nips, 2017, pp. 1–9.

Kochenderfer, Mykel J., et al. “Next-Generation Airborne Collision Avoidance System.” *Lincoln Laboratory Journal*, vol. 19, no. 1, 2013, pp. 17–33.

Kochenderfer, Mykel J., and James P. Chryssanthacopoulos. *Robust Airborne Collision Avoidance through Dynamic Programming*. no. ATC-371, 2011, http://www.ll.mit.edu/mission/aviation/publications/publication-files/atc-reports/Kochenderfer_2011_ATC-371_WW-21458.pdf.

Koenig, N., and A. Howard. “Design and Use Paradigms for Gazebo, an Open-Source Multi-Robot Simulator.” *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, pp. 2149–54, doi:10.1109/IROS.2004.1389727.

Koubaa, Anis, et al. “Micro Air Vehicle Link (MAVlink) in a Nutshell: A Survey.” *IEEE Access*, vol. 7, 2019, pp. 87658–80, doi:10.1109/ACCESS.2019.2924410.

Krizhevsky, Alex, et al. “ImageNet Classification with Deep Convolutional Neural Networks.” *Advances in Neural Information Processing Systems*, 2012, pp. 1097–105, doi:10.1201/9781420010749.

Kuchar, J. “Safety Analysis Methodology for Unmanned Aerial Vehicle (UAV) Collision Avoidance Systems.” *6 Th USA / Europe Seminar on Air Traffic Management Research and Development, Baltimore, MD, June*, vol. 27, 2005, p. 30.

Lacher, Andrew R., et al. “Unmanned Aircraft Collision Avoidance–Technology Assessment and Evaluation Methods.” *7th Air Traffic Managemtent Research & Development Seminar*, 2007, pp. 1–10.

Matsumoto, T. T., et al. “Training and Evaluation of a Learning-Based Autonomous Unmanned Aircraft for Collision Avoidance : Virtual Training Data Generation.” *Safety and Reliability of Complex Engineered Systems*, 2015, pp. 2771–79.

MATSUMOTO, THIAGO TOSHIO. “Modelagem e Simulação de Veículo Aéreo Não Tripulado Autônomo Para Avaliação de Risco de Colisão Com Outras Aeronaves.” *Dissertação (Dissertação Em Engenharia de Computacão) USP*, vol. 53, no. 9, 2013, pp. 1689–99, doi:10.1017/CBO9781107415324.004.

McCallie, Donald, et al. “Security Analysis of the ADS-B Implementation in the next Generation Air Transportation System.” *International Journal of Critical Infrastructure Protection*, vol. 4, no. 2, Elsevier B.V., 2011, pp. 78–87, doi:10.1016/j.ijcip.2011.06.001.

Mueller, Matthias, et al. *AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles*. no. September, 2017, pp. 1–14, doi:10.1007/978-3-319-67361-5_40.

---. *UE4Sim: A Photo-Realistic Simulator for Computer Vision Applications*. 2017,

http://arxiv.org/abs/1708.05869.

Nikolos, Ioannis K., et al. *Evolutionary Algorithm Based Offline / Online Path Planner for UAV Navigation*. Vol. 33, no. 6, 2003, pp. 898–912.

Pestana, Jesús, et al. “Vision Based GPS-Denied Object Tracking and Following for Unmanned Aerial Vehicles.” *2013 IEEE International Symposium on Safety, Security, and Rescue Robotics, SSRR 2013*, 2013, doi:10.1109/SSRR.2013.6719359.

Ramalingam, Karthik, et al. “Integration of Unmanned Aircraft System (UAS) in Non-Segregated Airspace: A Complex System of Systems Problem.” *2011 IEEE International Systems Conference, SysCon 2011 - Proceedings*, 2011, pp. 448–55, doi:10.1109/SYSCON.2011.5929108.

Ramasamy, Subramanian, et al. “Avionics Sensor Fusion for Small Size Unmanned Aircraft Sense-and-Avoid.” *2014 IEEE International Workshop on Metrology for Aerospace, MetroAeroSpace 2014 - Proceedings*, no. November, 2014, pp. 271–76, doi:10.1109/MetroAeroSpace.2014.6865933.

Ramasamy, Subramanian, et al. “LIDAR Obstacle Warning and Avoidance System for Unmanned Aerial Vehicle Sense-and-Avoid.” *Aerospace Science and Technology*, vol. 55, Elsevier Masson SAS, 2016, pp. 344–58, doi:10.1016/j.ast.2016.05.020.

Ramasamy, Subramanian, and Roberto Sabatini. *A Unified Approach to Cooperative And*. 2015.

Redmon, Joseph, and Ali Farhadi. *YOLOv3: An Incremental Improvement*. 2018, http://arxiv.org/abs/1804.02767.

Rosenblatt, F. “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain.” *Psychological Review*, vol. 65, no. 6, 1958, pp. 386–408, doi:10.1037/h0042519.

Ross, Stephane, et al. “Learning Monocular Reactive UAV Control in Cluttered Natural Environments.” *Proceedings - IEEE International Conference on Robotics and Automation*, 2013, pp. 1765–72, doi:10.1109/ICRA.2013.6630809.

Russell, Stuart, and Peter Norvig. *Artificial Intelligence A Modern Approach*. Pearson Education, 2010.

Sapkota, Krishna Raj, et al. *Vision-Based Unmanned Aerial Vehicle Detection and Tracking for Sense and Avoid Systems*. 2016, pp. 1556–61, doi:10.1109/IROS.2016.7759252.

Savva, Manolis, et al. *MINOS: Multimodal Indoor Simulator for Navigation in Complex Environments*. 2017, pp. 1–14, http://arxiv.org/abs/1712.03931.

Sendobry, Alexander, et al. *Comprehensive Simulation of Quadrotor UAVs Using ROS and Gazebo*. Vol. 7628, no. November 2012, 2012, doi:10.1007/978-3-642-34327-8.

Shah, Shital, et al. *AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles*. 2018, pp. 621–35, doi:10.1007/978-3-319-67361-5_40.

Shakhatreh, Hazim, et al. *Unmanned Aerial Vehicles : A Survey on Civil Applications and Key Research Challenges*. 2018, pp. 1–58.

Shi, Shaoshuai, et al. “PV-RCNN : Point-Voxel Feature Set Abstraction for 3D Object Detection.” *ArXiv Preprint ArXiv:1912.13192*, 2019.

Stauffer, Chris, and W. E. L. Grimson. *Adaptive Background Mixture Models for Real-Time Tracking*. Vol. 00, no. c, 1999.

Sun, Junzi. *The 1090MHz Riddle*. 2017.

Teichman, Alex, et al. “Towards 3D Object Recognition via Classification of Arbitrary Object Tracks.” *IEEE International Conference on Robotics and Automation*,

2011, pp. 4034–41.

Tekalp, A. Murat. “Digital Video Processing.” *Prentice Hall Press*, no. May, 2015.

Verma, Vivek, et al. “3D Building Detection and Modeling from Aerial LIDAR Data.” *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, 2006, pp. 2213–20, doi:10.1109/CVPR.2006.12.

Vismari, Lúcio Flávio. “Vigilância Dependente Automática No Controle De Tráfego Aéreo : Avaliação De Risco Baseada Em Modelagem Em Redes De Petri Controle De Tráfego Aéreo : Avaliação De Risco.” *Dissertação de Mestrado*, 2007, p. 289.

Wang, Dominic Zeng, and Ingmar Posner. “Voting for Voting in Online Point Cloud Object Detection.” *Robotics: Science and Systems*, vol. 11, 2015, doi:10.15607/RSS.2015.XI.035.

Watts, Adam C., et al. “Unmanned Aircraft Systems in Remote Sensing and Scientific Research: Classification and Considerations of Use.” *Remote Sensing*, vol. 4, no. 6, 2012, pp. 1671–92, doi:10.3390/rs4061671.

Wu, Yuanwei, et al. *Vision-Based Real-Time Aerial Object Localization and Tracking for UAV Sensing System*. 2017, pp. 1–9, http://arxiv.org/abs/1703.06527.

Xiao, Liang, et al. “CRF Based Road Detection with Multi-Sensor Fusion.” *IEEE Intelligent Vehicles Symposium, Proceedings*, vol. 2015-Augus, no. Iv, IEEE, 2015, pp. 192–98, doi:10.1109/IVS.2015.7225685.

Yang, Hanxuan, et al. “Recent Advances and Trends in Visual Tracking : A Review.” *Neurocomputing*, vol. 74, no. 18, Elsevier, 2011, pp. 3823–31, doi:10.1016/j.neucom.2011.07.024.

Yu, Xiang, and Youmin Zhang. “Sense and Avoid Technologies with Applications to Unmanned Aircraft Systems: Review and Prospects.” *Progress in Aerospace Sciences*, vol. 74, Elsevier, 2015, pp. 152–66,

doi:10.1016/j.paerosci.2015.01.001.

ANEXO A

Para realização deste trabalho foi construído um *framework* em linguagem Python, seguindo a mesma base teórica e arquitetura descritas no sistema apresentado nesta dissertação.

Sua estrutura foi criada de maneira a ser o mais modular possível, permitindo que novos programadores ou pesquisadores possam utilizá-lo para seus futuros trabalhos, poupando, assim, o tempo necessário para construção um sistema similar.

Seu código está disponível por meio do *github*, pelo *link* a seguir:

https://github.com/viniciusmdalmeida/msaco

- Utilização do *framework*

Para utilizar o *framework* é necessário ter o AirSim instalado como componente de um projeto do *Unreal*. Para esta instalação é possível seguir o tutorial descrito na documentação da biblioteca pelo *link* https://microsoft.github.io/AirSim/build_windows/ Porém, também é possível personalizar o sistema para ser utilizado em outro simulador ou em uma aeronave real.

- Movimentando a aeronave

O *framework* pode ser utilizado simplesmente para movimentar a aeronave. Para isso é necessário apenas iniciar o sistema principal com alguns pontos de rota, algo que pode ser executado por meio do código em Python seguir.

```
from msaco.Interface.Start import Start

#Criando lista de pontos da rota
ponto_1 = [10,5,3]
ponto_2 = [10,12,13]
lista_pontos_rota = [ponto_1, ponto_2]

#Iniciando simulador
run_simulation = Start(lista_pontos_rota)
```

```
#Executando simulador
run_simulation.start()
run_simulation.join()
```

- Utilizando o processo de *sense and avoid*

Para executar um processo de *sense and avoid* é necessário selecionar as classes dos algoritmos, que podem ser algoritmos de detecção de objetos (aeronave), fusão de sensores ou desvio. O *framework* tem seu código implementando todos os algoritmos testados no capítulo 8 deste trabalho.

O código abaixo monstra uma simulação utilizando o algoritmo *Random Forest* para detecção utilizando visão estéreo (binocular), um algoritmo de fusão de sensores aplicando a média simples e um algoritmo de desvio utilizado o desvio horizontal.

```
from msaco.Interface.Start import Start

#Importando algoritmo de detecção binocular
from msaco.AlgorithmsSensors.cam.StereoCam import VisionDetectRF

#Importando algoritmo de fusão de dados
from msaco.Detect.Fusion import FusionData_Mean

#Importando algoritmo de desvio
from msaco.Avoid.Avoid import HorizontalAvoid


#Criando lista de pontos da rota
ponto_1 = [10,5,3]
ponto_2 = [10,12,13]
lista_pontos_rota = [ponto_1, ponto_2]

#Iniciando simulador
run_simulation = Start(lista_pontos_rota,
                           sensorAlgorithms = VisionDetectRF,
                           fusionAlgorithm = FusionData_Mean,
                           avoidClass = HorizontalAvoid,
                           )

#Executando simulador
run_simulation.start()
run_simulation.join()
```

- Personalização de classes

O *framework* permite personalização das classes do algoritmo de maneira simples. Assim, novos pesquisadores podem utilizar deste *framework* para pesquisas focadas apenas em uma das etapas do *sense and avoid*. Para isso, é necessário apenas criar uma classe que tenha como base a classe principal da etapa, pois a classe criada herda a interface do algoritmo a ser personalizada.

Para a personalização do desvio, por exemplo, é necessário que a classe criada tenha como herança a classe `BaseAvoid` que está no caminho `msaco.Avoid.Avoid`. O código a seguir monstra a criação de uma classe quer permite o desvio, sempre seguindo 10 metros à esquerda.

```
from msaco.Avoid.Avoid import BaseAvoid

class HorizontalAvoid(BaseAvoid):
    def avoid_strategy(self,detection_data):
        my_position = detection_data.getDictData()['myPosition']
        new_position = my_position
          new_position[1] += 10
          velocity = 5

        return [list(new_position)], velocity
```

Outro exemplo pode ser a personalização do algoritmo de detecção. Para isso é necessário que o algoritmo criado herde a classe `AlgorithSensor` que está no caminho `msaco.AlgorithmsSensors`. O código a seguir monstra a criação de um algoritmo personalizado de detecção que sempre retorna o valor 100 de distância na posição x=10m, y=5m e z=8m.

```
from msaco.AlgorithmsSensors import AlgorithSensor

class ExDetect(AlgorithSensor):
    def detect(self,detection_data):
        distance = 100
        intruse_position = [10,5,8]
        self.detectData.updateData(distance=distance,otherPosition=intruse_positio
        n)
```