UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Simulação de Missões de VANTs em Ambientes 3D Fictícios e Gêmeos Digitais com Georreferenciamento Direto de Pixels

Lucas dos Santos Conde

DM: 25/2023

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2023

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Lucas dos Santos Conde

Simulação de Missões de VANTs em Ambientes 3D Fictícios e Gêmeos Digitais com Georreferenciamento Direto de Pixels

DM: 25/2023

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2023

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Lucas dos Santos Conde

Simulação de Missões de VANTs em Ambientes 3D Fictícios e Gêmeos Digitais com Georreferenciamento Direto de Pixels

Dissertação de mestrado submetida à avaliação da banca examinadora aprovada pelo colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal do Pará e julgada adequada para obtenção do Grau de Mestre em Engenharia Elétrica na área de Telecomunicações.

UFPA / ITEC / PPGEE
Campus Universitário do Guamá
Belém-Pará-Brasil
2023

Dados Internacionais de Catalogação na Publicação (CIP) de acordo com ISBD
Sistema de Bibliotecas da Universidade Federal do Pará
Gerada automaticamente pelo módulo Ficat, mediante os dados fornecidos pelo(a) autor(a)

C745s Conde, Lucas dos Santos.
Simulação de Missões de VANTs em Ambientes 3D Fictícios e Gêmeos Digitais com Georreferenciamento Direto de Pixels / Lucas dos Santos Conde. — 2023.
100 f. : il. color.

Orientador(a): Prof. Dr. Aldebaro Barreto da Rocha Klautau Júnior
Dissertação (Mestrado) - Universidade Federal do Pará, Instituto de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica, Belém, 2023.

1. VANT. 2. Georreferenciamento. 3. Inteligência Artificial. 4. Unreal Engine. 5. AirSim. I. Título.

CDD 621.3

UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

"SIMULAÇÃO DE MISSÕES DE VANTS EM AMBIENTES 3D FICTÍCIOS E GÊMEOS DIGITAIS COM GEORREFERENCIAMENTO DIRETO DE PIXELS"

AUTOR: **LUCAS DOS SANTOS CONDE**

DISSERTAÇÃO DE MESTRADO SUBMETIDA À BANCA EXAMINADORA APROVADA PELO COLEGIADO DO PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA, SENDO JULGADA ADEQUADA PARA A OBTENÇÃO DO GRAU DE MESTRE EM ENGENHARIA ELÉTRICA NA ÁREA DE TELECOMUNICAÇÕES.

APROVADA EM: 14/08/2023

BANCA EXAMINADORA:

Prof. Dr. Aldebaro Barreto da Rocha Klautau Júnior
(Orientador – PPGEE/UFPA)

Prof. Dr. Leonardo Lira Ramalho
(Avaliador Interno – PPGEE/UFPA)

Prof. Dr. Ilan Sousa Correa
(Avaliador Externo ao Programa – PESQUISADOR/UFPA)

VISTO:

Prof. Dr. Diego Lisboa Cardoso
(Coordenador do PPGEE/ITEC/UFPA)

Agradecimentos

Agradeço em primeiro lugar a Deus, que além de me criar, me mantém diariamente com Sua graça, me advertindo e dando sabedoria para que possa continuar e não desistir em ser uma versão melhor de mim a cada dia.

A minha esposa (a qual no início dessa jornada ainda era minha namorada) que em todos os momentos me apoiou e me incentivou a continuar, sendo meu suporte e cuidando de mim para que eu pudesse alcançar o objetivo do desenvolvimento deste trabalho. A meus familiares e amigos, especialmente meus pais, Sandra e Manoel Conde, e meus irmãos, Matheus e Vitória Conde, que também foram alicerces para que eu pudesse almejar um alvo mais elevado e ter condições para atingi-lo.

Ao Núcleo de Pesquisa e Desenvolvimento em Telecomunicações, Automação e Eletrônica (LASSE), que é uma família, e como tal demonstrou todo apoio, não só técnico, mas também compreensão, paciência e companheirismo para que eu pudesse avançar na pesquisa acadêmica, mesmo diante dos obstáculos. Em especial, ao meu orientador, Aldebaro Klautau, que durante esses anos se mostrou como um exímio professor, não só na área técnica da engenharia, mas também um conselheiro e treinador para o jogo da vida. E também a equipe dos projetos AGRO e outros relacionados a aplicações em ambientes simulados, especialmente ao Rafael, Arthur, Amanda, Emerson, Felipe e João, os quais contribuíram com as suas experiências para que a construção desta dissertação fosse melhor fundamentada.

Lucas Conde

Tudo posso naquele que me fortalece.

(Bíblia Sagrada, Filipenses 4:13)

Lista de Acrônimos

API *Application Programming Interface*

CARLA *Car Learning to Act*

CNN *Convolutional Neural Network*

DCM *Directional Cosine Matriz*

DD *Decimal Degrees*

DLSS *Deep Learning Super Sampling*

DM *Decimal Minutes*

DMS *Degrees Minutes and Seconds*

ECEF *Earth-Centered, Earth-Fixed*

ENU *East-North-UP*

EPSG *European Petroleum Survey Group*

EXIF *Exchangeable Image File Format*

FOV *Field of View*

GPS *Global Positioning System*

GSD *Ground Sampling Distance*

HITL *Hardware-In-The-Loop*

HTML HyperText Markup Language

IA Inteligência Artificial

IoT *Internet of Things*

IRM *IERS Reference Meridian*

LASSE Núcleo de Pesquisa e Desenvolvimento em Telecomunicações, Automação e Eletrônica

NED *North-East-Down*

ODM *OpenDroneMap*

PCTs Pontos de Controle Terrestre

PTL Plano Tangente Local

RPG *Role Playing Game*

RTX *Ray Tracing Extreme*

SGR Sistemas Geodésicos de Referência

SITL *Software-In-The-Loop*

SRC Sistema de Referência de Coordenadas

SRID *Spatial Reference Identifier*

UAS *Unmanned Aircraft System*

UE4 *Unreal Engine 4*

UTM Universal Transversa de Mercator

VANT Veículo Aéreo Não Tripulado

WGS84 *World Geodetic System*

YOLO *You Look Once*

Lista de Figuras

Lista de Tabelas

Sumário

Resumo

O mundo está rapidamente adentrando em uma realidade onde a Inteligência Artificial (IA) se faz cada vez mais presente em diversos sistemas, abrangendo desde o ambiente doméstico até setores como indústria, mobilidade urbana e agronegócio, entre outros. Nesse contexto, com o avanço do poder computacional, simuladores para o desenvolvimento e teste de sistemas autônomos têm despertado grande interesse, tanto por parte de grandes empresas quanto da comunidade científica, devido à fidelidade visual e física que oferecem. Esses simuladores são frequentemente considerados como "gêmeos digitais" de cenários e sistemas reais, e têm trazido vantagens significativas em termos de custo e tempo, poupando recursos físicos e humanos durante o processo de concepção e aprimoramento de algoritmos. Entre esses sistemas, encontram-se os Veículos Aéreos Não Tripulados (VANTs), que têm demonstrado ser de grande utilidade em contextos como a mobilidade e monitoramento urbano e rural. Eles são aplicados, por exemplo, na detecção de defeitos em painéis fotovoltaicos, identificação de ervas daninhas em plantações, extensão da rede móvel e na busca e resgate de pessoas. Diante disso, este trabalho apresenta a concepção de uma metodologia que integra a simulação realista de missões com VANTs, utilizando o simulador AirSim em conjunto com o motor gráfico Unreal Engine, juntamente com recursos de visão computacional. O objetivo é realizar a detecção de objetos (empregando o modelo de IA YOLO) associada à localização georreferenciada dos mesmos, além de gerar arquivos de imagens geolocalizados, os quais são compatíveis com softwares comerciais para processamento de imagens aéreas. Os resultados foram avaliados usando o software WebODM para reconstrução do cenário a partir dos arquivos de imagens geolocalizados (gerando ortofotos). E para avaliação do algoritmo de georreferenciamento direto de pixels, foi testada a capacidade do drone voltar a posição da pessoa (ou objeto) detectado após a posição GPS ser retornada pelo algoritmo, com erros menores que 5 metros em relação à posição (em coordenadas UTM) real do elemento no ambiente 3D.

Palavras-chave — VANT, Georreferenciamento, Inteligência Artificial, Unreal Engine, AirSim

Abstract

The world is rapidly entering a reality where Artificial Intelligence is becoming increasingly present in various systems, from the domestic environment to sectors such as industry, urban mobility, and agriculture. In this context, with the advancement of computational power, simulators for developing and testing autonomous systems have garnered significant interest from large companies and the scientific community due to the visual and physical fidelity they offer. These simulators are often regarded as "digital twins" of real scenarios and systems and have brought significant advantages in terms of cost and time, saving physical and human resources during the conception and improvement of algorithms. Among these systems are Unmanned Aerial Vehicles (UAVs), which have proven to be of great utility in contexts such as urban and rural mobility and monitoring. They are applied, for example, in detecting defects in photovoltaic panels, identifying weeds in crops, extending the mobile network and in search and rescue missions. Therefore, this work presents the conception of a methodology that integrates realistic mission simulation with UAVs, using the AirSim simulator in conjunction with the Unreal Engine graphics engine and computer vision capabilities. The objective is to perform object detection (employing the YOLO AI model) associated with their georeferenced location and generate geolocated image files that are compatible with commercial software for aerial image processing. The results were evaluated using the WebODM software for scene reconstruction from geolocated image files (generating orthophotos). And to evaluate the direct pixel georeferencing algorithm, the ability of the drone to return to the position of the detected person (or object) after the algorithm provided the GPS position was tested, with errors smaller than 5 meters in relation to the real position (in UTM coordinates) of the element in the 3D environment.

***Keywords* —** UAV, Georeferencing, Artificial Intelligence, Unreal Engine, AirSim

Capítulo 1

Introdução

Com o avanço do poder computacional, como o desenvolvimento de Unidades de Processamento Gráfico - Graphics Processing Units (GPUs) - capazes de renderizar imagens de maneira cada vez mais fidedigna, incluindo efeitos naturais como o reflexo da luz em superfícies refletoras (por meio da tecnologia Ray Tracing), a indústria de jogos e, por consequência, a criação de mundos virtuais para fins além do entretenimento, têm ganhado força no mercado [1]. Além disso, exemplos como o uso de IA capaz de gerar quadros em alta resolução em tempo real para que jogos possam ser executados com uma boa taxa de quadros por segundo (como a tecnologia *Deep Learning Super Sampling* (DLSS) da NVIDIA), mostram como o mundo tem caminhado para uma imersão em mundos virtuais realísticos, com cada vez mais dificuldade de distinguir dados reais dos que são sintetizados por motores gráficos associados a técnicas de aprendizado de máquina [2].

Neste sentido, plataformas como NVIDIA Omniverse, Unreal Engine e Unity têm avançado no provimento de funcionalidades capazes de gerar dados sintéticos próximos da realidade por meio da implementação de gêmeos digitais (*digital twins*). Esse conceito se refere a criar réplicas digitais exatas de objetos, processos e sistemas do mundo real (como até cidades inteiras) combinando tecnologias como IA, *Internet of Things* (IoT), Metaverso, Realidade Virtual e Aumentada [3]. Assim, esses modelos digitais podem ser usados em linhas de produção industriais, como por exemplo a empresa *Ford*, que usa gêmeos digitais para prever com precisão áreas onde a energia pode ser conservada e melhorar o desempenho geral das linhas de produção [4]. A empresa BMW também já tem usado gêmeos digitais produzidos com o NVIDIA Omniverse para otimizar layouts, robótica e sistemas de logística anos antes do início da produção [5]. Já a empresa *CAD Center* tem usado o motor gráfico Unreal Engine para construir

cidades 3D para gêmeos digitais e planejamento urbano, como o exemplo da cidade de Tokyo, onde é possível treinar técnicas de fotografia aérea e visualização dos ambientes urbanos [6].

Uma das utilidades de se ter ambientes virtuais com alto grau de fidelidade é a viabilização de dados para desenvolvimento de algoritmos de IA relacionados a áreas como visão computacional e robótica, incluindo veículos autônomos. Isso porque técnicas de aprendizado de máquina como, aprendizado por reforço (*reinforcement learning*), aprendizado por demonstração (*learning-by-demonstration*) e aprendizado por transferência (*transfer learning*) têm como um dos desafios chave para o sucesso na implementação, a necessidade de uma grande quantidade de dados para a fase de treinamento. Entretanto, obter esses dados e treinar os sistemas não é uma tarefa trivial visto os custos e riscos advindos de operar robôs e veículos autônomos reais ainda na fase de treinamento. Isso pode trazer custos significativos (tanto de tempo quanto a estrutura física) para uma planta industrial, por exemplo, onde se deseja otimizar a catalogação de produtos e movimentação de braços mecatrônicos [7]. Ou para testar a automação de serviços de entrega utilizando VANTs.

Percebe-se então que, para reduzir esses custos é necessário treinar esses modelos de sistemas autônomos, muitas vezes associados a visão computacional para a tomada de decisão, em ambientes com modelos complexos de objetos do mundo real, como árvores, estradas, lagos, postes elétricos, casas e pessoas, juntamente com renderizações que incluam detalhes refinados, como sombras suaves, reflexões especulares, inter-reflexões difusas e assim por diante. Além disso, é necessário que os modelos desses sistemas (robôs) que serão simulados e inseridos nos ambientes virtuais sejam precisos quanto a dinâmica física real, incluindo a simulação dos sensores (ou dos dados advindos dos mesmos) comumente associados a esses sistemas automatizados. Por isso, plataformas de simulação de veículos autônomos, como AirSim e CARLA, têm sido fortemente usadas para diminuir a distância entre a simulação e a realidade e assim proporcionar agilidade no desenvolvimento de algoritmos de visão computacional, aprendizado de máquina e automação de VANTs e carros [8] [9].

Outra iniciativa que pode ser encontrada a qual mostra o interesse no desenvolvimento de veículos autônomos, em especial VANTs, é o Desafio Petrobras, que está inserido na Competição Brasileira de Robótica. Este desafio visa estimular o desenvolvimento de robôs voadores autônomos e inteligentes na inspeção e operação em faixas de dutos e instalações, o que traz vários desafios quanto ao desenvolvimento de controladores de voos robustos (incluindo o controle de trajetória, altitude e atitude) com rastreamento e localização precisa e independentes

por câmera [10]. Durante os anos da pandemia de COVID-19, esse desafio teve suas edições realizadas com auxílio do simulador Gazebo [11], o qual é muito utilizado no treino de aplicações voltadas ao contexto de desenvolvimento de robôs.

Ainda no contexto do uso de VANTs, esses têm-se tornado ferramentas em alta para aplicações onde é necessário realizar detecção de objetos, como em missões de busca e resgate. O uso de vigilância pelos mais popularmente conhecidos drones, com tecnologias atuais de visão computacional pode aumentar o número de humanos salvos no momento de um desastre. Apesar de opiniões divididas quanto ao uso, os drones estão sendo mais usados, fornecendo excelente suporte em operações de risco, como nos departamentos de polícia e bombeiros para situação de resgastes urgentes [12]. Em [13], missões de busca e resgate de pessoas são conduzidas em desertos da Escócia pela policia local, usando um sistema que compreende em um VANT acoplado a um sistema de detecção de objetos baseado em aprendizado de máquina em tempo real, usando o algoritmo *Tiny-YOLOv3*, incorporado a um smartphone. Nessa aplicação os autores revelam alcançar uma acurácia de 94,73% na tarefa de detecção, trazendo contribuições significativas para maximizar as chances de salvar vidas.

Devido a necessidade de cobrir áreas de busca grandes e complexas, esforços estão sendo feitos para implementar sistemas com múltiplos VANTs cooperando simultaneamente, visando encurtar o tempo e o consumo de energia (por drone) nas missões de busca e resgate. Assim, é possível melhorar a taxa de sucesso com o uso de tecnologias de detecção de alvos, com algoritmos como a YOLOv5 - onde a série *You Look Once* (YOLO) consiste de um método de regressão de um estágio baseado em aprendizado profundo, que prioriza a velocidade em tarefas de detecção de objetos [14].

Em tarefas de detecção de pessoas em situação de risco, é muitas vezes necessário retornar a localização exata da pessoa para a Estação de Controle Terrestre, a fim de que assim, uma equipe de profissionais possa ir ao resgate do(s) indivíduo(s). Visto isso, tarefas como georreferenciamento de imagens, as quais tradicionalmente necessitam de Pontos de Controle Terrestre (PCTs), podem se tornar inviáveis pois na maioria das vezes o operador não pode colocar esses pontos antes da missão. PCTs são pontos de marcação posicionados no terreno que será mapeado, ou monitorado, que possuem uma localização geográfica conhecida. O processo de georreferenciamento de imagens aéreas (obtidas por VANTs, nesse contexto) sem o uso dos PCTs é conhecido como georreferenciamento direto. Este processo se torna mais interessante pois depende do uso de informações conhecidas do veículo aéreo e a câmera embarcada no

mesmo (como posição, atitude, sensor de imagem e características da lente) para calcular a posição do objeto de interesse na imagem aérea no sistema de coordenadas geográfico [15] [16]. Além da tarefa de detecção de pessoas em missões de busca e resgate, o algoritmo de georreferenciamento direto pode ser aplicado a nível de pixels a fim de retornar com mais exatidão a localização de defeitos em módulos fotovoltaicos [17] [18], de isoladores na rede elétrica [19], ervas daninhas em uma plantação [20] ou até mesmo de outros VANTs [21].

Dessa forma, a contribuição desse trabalho é a concepção e implementação de uma metodologia para simulações com VANTs e visão computacional, que permita gerar arquivos de imagens com os objetos detectados, associados a informações georreferenciadas. Esses arquivos estão em formato compatível com o usado por câmeras de drones (JPEG, PNG, TIFF), de forma que as imagens geradas pelo sistema desenvolvido sejam compatíveis com softwares comerciais. Com isso, é possível treinar sistemas *Unmanned Aircraft System* (UAS) (que consistem na junção do drone, com a estação de controle terrestre e o link de comunicação) por meio de simulações *Software-In-The-Loop* (SITL), gerando dados de vôo para o desenvolvimento de algoritmos voltados a visão computacional de forma menos custosa (já que o sistema é simulado). A solução apresentada foi implementada com o motor gráfico *Unreal Engine 4* (UE4), associada ao simulador de VANTs, AirSim, o qual proporciona uma série de recursos para trazer realismo visual e físico a simulação, incluindo uma *Application Programming Interface* (API) para auxiliar na movimentação e obtenção de dados do drone simulado. Associado a essas ferramentas, foi integrado o algoritmo da YOLOv5 para detecção de pessoas [22].

Com relação ao trabalhos pesquisados e usados como base para o desenvolvimento de uma metodologia para simulação de missões de Veículos Aéreos Não Tripulados (VANT) em ambientes 3D com integração de georreferenciamento direto de pixels, a Tabela 1.1 mostra uma comparação com 2 artigos que mais se aproximam do que foi desenvolvido nessa dissertação. A comparação se dá com relação as ferramentas utilizadas, qual o(s) objetivo(s) principais e o diferencial do trabalho, e os resultados apresentados.

1.1 Organização do Trabalho

Os capítulos que se seguem estão divididos da seguinte forma:

- O Capítulo 2 aborda alguns trabalhos relacionados que foram usados como base para desenvolvimento ou comparação enquanto se estruturava a simulação proposta, incluindo

um detalhamento dos fundamentos necessários para o entendimento do que foi implementado.

- O Capítulo 3 indica os passos tomados assim como explica o fluxo da simulação implementada.
- O Capítulo 4 mostra os resultados obtidos a partir da integração de todas as ferramentas utilizadas para compor a metodologia proposta.
- O Capítulo 5 apresenta as considerações finais a partir dos resultados obtidos, assim como propostas de como o sistema pode ser melhorado tanto em funcionalidades como em aplicações.

Esta dissertação aborda um tópico distinto da publicação [23] pois a pandemia impediu que a pesquisa inicial continuasse. Assim, o tema da dissertação foi modificado.

Tabela 1.1: Comparação com trabalhos relacionados [17] [15]

Título do trabalho	**Ferramentas utilizadas**	**Qual o foco e o diferencial do trabalho?**	**Resultados**
Virtual Reality Simulation of Autonomous Solar Plants Inspections with Unmanned Aerial Systems	Unreal Engine 4; AirSim; Blender; Ambiente 3D fictício; Simulações SITL; Controlador de vôo ArduPilot; Georreferenciamento direto de pixels; Biblioteca Python Piexif para geolocalização das imagens	Foco no desenvolvimento de uma ferramenta para gerar dados sintéticos realistas no contexto de um ambiente de simulação para inspeção de usinas solares. Diferencial está na integração da simulação SITL na Unreal Engine 4 com o algoritmo de georreferenciamento.	- Ambiente 3D para inspeção de usinas solares - Mais de 1000 imagens geolocalizadas - Georreferenciamento dos defeitos nos paineis solares com erro de 0,34 metros na latitude e 0,26 metros na longitude.
Comprehensive Direct Georeferencing of Aerial Images for Unmanned Aerial Systems Applications	Unreal Engine 4; AirSim; Ambiente 3D fictício; Georreferenciamento direto de pixels; Robot Operationg System (ROS) para experimento real	Foco na justificativa e explicação matemática do algoritmo de georreferenciamento direto de pixels. Diferencial está em apresentação de resultados em um experimento real.	Algoritmo de georreferenciamento direto de pixels implementado em Python e no Matlab com 100% de acurácia arredondando para segunda casa decimal.
Simulação de Missões de VANTs em Ambientes 3D Fictícios e Gêmeos Digitais com Georreferenciamento Direto de Pixels	Unreal Engine 4; Airsim; Blender (blender_osm e bpyproj); Ambientes 3D fictícios e gêmeos digitais; Simulações SITL; Controladores de vôo PX4 e *simple_flight*; Georreferenciamento direto de pixels; Biblioteca Python *Piexif* para geolocalização das imagens; Detecção de objetos com YOLOv5; Transmissão de imagens em tempo real com micro-framework Flask; WebODM	Foco na concepção de uma metodologia que permita integrar a simulação de VANTs em ambientes 3D fictícios e gêmeos digitais com a detecção e objetos, georreferenciamento direto de pixels e a geolocalização de imagens. Diferencial está na implementação e explicação da metodologia fim a fim, desde a transmissão de imagens em tempo real de simulação, passando pela detecção de objetos (com YOLOv5), georreferenciamento de objetos de interesse nos arquivos de imagem e teste das imagens geolocalizadas em softwares comerciais como o WebODM.	- Ambientes 3D fictício e gêmeo digital para teste dos algoritmos desenvolvidos; - Algoritmo de georreferenciamento direto de pixels em Python que permite trabalhar em qualquer sistema de referência de coordenadas. Erros máximos de 3,84 metros na latitute e 1,43 metros na longitude; - Integração com detecção de objetos em tempo real com a YOLOv5; - Código para transmitir imagens do ambiente simulado em tempo real; - Código para geolocalização de imagens e teste no WebODM com geração de ortofotos.

Capítulo 2

Fundamentação Teórica e Trabalhos Relacionados

Neste capítulo serão discutidos alguns trabalhos relacionados e a fundamentação teórica que embasa a implementação da metodologia de simulação na aplicação de integrar detecção de objetos com georreferenciamento direto de pixels e a inserção de dados no formato compatível com o usado no Sistema de Posicionamento Global - *Global Positioning System* (GPS) - nos metadados *Exchangeable Image File Format* (EXIF) dos arquivos de imagens. Assim, será possível ao final deste capítulo contextualizar melhor este trabalho com as pesquisas atuais relacionadas a geração de dados sintéticos voltadas, mas não restritas as aplicações propostas.

2.1 Motores Gráficos e Simuladores de Veículos Autônomos em Ambientes Virtuais

Com o crescimento da indústria de jogos e o avanço dos ambientes de realidade virtual, o desenvolvimento de aplicações em ambientes 3D está se tornando cada vez mais comum, com o objetivo de criar cenários virtuais cada vez mais realistas. Nesse sentido, serão apresentados alguns softwares de modelagem de cenários 3D e simuladores que aproveitam o realismo virtual para gerar dados sintéticos, com o propósito de testar algoritmos de IA e desenvolvimento de veículos autônomos.

2.1.1 Motor gráfico Unreal Engine

Unreal Engine é um motor de desenvolvimento de jogos criado pela Epic Games em 1988, tendo como um dos primeiros jogos o *Unreal Tournament*, com a modalidade de tiro em primeira pessoa. Atualmente, é utilizado em jogos de aventura, *Role Playing Game*s (RPGs), ação, esportes, corrida entre outros. Um motor gráfico é basicamente um esqueleto de um jogo, capaz de gerar não apenas os gráficos realistas, mas integrar a simulação da física, cálculos de programação e áudio, a fim de detectar quando objetos e personagens colidem, simular o comportamento de um projétil ou de um veículo, entre várias outras outras possibilidades [24].

A Unreal Engine usa a linguagem de programação C++ como base, mas facilita o desenvolvimento dos games por meio de um sistema visual de script, as Blueprints, que permite aos usuários uma maneira de programar os modelos 3D e a dinâmica de comportamento dos mesmos dentro do ambiente virtual sem o uso de uma linguagem de programação formal. Devido a esses recursos e seu poder gráfico, a Unreal alcança não apenas o publico de desenvolvimento de jogos, mas também áreas como a arquitetura, para visualização realista da parte interna de cenários residencias. Outro contexto de uso é a criação de gêmeos digitais para aplicações como o gerenciamento de instalações no geral, planejamento urbano e testes com veículos autônomos. A empresa *51World* [25] tem investido nesse sentido, criando, por exemplo, um gêmeo digital da cidade chinesa Shanghai, como mostra a Figura 2.1, de forma a permitir que operadores da cidades possam testar estratégias de otimização do ambiente urbano, como monitoramento de operações de tráfego, manutenção de pontes e até mesmo para simular inundações para planejamento de desastres [26] [20].

2.1.2 NVIDIA Omniverse

A plataforma da NVIDIA Omniverse é outra ferramenta que está em alta para criação de gêmeos digitais e geração de dados sintéticos realistas para testes de soluções voltadas a automação industrial e residencial, mobilidade de veículos autônomos, redes 5G, usinas e centros de pesquisas climáticas, entre outros. Sendo uma plataforma facilmente extensível usada para colaboração de desenvolvimento de designs 3D em tempo real com várias GPUs, Omniverse traz a integração com tecnologias complexas como *Ray Tracing* e IA por meio do uso das placas de vídeo da série *Ray Tracing Extreme* (RTX) [7].

Empresas como BMW, Ericsson, Siemens Energy, Amazon e Lockheed Martin tem usado a plataforma para a criação de gêmeos digitais fisicamente precisos de objetos, processos ou

Figura 2.1: Gêmeo digital da cidade de Shanghai desenvolvido na Unreal Engine [26].

ambientes únicos, com os dados virtuais e reais sincronizados e com tecnologias de IA. A Figura 2.2 mostra um braço mecânico que se encontra tanto em um armazém real da Amazon como em sua versão digital criado com NVIDIA Omniverse. Dessa forma, é possível treinar de forma mais otimizada as soluções para automatizar os processos de classificação e separação das caixas. Treinar os sistemas de percepção desses robôs com precisão suficiente para evitar falhas no sistema requer grandes quantidades de dados com o máximo de qualidade (realismo) possível, porém, muitas vezes esses dados são inexistentes ou insuficientes. Com o uso da plataforma de simulação é possível economizar semanas de retreinamento e aumentar a precisão do modelo [7]. A Ericsson tem investido em pesquisas usando gêmeos digitais para ajudar a simular com precisão a interação entre células 5G e o ambiente para o máximo desempenho e cobertura [27].

Apesar do poder do NVIDIA Omniverse para criação de ambientes 3D e geração de dados sintéticos realistas, a plataforma só veio ser disponibilizada gratuitamente em 2022, o que dificultou o uso da mesma no desenvolvimento desse trabalho e por isso, ela não foi utilizada na metodologia proposta. Além disso, a Unreal Engine já está integrada como motor gráfico principal para o uso do simulador de drones AirSim, que será explanado a seguir e traz recursos vantajosos para o desenvolvimento voltado a missões com VANTs.

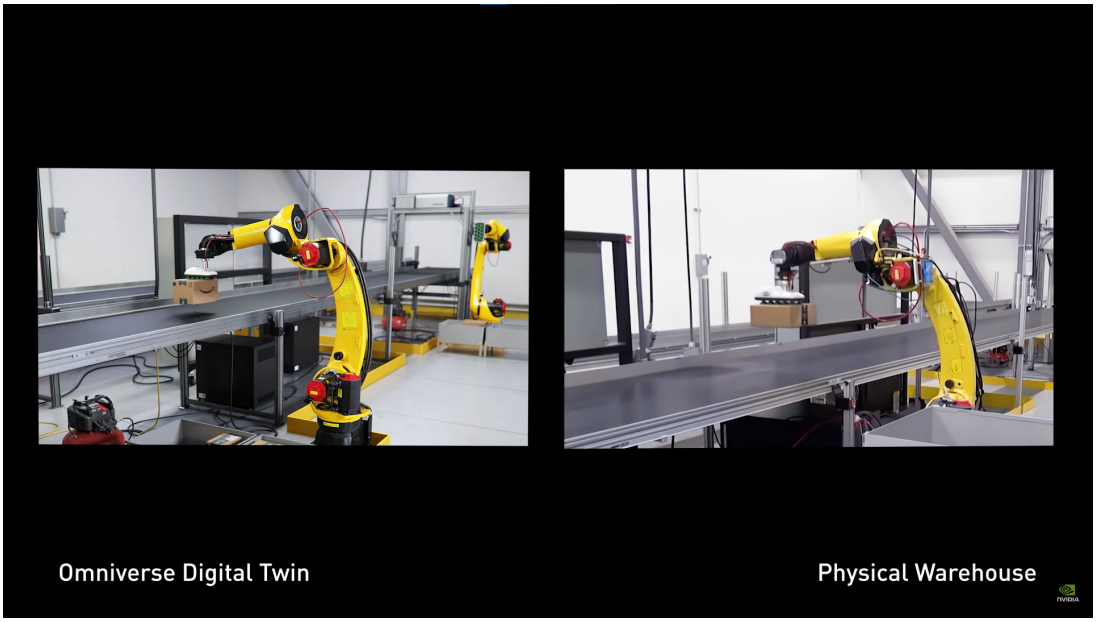

Figura 2.2: Gêmeo digital de um braço mecânico em um armazém da Amazon [7].

2.1.3 Simulador de mobilidade AirSim

O AirSim é um simulador de drones, carros e outros veículos que podem se mover de forma manual ou autônoma dentro de ambientes virtuais 3D. Desenvolvido por funcionários da *Microsoft*, foi projetado principalmente para integração com a UE4, mas também existe documentação para ser utilizado na Unity (que é outro motor gráfico). O AirSim é uma plataforma de código aberto e que visa diminuir a distância entre a simulação e a realidade para auxiliar no desenvolvimento de veículos autônomos. Dessa forma, é possível testar algoritmos de aprendizado profundo, visão computacional e aprendizado por reforço. Para isso, o AirSim disponibiliza uma API com métodos para se obter dados do veículo que está sendo simulado (aceleração, posição, atitude) assim como de componentes, como câmeras, que podem estar associadas aos veículos simulados. Além disso, a API possui métodos para o controle dos veículos de forma autônoma. O AirSim também permite que o veículo simulado interaja com o ambiente 3D, modelando não apenas a física do veículo, mas também variáveis do ambiente como gravidade, densidade, pressão do ar e campo magnético [8].

O simulador permite a integração com *firmware* de controladores de vôo populares como o *PX4* e *ArduPilot*, por meio de simulações no modo SITL. Também é possível realizar simulações no modo *Hardware-In-The-Loop* (HITL) utilizando o *PX4*. Simulações nos modos SITL e HITL permitem buscar um realismo não apenas visual mas também com relação a física de

funcionamento do veículo [8].

2.1.3.1 Simulações Software-In-The-Loop e Hardware-In-The-Loop

Ao trabalhar com simuladores de vôo de drones, tem-se o bloco responsável por simular a mecânica e física tanto do drone quanto do ambiente onde ele está voando. Nesta aplicação, as ferramentas utilizadas para isso são a Unreal Engine, para renderização do ambiente 3D, em associação ao plugin AirSim [8].

O responsável por obter os parâmetros da missão e gerenciar os sensores e motores do drone de acordo com os comandos passados pelo usuário (seja de forma manual por um joystick, por exemplo, ou automaticamente, via código) é o Controlador de vôo. Ou seja, dado o estado atual do drone (posição e velocidade) e o estado desejado pelo usuário, o Controlador de vôo gerencia os motores e sensores a fim de que o drone chegue até o objetivo. Para quadrirrotores, o estado desejado pode ser especificado por meio dos ângulos de Euler: *roll* (rolagem), *pitch* (arfagem) e *yaw* (guinada). A medida desses ângulos em um VANT (ou em robôs em geral) em função de um referencial inercial é designado como a atitude do veículo. Porém, esses valores angulares não são disponibilizados diretamente por nenhum sensor [28]. Por isso, o controlador de vôo estima o estado atual desses ângulos por meio de técnicas como o Filtro de Kalman Estendido, usando os dados brutos vindos de sensores, como o giroscópio e acelerômetro, e após esse processo de estimação, é possível que controlador gere os sinais para acionar os motores a fim de que o drone caminhe para o estado desejado [8].

Por padrão, o AirSim usa o controlador *simple_flight*, o qual é um controlador implementado para facilitar o uso da simulação. Ele já vem embutido no simulador, ou seja, não é necessário instalar ou configurar nada após instalar o AirSim [8]. Mas é possível usar também os *firmwares* de controladores reais como o PX4 e o ArduPilot, os quais são usados em controladores de vôo reais como o Pixhawk e o Arducopter. Dessa forma, é possível deixar a simulação ainda mais próxima do real, com o modo de SITL, pois nesse modo, o firmware do controlador de vôo é executado no computador (onde geralmente o simulador de vôo também está rodando). A Figura 2.3 abaixo mostra a integração dos blocos descritos.

Para se aproximar ainda mais da realidade, é possível usar os hardwares dos controladores de vôo citados, assim como sensores reais conectados aos mesmos [8]. Para monitorar e modificar a missão, pode se ter uma Ground Control Station (GCS) ou Estação de Controle Terrestre, como o QGroundControl, que pode ser usado tanto no modo SITL quanto no HITL.

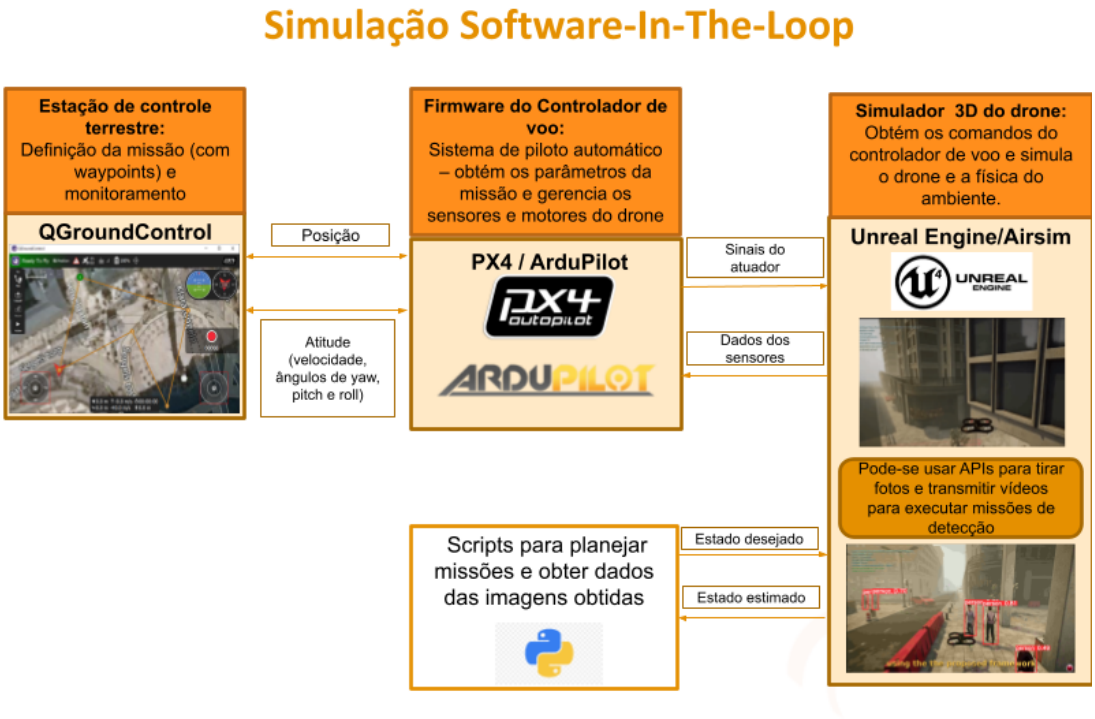

Figura 2.3: Diagrama de blocos de uma simulação no modo Software-In-The-Loop.

Mas também é possível usar scripts com base na API do AirSim para realizar o planejamento e o monitoramento da missão, por meio de dados como velocidade, posição e obter dados das imagens capturadas pela câmera embarcada ao drone simulado.

2.1.4 Simulador de mobilidade CARLA

Outro simulador para o desenvolvimento, treino e validação de sistemas de veículos autônomos interessante de ser citado é o *Car Learning to Act* (CARLA). Essa plataforma de simulação de código aberto, também desenvolvida com auxílio da UE4, foca em mobilidade de carros em ambientes urbanos. Assim, o simulador tem sido desenvolvido para dar suporte ao treinamento, prototipagem e validação de modelos de direção autônoma, incluindo percepção e controle. Um diferencial da plataforma CARLA é a disponibilização dos assets e ambientes 3D completos e sem custo. Esses ambientes incluem todo o layout urbano com prédios, pedestres, vários modelos de carros, sinais de trânsito, entre outros recursos que, com o auxílio do motor gráfico, fornecem alta qualidade de renderização e física realista, tornando a ferramenta atrativa para geração de dados sintéticos. O simulador também oferece suporte à configuração customizada de vários sensores e fornece sinais que podem ser usados para treinar estratégias

de direção, com coordenadas de GPS, velocidade, aceleração e dados detalhados sobre colisões e outras infrações. Além disso, uma ampla gama de condições ambientais podem ser especificadas, como o clima e o horário do dia, assim como a densidade de pedestres e carros, como ilustrado na Figura 2.4 [9].

Figura 2.4: Diferentes cenários e condições climáticas disponibilizados pelo simulador CARLA [9].

Para simular as dinâmicas do ambiente 3D e do agente virtual que interage com cenário, a plataforma CARLA implementa uma arquitetura cliente-servidor. O servidor é responsável por executar a simulação e renderizar as cenas, incluindo os sensores, cálculo da física envolvida, atualizações do estado no ambiente 3D e seus atores. Já o lado do cliente é responsável pela interação entre o agente autônomo e o servidor por meio de sockets. Isto acontece usando a API do CARLA, que pode ser usadas em Python ou C++, e assim, o cliente pode enviar comandos e meta-comandos para o servidor e receber as leituras dos sinais dos sensores como retorno. Os comandos que controlam o veículo incluem direção, aceleração e frenagem. Já os meta-comandos controlam o comportamento do servidor e podem ser usados para reiniciar a simulação, mudar as propriedades do ambiente e o conjunto de sensores relacionados ao agente. A Figura 2.5 mostra a dinâmica de simulação descrita, onde o lado do *Simulator* é onde se encontra o servidor e os *Users Scripts* se encontram no lado do cliente.

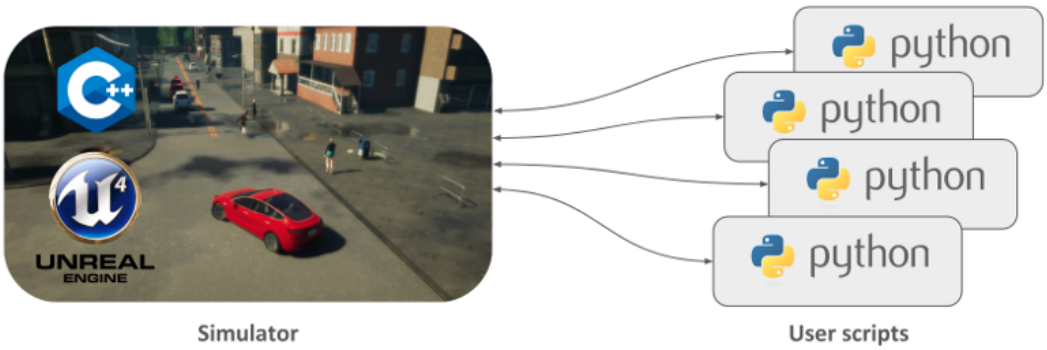

Figura 2.5: Dinâmica de funcionamento do simulador CARLA por meio de uma arquitetura cliente-servidor [9].

2.1.5 Software para modelagem 3D: Blender

O software Blender pode ser utilizado para modelagem dos modelos 3D que irão compor os cenários virtuais criados. Sendo um software de código aberto, ele possibilita todo o fluxo de criação 3D, como a modelagem, animação, simulação, renderização, iluminação, entre outras ferramentas, podendo ser usado até para edição de vídeos e como motor gráfico [29].

Assim, é possível modelar componentes de interesse para o cenário desejado e depois exportá-los para um formato compatível para o uso na UE4, por exemplo. Para aplicações como na agricultura de precisão, é possível modelar representações de ervas daninhas, como o Capim Colonião, comum em solos secos e férteis, assim como outros modelos que podem estar presentes em um cenário real, como aves [20]. A Figura 2.6 mostra exemplos desses modelos criados com auxílio do Blender.

Por ser de código aberto, outra vantagem do Blender é a integração com plugins desenvolvidos a fim de estender suas funcionalidades e torná-lo ainda mais versátil. Um exemplo é o *blender-osm* [30], que é um plugin desenvolvido em Python, capaz de importar dados do mapa do mundo de licença aberta, *OpenStreetMap* [31]. Este plugin tem uma versão gratuita e outra paga, mas na opção mais básica já é possível importar dados como prédios, rodovias, lagos, rios e vegetação (apesar de não oferecer muitos recursos de textura). Esse recurso pode ser interessante para iniciar de forma mais direta a criação de cenários baseados em lugares reais da Terra e assim, associar informações de localização de forma georreferenciada. Outro plugin para o Blender que auxilia nesta tarefa é o *bpyproj* [32] (também desenvolvido em Python) capaz de associar diferentes projeções geográficas ao ambiente 3D (importado com auxílio do *blender-osm*) usando apenas o código de referência *Spatial Reference Identifier* (SRID) - mais

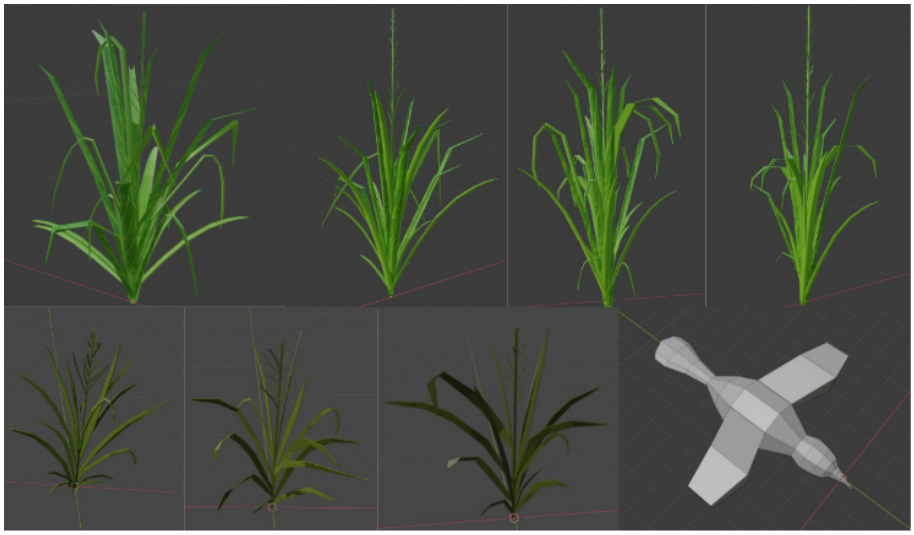

Figura 2.6: Imagens dos modelos 3D de Capim Colonião e ave construídos no Blender [20].

detalhes sobre projeções geográficas são dadas na Subseção 2.4.2.

Assim, é possível exportar a partir do Blender cenários baseados em locais reais para serem utilizados na Unreal e, integrando ao AirSim, conseguir mover o drone simulado de forma precisa usando comandos baseados em posições GPS [33]. Essa precisão é possível por meio do módulo *AirsimGeo*, o qual basicamente estende as funcionalidades da API do AirSim, adicionando métodos para localização e movimentação do drone em coordenadas GPS de forma coerente com o sistema de projeção utilizado [34]. Como o AirSim funciona com o sistema de referência *North-East-Down* (NED) por padrão, essa biblioteca permite realizar as conversões em qualquer sistema de coordenadas projetadas que o módulo Python, *pyproj* [35], consiga manipular.

2.2 Detecção de objetos

A detecção de objetos é uma técnica de visão computacional para localizar objetos de interesse em imagens ou vídeos. Os algoritmos de detecção de objetos normalmente aproveitam o aprendizado de máquina (*Machine Learning*) ou o aprendizado profundo (*Deep Learning*) para produzir resultados significativos. As tarefas desses algorítimos envolvem, além de classificar os elementos de uma imagem em diferentes categorias, identificar a posição e delimitação dos objetos em uma imagem (o que diferencia da tarefa de apenas classificar) [36].

O estado da arte na detecção de objetos pode ser separado em algoritmos que seguem dois

métodos principais:

- Métodos de um estágio, que priorizam a velocidade da inferência. Por exemplo: YOLO, SSD, RetinaNet.

- Métodos de dois estágios os quais priorizam a precisão da detecção. Modelos de exemplo: Faster R-CNN, Mask R-CNN e Cascade R-CNN.

Pode-se perceber que os modelos de detecção de objetos derivam naturalmente de Redes Neurais Convolucionais - *Convolutional Neural Network*s (CNNs) - as quais serão brevemente explanadas a seguir.

2.2.1 Redes Neurais Convolucionais

Redes Neurais Convolucionais são redes de aprendizado profundo que aprendem diretamente com os dados. Esse tipo de Inteligência Artificial (IA) é usada para encontrar padrões em imagens para reconhecer objetos, classes e categorias. Ela também pode ser bastante eficaz para classificar sinais, séries temporais e dados de áudio [37].

Redes Neurais convolucionais podem ter centenas de camadas (por isso são consideradas redes profundas), onde cada uma dessas camadas pode aprender a extrair uma característica específica da imagem. Em cada camada, filtros são aplicados às imagens com diferentes resoluções e o resultado da convolução de cada imagem com esses filtros é usado como entrada para a próxima camada. Assim, com o uso de filtros pode-se extrair características como brilho, bordas até elementos que definem exclusivamente cada objeto [38].

Assim, a estrutura de uma CNN é composta de uma camada de entrada e uma camada de saída com várias camadas ocultas entre elas. Dessa forma, as camadas ocultas executam operações (ou aplicam filtros) a fim de alterarem os dados de entrada e com isso aprenderem características específicas da imagem. Nesta etapa, existem três principais camadas que são as convolucionais, de ativação ou ReLU (que é uma das funções de ativação mais usada pra esse tipo de rede) e Pooling. A Figura 2.7 mostra como essas camadas podem estar conectadas [37].

De acordo com a Figura 2.7, pode-se dividir o algoritmo de uma CNN em duas etapas: Aprendizado das característica e a Classificação. A etapa de aprendizado das características dos dados de entrada é onde se encontram as camadas convolucionais, que são a essência das CNNs. Essas camadas contêm os *kernels*, que são matrizes de pesos (inicializados aleatoriamente e

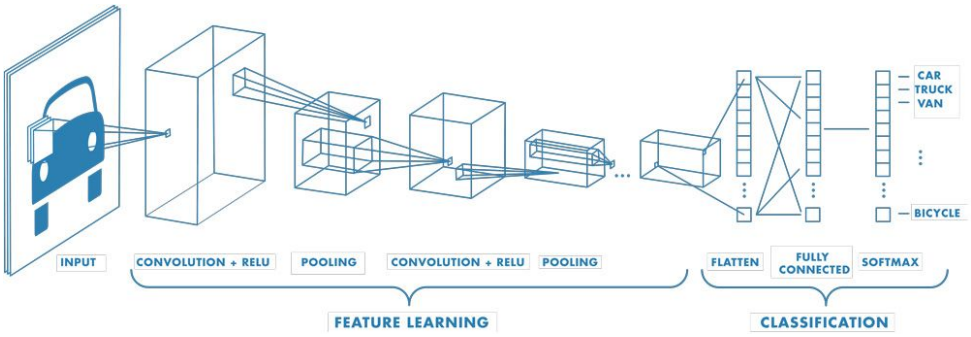

Figura 2.7: Estrutura de uma Rede Neural Convolucional [37].

atualizados a cada nova entrada durante o processo de *backpropagation*) usados para extrair as características que distinguem uma imagem da outra.

Quando se trata de classificação de imagem, essas podem ser tomadas como um conjunto de dados organizados em matrizes tridimensionais, como altura, largura e profundidade, que é determinada pela quantidade de canais de cores. Geralmente as imagens utilizam três canais de cores, RGB (*Red, Green and Blue*) com a quantidade de cada cor por pixel. A convolução ocorre quando os kernels (ou filtros), que também são matrizes, percorrem a imagem inteira (por cada um dos canais) fazendo o produto escalar com a área da imagem que está sobreposta ao filtro em questão. O resultado final de cada uma dessas operações forma o *activation map*. O passo que o kernel executa a fim de deslizar sobre a matriz de dados da imagem é chamado de *stride*. A Figura 2.8 representa como se dá o movimento do kernel no conjunto de dados. Dentro desse processo haverão diversos kernels para que diferentes característica da imagem possam ser extraídas, como linhas, curvas e bordas e a cada camada acrescida, essa filtragem se torna mais completa e precisa [39] [40] [38].

Após a etapa de convolução, os resultados passam pela função de ativação, que serve para trazer não-linearidades à rede e assim, permitir que ela aprenda qualquer qualquer funcionalidade. Além disso, as não-linearidades podem ser úteis para ajustar ou saturar (limitar) a saída gerada. Dentre as funções de ativação (*sigmoid*, *tanh* e *softmax*), a mais indicada para CNNs é a ReLU, pois é mais simples e eficiente para ser implementada sem muitas diferenças de acurácia comparada com outras funções. A função ReLU é expressa como:

$$ReLU(x) = \max(0, x), \tag{2.1}$$

ou seja, pode-se notar que ela zera os valores negativos do *activation map* [40] [41].

Por fim, na etapa de aprendizado das característica, tem-se a camada de *Pooling*, que como

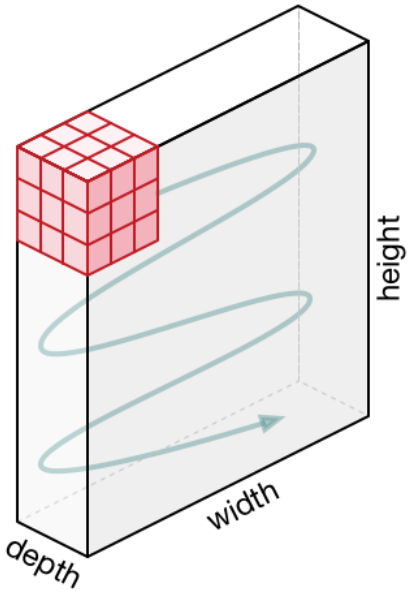

Figura 2.8: Movimento do kernel em um conjunto de dados. [39].

a camada convolucional, é responsável por reduzir o tamanho da matriz de dados (resultante da etapa anterior). Assim, é possível reduzir o poder computacional necessário para processar os dados por meio da redução de dimensionalidade, além de extrair características dominantes. Há dois tipos de *Pooling* que são mais utilizados: *Max Pooling* e *Average Pooling*. O *Max Pooling* retorna o maior valor da porção de dados coberta pelo kernel. Já o *Average Pooling* retorna a média dos valores da mesma porção de dados em questão. A Figura 2.9 ilustra os dois tipos de *Pooling* descritos. Dentre esses dois, o *Max Pooling* costuma ser o mais utilizado, pois também funciona como um supressor de ruído, descartando completamente as ativações ruidosas [39].

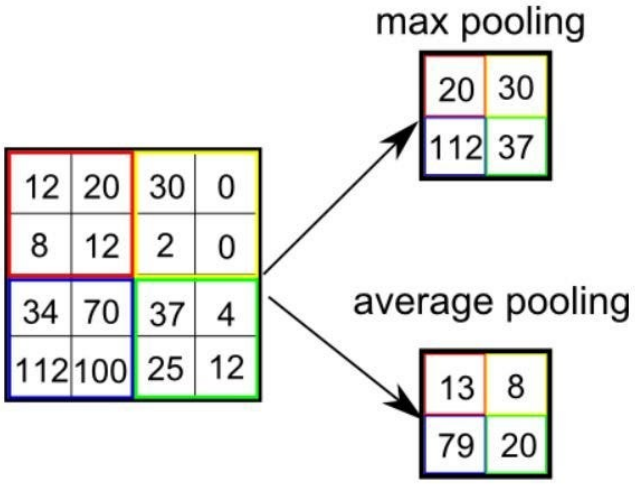

Figura 2.9: Tipos de Pooling. [39].

Já na etapa de classificação, os dados são colocados em único vetor, de forma unidimensional, em uma camada chamada de *Flatten* (o que remete a que o conjunto de dados foi achatado em um vetor). A partir dessa, os dados são submetidos a uma camada totalmente conectada (*Fully Connected*), onde encontra-se uma rede Perceptron de múltiplas camadas e assim, os dados podem ser classificados de acordo com as probabilidades para cada classe da imagem (ou dados) que está sendo classificada. A Figura 2.10 mostra de forma mais detalhada o algoritmo de uma CNN para a classificação de dígitos manuscritos [37] [39].

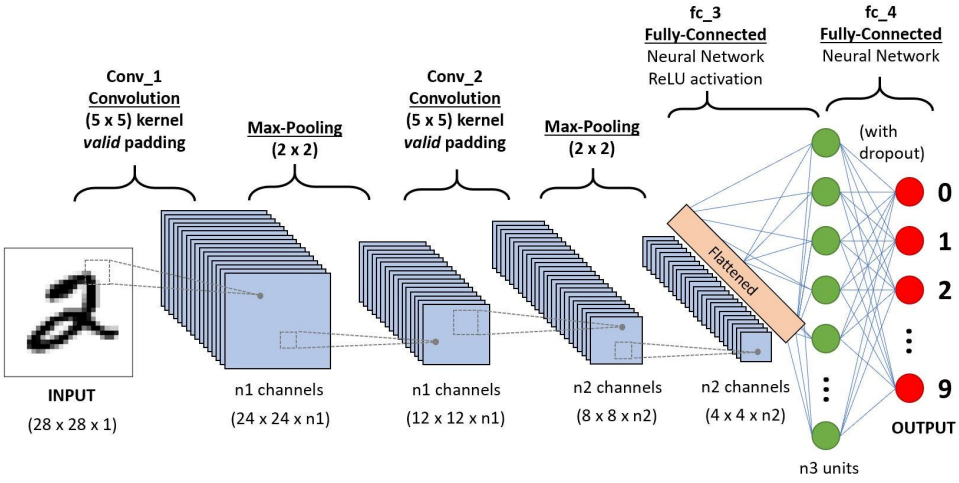

Figura 2.10: Estrutura de uma CNN utilizada para classificar dígitos manuscritos [39].

2.2.2 Introdução ao modelo *You Look Once* (YOLO)

Como já introduzido, o modelo de inteligência artificial YOLO é um método de detecção de objetos de um estágio. Nesse tipo de rede neural, a CNN produz previsões dos tipos de objetos (fazendo a classificação) presentes na imagem usando *anchor boxes*. Então essas previsões são decodificadas para gerar as caixas delimitadoras (*bounding boxes*) finais, onde tem-se a localização e dimensão que cada objeto está ocupando na imagem. A Figura 2.11 ilustra de forma geral esse processo.

2.3 Metadados EXIF

Em arquivos de imagem e vídeos, além dos dados primariamente visíveis, as fotos tiradas com smartphones e câmeras digitais modernas contêm metadados, que são informações digitais

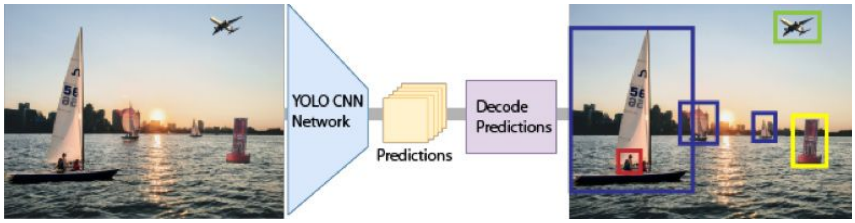

Figura 2.11: Visão geral de um processo de detecção de objetos usando YOLO [36].

sobre as condições técnicas (ou configurações) da câmera na hora da captura da foto. Esses metadados são gravados em um formato chamado EXIF (*Exchangeable Image File Format*) e comumente guardam informações como, dimensões e densidade de pixels da fotografia, data e hora que foi capturada, marca, modelo e informações técnicas da câmera (como distância focal, tipo de sensor, abertura, obturador e configurações da ISO), coordenadas GPS e etc. Esses metadados são úteis para classificar, catalogar e pesquisar fotos e por isso o padrão EXIF foi definido [42].

Apesar de apresentar questões sensíveis quanto a segurança e privacidade de informações, dentro do contexto desse trabalho, ter dados da localização de um objeto de interesse presente em uma fotografia pode ser útil para missões monitoramentos, busca e resgate. O processo de inclusão de coordenadas GPS em uma imagem é conhecido como *geotagging* [43]. Atualmente existem bibliotecas dentro da linguagem de programação Python que permitem manipular os metadados EXIF de uma imagem e assim, pode-se trabalhar em aplicações diversas onde dados de interesse podem ou não estar presentes nas fotografias. Dois exemplos dessas bibliotecas são a *exif* [44] e *Piexif* [45] (que é a usada neste trabalho). As imagens com informações de GPS gravadas nos metadados EXIF serão chamadas de imagens geolocalizadas.

2.4 Algoritmo de Georreferenciamento Direto de Pixels de uma Imagem

Georreferenciamento é o processo de atribuir a um objeto ou alvo uma localização no globo terrestre. Nesta seção serão explicados os passos adotados para implementar um algoritmo que seja capaz de georreferenciar os pixels de interesse de uma imagem obtida por câmeras carregadas por VANTs. Para isso, é necessário usar informações de posição do VANT e da posição da câmera em relação ao mesmo, assim como características do sensor de imagem e da

lente - como a resolução e o *Field of View* (FOV) - e com isso, localizar o objeto de interesse da imagem no plano geográfico. Considerando que esse processo não envolve PCTs, este algoritmo é definido mais especificamente como Georreferenciamento Direto. As subseções que se seguem foram escritas principalmente com base em [15].

2.4.1 Fundamentação matemática

Para melhor compreensão do algoritmo implementado, serão introduzidos alguns conceitos de Álgebra Linear pertinentes, como rotação de matrizes e fundamentos relacionados aos ângulos de rotação de VANTs.

2.4.1.1 Matriz de rotação

Matrizes de rotação são matrizes de transformação, quadradas, que viabilizam a rotação de um vetor no espaço euclidiano. Ou seja, com essa ferramenta matemática é possível mudar a direção de vetores sem alterar os módulos dos mesmos. Assim, dado dois sistemas de coordenadas distintos, com eixos perpendiculares entre si, é possível fazer a transformação de um ponto que se encontra no primeiro sistema para o segundo. Para demonstrar essa afirmação, dado os sistemas de coordenadas ortogonais representados na Figura 2.12, XYZ e X'Y'Z', serão utilizados os ângulos entre o eixo X' e cada eixo do sistema XYZ para demonstrar a transformação de coordenadas entre o sistemas. As projeções dos vetores de base unitária do sistema X'Y'Z' ($\hat{i'}$, $\hat{j'}$ e $\hat{k'}$) no sistema XYZ são dados pelas equações:

$$\hat{i'} = \cos A_{X'X}\hat{i} + \cos A_{X'Y}\hat{j} + \cos A_{X'Z}\hat{k}, \tag{2.2}$$

$$\hat{j'} = \cos A_{Y'X}\hat{i} + \cos A_{Y'Y}\hat{j} + \cos A_{Y'Z}\hat{k}, \tag{2.3}$$

$$\hat{k'} = \cos A_{Z'X}\hat{i} + \cos A_{Z'Y}\hat{j} + \cos A_{Z'Z}\hat{k}, \tag{2.4}$$

onde, $A_{eixo'-eixo}$ é o ângulo formado entre os eixos dos dois sistemas. e $\cos A_{eixo'-eixo}$ é o cosseno diretor do vetor no eixo.

Essas equações podem ser escritas na forma matricial como:

$$\begin{bmatrix} \hat{i'} \\ \hat{j'} \\ \hat{k'} \end{bmatrix} = \begin{bmatrix} \cos A_{X'X}\hat{i} & \cos A_{X'Y}\hat{j} & \cos A_{X'Z}\hat{k} \\ \cos A_{Y'X}\hat{i} & \cos A_{Y'Y}\hat{j} & \cos A_{Y'Z}\hat{k} \\ \cos A_{Z'X}\hat{i} & \cos A_{Z'Y}\hat{j} & \cos A_{Z'Z}\hat{k} \end{bmatrix} = R_{XYZ \to X'Y'Z'} \begin{bmatrix} \hat{i} \\ \hat{j} \\ \hat{k} \end{bmatrix}, \tag{2.5}$$

onde $R_{XYZ \to X'Y'Z'}$ é a matriz de rotação formada pelos cossenos diretores - *Directional Cosine Matriz* (DCM) - dada por:

$$R_{XYZ \to X'Y'Z'} = \begin{bmatrix} \cos A_{X'X} & \cos A_{X'Y} & \cos A_{X'Z} \\ \cos A_{Y'X} & \cos A_{Y'Y} & \cos A_{Y'Z} \\ \cos A_{Z'X} & \cos A_{Z'Y} & \cos A_{Z'Z} \end{bmatrix}. \tag{2.6}$$

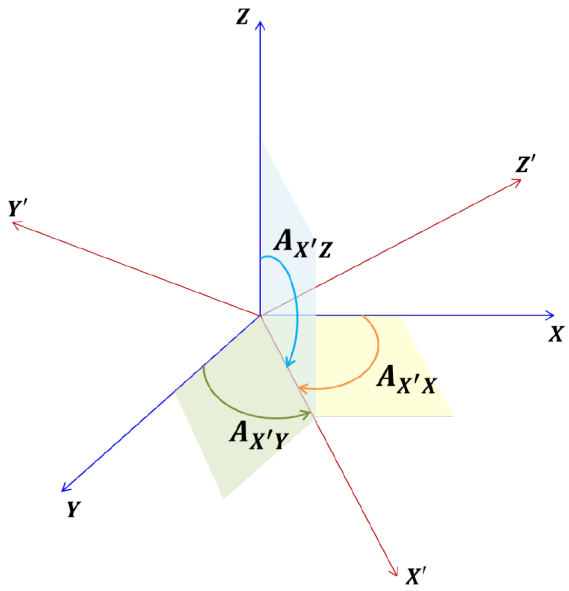

Figura 2.12: Sistemas de coordenadas ortogonais.

Dado um vetor, V, no sistema de coordenadas XYZ,

$$V = a\hat{i} + b\hat{j} + c\hat{k} = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} \hat{i} \\ \hat{j} \\ \hat{k} \end{bmatrix}, \tag{2.7}$$

pode-se representar o vetor V' como uma transformação do vetor V para sistema X'Y'Z':

$$V' = \begin{bmatrix} a'\hat{i} \\ b'\hat{j} \\ c'\hat{k} \end{bmatrix} = R_{XYZ \to X'Y'Z'} V = \begin{bmatrix} \cos A_{X'X} & \cos A_{X'Y} & \cos A_{X'Z} \\ \cos A_{Y'X} & \cos A_{Y'Y} & \cos A_{Y'Z} \\ \cos A_{Z'X} & \cos A_{Z'Y} & \cos A_{Z'Z} \end{bmatrix} \begin{bmatrix} a\hat{i} \\ b\hat{j} \\ c\hat{k} \end{bmatrix}. \tag{2.8}$$

Ou:

$$V' = R_{XYZ \to X'Y'Z'} V. \tag{2.9}$$

Para obter a transformação inversa, ou seja, do sistema X'Y'Z' para o sistema XYZ, a matriz de rotação $R_{X'Y'Z'\rightarrow XYZ}$ é dada por:

$$R_{X'Y'Z'\rightarrow XYZ} = \begin{bmatrix} \cos A_{XX'} & \cos A_{XY'} & \cos A_{XZ'} \\ \cos A_{YX'} & \cos A_{YY'} & \cos A_{YZ'} \\ \cos A_{ZX'} & \cos A_{ZY'} & \cos A_{ZZ'} \end{bmatrix}. \tag{2.10}$$

Considerando a congruência dos ângulos formados entre os eixos dos dois sistemas, ou seja:

$$A_{X'X} = A_{XX'}; A_{X'Y} = A_{YX'}; A_{X'Z} = A_{ZX'}, \tag{2.11}$$

$$A_{Y'X} = A_{XY'}; A_{Y'Y} = A_{YY'}; A_{Y'Z} = A_{ZY'}, \tag{2.12}$$

$$A_{Z'X} = A_{XZ'}; A_{Z'Y} = A_{YZ'}; A_{Z'Z} = A_{ZZ'}, \tag{2.13}$$

pode-se concluir que:

$$R_{X'Y'Z'\rightarrow XYZ} = \begin{bmatrix} \cos A_{XX'} & \cos A_{XY'} & \cos A_{XZ'} \\ \cos A_{YX'} & \cos A_{YY'} & \cos A_{YZ'} \\ \cos A_{ZX'} & \cos A_{ZY'} & \cos A_{ZZ'} \end{bmatrix} \tag{2.14}$$

$$= \begin{bmatrix} \cos A_{X'X} & \cos A_{Y'X} & \cos A_{Z'X} \\ \cos A_{X'Y} & \cos A_{Y'Y} & \cos A_{Z'Y} \\ \cos A_{X'Z} & \cos A_{Y'Z} & \cos A_{Z'Z} \end{bmatrix} = R^T_{XYZ\rightarrow X'Y'X'}. \tag{2.15}$$

Desta forma, para fazer a transformação inversa do vetor V' no sistema X'Y'Z' para o sistema XYZ, a nova matriz de rotação será dada pela matriz transposta da primeira matriz de rotação, ou seja:

$$V' = R_{XYZ\rightarrow X'Y'Z'}V \tag{2.16}$$

$$V = R_{X'Y'Z'\rightarrow XYZ}V' = R^T_{XYZ\rightarrow X'Y'Z'}V'. \tag{2.17}$$

2.4.1.2 Sequência de ângulos de Euler para aerodinâmica

Para realizar a rotação de vetores entre dois sistemas de coordenadas, pode-se utilizar a sequência de ângulos de Euler, que consiste em escolher uma sequência de 3 ângulos para projetar os eixos de um sistema no outro, e com isso, transformar um conjunto de coordenadas de um sistema para outro. Na aerodinâmica, há uma sequência específica de ângulos a qual é

mais conveniente para ser usada devido sua relação com os movimentos dos veículos aéreos. São estes: *yaw*, *pitch* e *roll*.

Em relação a VANTs, mais especificamente do tipo quadrirrotores, o movimento de *roll* é a rotação em torno do eixo X, como mostra a Figura 2.13, que permite o VANT se movimentar para esquerda ou para direita. Já o movimento de *pitch*, rotaciona o VANT em torno do eixo Y, fazendo com que o veículo se movimente para frente ou para trás. Por fim, o movimento de *yaw* é rotação em torno do eixo Z, permitindo variar a visada do VANT para a direita ou esquerda.

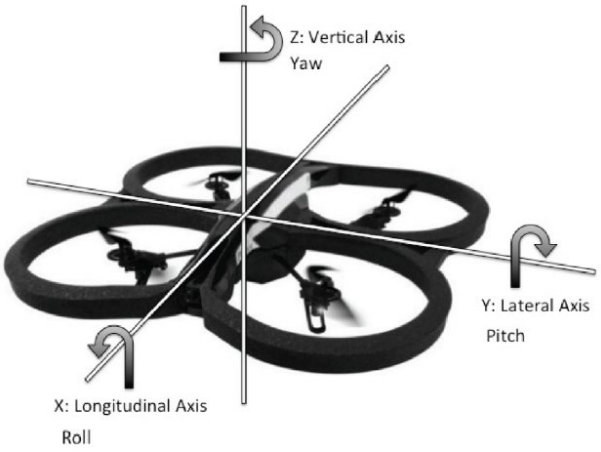

Figura 2.13: Movimentos de yaw, pitch e roll em VANTs [46].

A Figura 2.14 mostra a sequência de ângulos de Euler para aerodinâmica. A Figura 2.14a mostra a primeira rotação em torno do eixo Z, pelo ângulo (ou movimento) *yaw*, ψ, o que faz a transformação do eixo XYZ para o sistema $X_1Y_1Z_1$, onde o eixo Z é congruente ao eixo Z_1. A Figura 2.14b mostra a segunda rotação, em torno do eixo Y, pelo ângulo *pitch*, θ, movendo o sistema $X_1Y_1Z_1$ para o sistema $X_2Y_2Z_2$, onde agora o eixo Y_1 é congruente ao Y_2. Por fim, a Figura 2.14c mostra a terceira rotação, em torno do eixo X, pelo ângulo *roll*, ϕ, rotacionando do sistema de referência $X_2Y_2Z_2$ para o sistema $X'Y'Z'$. Neste a congruência está nos eixos X_2 e X'.

A matriz de rotação para a sequência de ângulos de Euler *yaw*, *pitch* e *roll*, R_{YPR}, é dada por:

$$R_{YPR} = \begin{bmatrix} \cos\psi\cos\theta & \text{sen}\psi\cos\theta & -\text{sen}\theta \\ \cos\psi\,\text{sen}\theta\,\text{sen}\phi - \text{sen}\psi\cos\theta & sin\psi sin\theta sin\phi + cos\psi cos\phi & \cos\theta\,\text{sen}\phi \\ \cos\psi\,\text{sen}\theta\cos\phi + \text{sen}\psi\,\text{sen}\phi & sin\psi sin\theta cos\phi - cos\psi sin\phi & \cos\theta\cos\phi \end{bmatrix}. \tag{2.18}$$

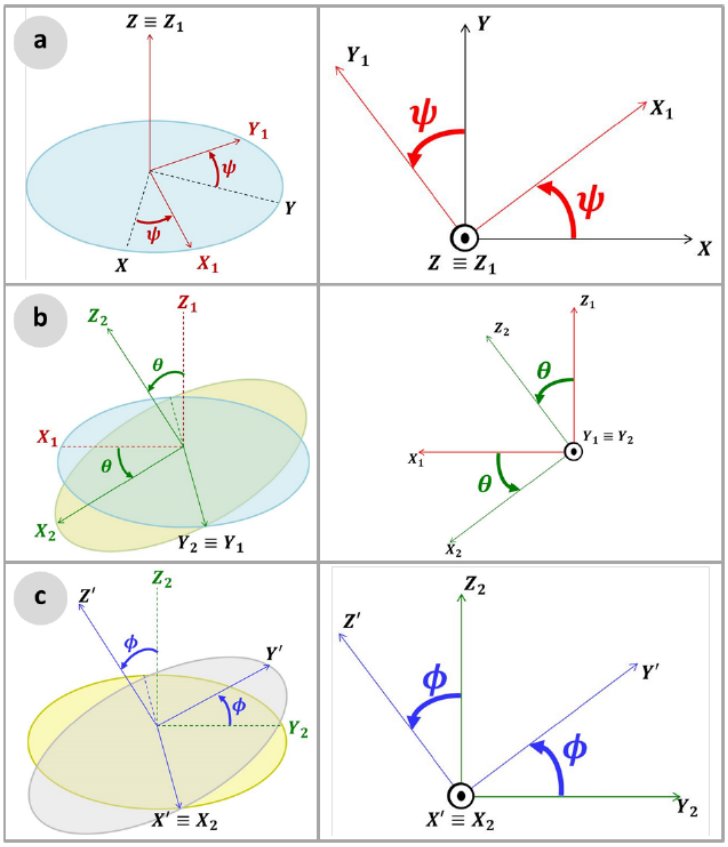

Figura 2.14: Sequência de ângulos de Euler para aerodinâmica: a) Yaw; b) Pitch, e c) Roll [15].

Os valores dos ângulos de Euler são dados por:

$$\mathrm{tg}\psi = \frac{\cos A_{XY'}}{cosA_{XX'}} \tag{2.19}$$

$$\mathrm{sen}\theta = -\cos A_{XZ'} \tag{2.20}$$

$$\mathrm{tg}\phi = -\frac{cosA_{YZ'}}{\cos A_{ZZ'}} \tag{2.21}$$

2.4.2 Sistemas de Coordenadas

Nesta Subseção serão introduzidos conceitos em relação aos sistemas de coordenadas necessários para o melhor entendimento de como a localização do pixel georreferenciado pode ser interpretada. Para se chegar a um sistema de coordenadas geográficas é necessário utilizar uma projeção cartográfica, as quais tentam retratar a superfície da Terra, ou uma parte dela, em um plano. Ou seja, as projeções cartográficas procuram transformar a Terra do seu plano tridimensional (3D) para a forma plana ou bidimensional (2D). Existem 3 famílias de projeções cartográficas, sendo elas as planares, cilíndricas e cônicas, como mostra a Figura 2.15. Cada projeção tem suas vantagens e desvantagens, sendo que a escolha da melhor projeção cartográfica depende da escala do mapa que será produzido e os objetivos para os quais será usado. Assim, como resultado do processo de projeção, cada mapa mostra distorções de ângulo, distância e área. Uma projeção pode combinar várias destas características ou pode ser um compromisso que distorce todas as propriedades de área, distância, e conformidade angular, dentro de alguns limites aceitáveis, sendo impossível preservar todas as características simultaneamente [47].

Um sistema de referência de coordenadas define como o mapa bidimensional projetado se relaciona com lugares reais na Terra. Dessa forma, é possível especificar cada lugar da terra por um conjunto de 3 números, chamados coordenadas [47].

2.4.2.1 Sistemas de coordenadas geográficas

Os sistemas de coordenadas geográficas representam as coordenadas de uma localização na superfície da Terra em latitude e longitude (em graus) e por vezes um valor de altura (ou altitude). As linhas referente a latitude são as que correm paralelas ao equador (por isso, são chamadas de paralelos), que é linha de referência desta medida, e dividem o globo terrestre em

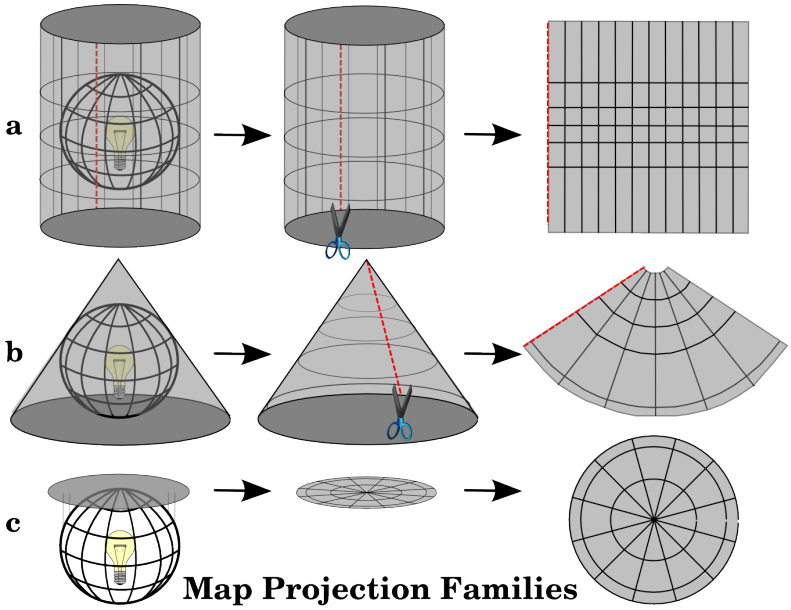

Figura 2.15: Os 3 tipos de projeções cartográficas: a) cilíndricas, b) cônicas e c) planares [47].

180 seções igualmente espaçadas de a norte a sul. Assim, cada hemisfério possui 90 seções, cada uma representando um grau de latitude, contadas a partir 0° (no equador) até os polos. Para simplificar a digitação de mapas, geralmente são atribuídos valores positivos para os graus de latitude no hemisfério norte (0 a 90°) e valores negativos para o hemisfério sul (0 a -90°) [47].

Já as linhas referentes a longitude estão dispostas perpendicularmente ao equador e convergem nos polos. A linha de referência para a longitude (o meridiano principal) vai do polo Norte ao polo Sul passando por Greenwich, Inglaterra. Linhas subsequentes de longitude são medidas de 0 a 180 graus leste (assumindo valores positivos) ou oeste (assumindo valores negativos) a partir do meridiano principal. Logo, cada meridiano possui o seu anti-meridiano, isto é, um meridiano oposto que junto com ele formam uma circunferência, e todos os meridianos possuem o mesmo tamanho. O valor de cada unidade de longitude é bem definido, pois, dado que a metade da circunferência da Terra tem 2003,93 Km, dividindo esse valor por 180, conclui-se que um grau equivale a 111,133 Km. Dividindo um grau por 60, tem-se que um minuto equivale a 1852,22 m. Por último, dividindo-se um minuto por 60, tem-se que um segundo equivale a 30,87 m [47].

Dessa forma, as coordenadas geográficas podem ser representadas em:

- Graus, minutos, segundos - *Degrees Minutes and Seconds* (DMS) - Nesta representação, cada grau é dividido em 60 minutos, que por sua vez se subdividem em 60 segundos cada.

E os segundos podem ser divididos decimalmente em frações cada vez menores. Neste sistema usam-se letras no final da coordenada para indicar se a coordenada se refere ao hemisfério Norte ("N"de *North*) ou Sul ("S"de *South*) em latitude, e em longitude se está a Leste ("E"de *East*) ou Oeste ("W"de *West*). Ex: 40° 45’ 300” N (coordenada de latitude).

- Minutos decimais - *Decimal Minutes* (DM) - Nesta representação, cada grau é dividido em 60 minutos, que por sua vez são divididos decimalmente. O mesmo vale sobre as letras usadas no final da coordenada em relação a representação DMS. Ex: 70° 59,04’ W (coordenada de longitude).

- Graus decimais - *Decimal Degrees* (DD) - Neste sistema, cada grau é dividido em frações decimais e são usados os valores negativos para o Sul (latitude) e o Oeste (longitude). Ex: 40.76867; -75.9678.

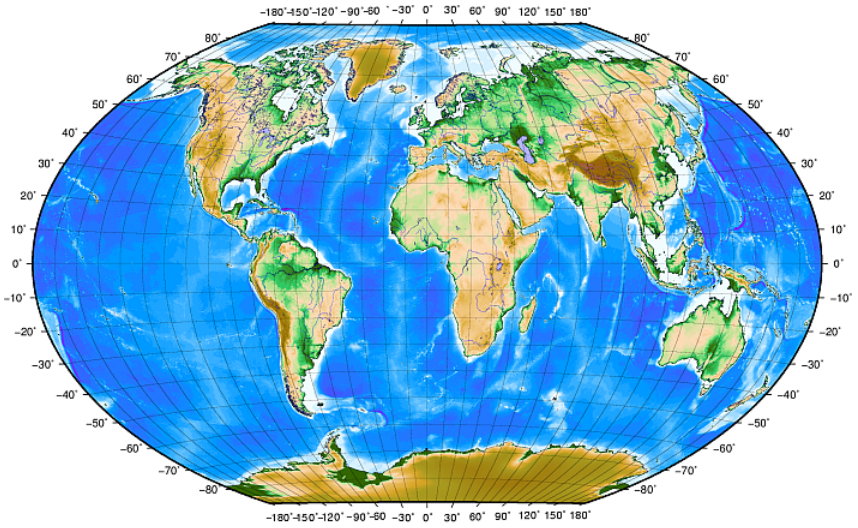

Figura 2.16: Sistemas de coordenadas geográficas com linhas de latitude paralelas ao equador e linhas de longitude com o meridiano principal em Greenwich [47].

2.4.2.2 Sistema de Coordenadas Universal Transversa de Mercator (UTM)

O sistema de referência de coordenadas UTM é baseado no plano cartesiano (com eixos X e Y) e usa o metro como unidade para medir distâncias e determinar a posição de um objeto. É derivado de uma projeção cilíndrica e sendo uma projeção conforme, preserva os ângulos e a forma em pequenas áreas. Por não acompanhar a curvatura da Terra, seus pares de coordenadas também são chamados de coordenadas planas [48]. Para evitar muita distorção nas medidas,

neste sistema a Terra é divida em 60 fusos iguais, com 6 graus de largura de leste a oeste. A origem do sistema é a linha do Equador e cada fuso possui seu Meridiano Central como referência vertical. Assim, as zonas UTM são numeradas de 1 a 60 começando no antimeridiano (zona 1, a 180° de longitude oeste) e avançando a contagem para leste, voltando ao antimeridiano (zona 60, a 180° de longitude leste), como representado na Figura 2.17 [47]. Logo, para que a posição de uma localidade da Terra se dê corretamente, é necessário indicar a que fuso do sistema UTM as coordenadas pertencem, uma vez que o mesmo par de coordenadas se repete nos 60 fusos diferentes.

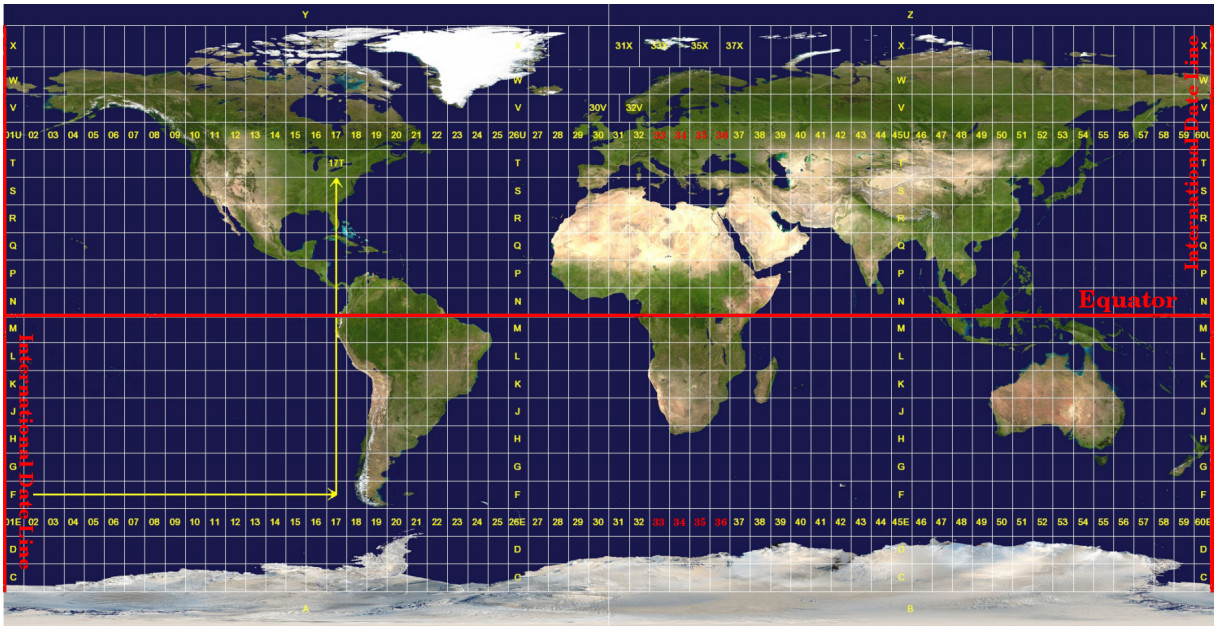

Figura 2.17: Mapa global dividido com as zonas do sistemas de coordenadas UTM [47].

A Figura 2.18 mostra como é representado um fuso UTM. Neste está indicado o meridiano central do fuso (representando o eixo Y), cujo o valor é de 500.000 metros, e a linha do Equador (representando o eixo X), que tem valor de 0 metros para coordenadas no hemisfério norte e 10.000.000 metros para coordenadas no hemisfério sul. Dessa forma a posição de uma coordenada em UTM é dada pelo número da zona, a letra referente se a localidade está no hemisfério norte (N) ou sul (S), o valor do norte (eixo Y) e leste (eixo X). As constantes $500.00\ m$ e $10.000.000\ m$ são chamadas respectivamente de Falso Leste e Falso Norte. Para evitar coordenadas negativas, essas constantes são acrescidas à origem do sistema de coordenadas, onde, para o hemisfério sul, os valores são decrescentes, a partir de $10.000.000\ m$, em direção ao Polo Sul. Já com referência ao eixo X, os valores são crescentes, a partir de $500.000\ m$, indo em direção ao leste. Como convenção, atribui-se a letra N para coordena-

das no eixo Y (sentido norte-sul) quando a localidade está no hemisfério norte, e a letra *S* para quando a localidade está no hemisfério sul. A letra *E* é usada para representar as coordenadas no eixo X (sentido leste-oeste) [47].

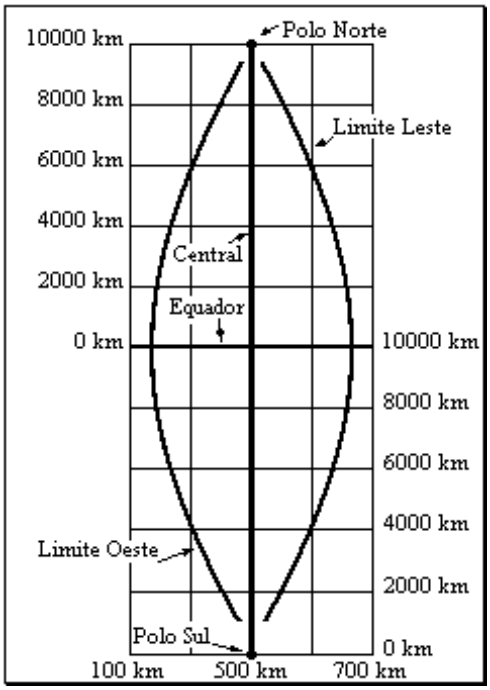

Figura 2.18: Fuso UTM [47].

Sistemas Geodésicos de Referência (SGR)

Para contribuir ainda mais com a segurança e precisão dos dados espaciais, existem os Sistemas Geodésicos de Referência (SGR), que consistem em elipsoides de referência para cálculos e medições. Estes elipsoides consistem em superfícies matemáticas que se aproximam do geoide, ou seja, do formato real da Terra. O elipsoide pode ter como base um vértice na superfície terrestre que conecta o geoide ao elipsoide (topocêntrico) ou ter como referência o centro de massa da Terra (geocêntrico) [49] [50].

Também conhecidos como datum, dentre os SGR conhecidos tem-se o SIRGAS 2000 (Sistema de Referência Geocêntrico para as Américas) e o WGS84 (World Geodetic System). O primeiro é o sistema de referência utilizado atualmente no Brasil de forma oficial (desde 2005). Já o WGS84, definido em 1984, é um datum global usado para fornecer posicionamento e navegação em qualquer parte do mundo. Por isso, esse SGR é utilizado como referência no sistema GPS [49]. Associado ao datum utilizado pode-se indicar o SRID referente ao Sistema

de Referência de Coordenadas (SRC), o qual dá sentido e localização aos valores das coordenadas utilizados na projeção adotada. Em softwares que utilizam os sistemas de informação geográficas, com o QGIS, o SRID é apresentado por um código chamado de *European Petroleum Survey Group* (EPSG) [51].

Ter o conhecimento da zona UTM em que se está realizando uma missão com VANTs integrada com georreferenciamento é importante para correta geolocalização do veículo e dos processos gerados a partir desses dados (como imagens e pixels georreferenciados). Como será visto a seguir, para correta implementação do algoritmo de georreferenciamento direto dos pixels, é necessário converter as coordenadas geográficas de graus decimais para UTM e para isso, é preciso conhecer a zona UTM correspondente a localização do drone e o SGR ou EPSG adotado.

Sistemas de referência para navegação

Ao se referir a posição, velocidade, aceleração e orientação de um veículo aéreo ou objeto de interesse relacionado, é importante mencionar o sistema de referência adotado para melhor entendimento do comportamento dos elementos da missão. Esses sistemas procuram comparar o movimento do objeto de interesse a um padrão conhecido no referencial adotado. Nesse sentido, existem vários sistemas de referência (conhecidos em inglês como *reference frames*) que podem ser usados para medir o movimento de um objeto, dependendo do tipo de aplicação e resultados desejados. Mas, existem aqueles mais comuns na aerodinâmica como: o Sistema do Sensor, Sistema do Corpo (*Body Frame*), Sistema ECEF e o Sistema NED [52] [53].

A origem e orientação de um sistema de referência podem ser diferentes de uma aplicação para outra, desde que, em uma mesma aplicação - onde se deseja fazer uma transformação de coordenadas de um sistema para o outro, por exemplo (como é o caso desse trabalho) - sejam respeitadas as convenções adotadas desde o sistema de origem até o final [52]. Entretanto, existem convenções comumente usadas principalmente no Sistema ECEF e no Sistema NED, os quais serão descritos com mais detalhes a seguir.

Sistema ECEF – Devido aos movimentos de translação e rotação da Terra, é necessário ter um sistema que esteja preso a ela. Isso é interessante pois parâmetros comuns em missões de vôo são expressos em relação ao solo (como *waypoints* e locais de pouso na superfície). Assim, o Sistema ECEF é um sistema de referência o qual sua origem está no centro da Terra e os seus

3 eixos ortogonais estão fixos no globo terrestre. Como mostra a Figura 2.19, o eixo *X* aponta para a interseção entre o primeiro meridiano - *IERS Reference Meridian* (IRM) - e a linha do Equador (ou seja, latitude 0° e longitude 0°); o eixo *Y* aponta para a linha do Equador onde a longitude é de 90°, e o eixo *Z* aponta para para o Polo Norte. Este sistema rotaciona com a Terra com uma velocidade angular de aproximadamente 15° por hora (totalizando 360° em 24h) [52] [53].

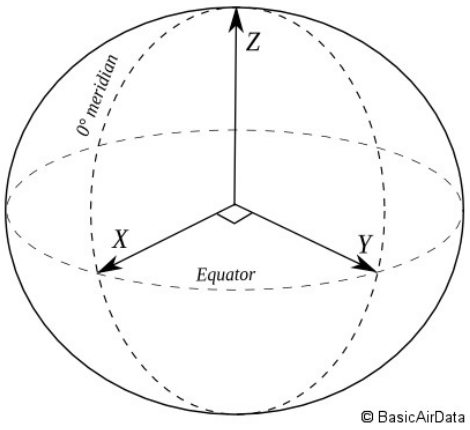

Figura 2.19: Sistema de Referência ECEF [53].

Sistema NED – Assumindo uma pequena área do globo terrestre onde o o veículo está em operação, é possível negligenciar a curvatura da Terra e assumir um sistema de referência local para descrever o movimento do veículo. Para isso, um plano tangente a superfície da terra relacionado aquela região reduzida é definido e chamado de Plano Tangente Local (PTL). Para descrever as trajetórias nesse PTL, é usado o Sistema NED, onde o eixo *x*, ou "*North*", aponta para o norte verdadeiro (na direção do meridiano local); o eixo *y*, "*East*", aponta para o leste na direção paralela ao local, e o *z*, "*Down*", é perpendicular ao outros dois e aponta para o interior da Terra, para completar o sistema de referência com os eixos ortogonais, como mostra a Figura 2.20. A origem do Sistema NED é fixo de acordo com as coordenadas no Sistema ECEF, estando relacionado a um ponto da superfície no local onde se está operando o veículo [52] [53].

Fazendo uma analogia ao que será visto nos passos de descrição do algoritmo de georreferenciamento de pixels (Subseção 2.4.3) com o que foi visto sobre os sistemas de referência para navegação, o sensor associado ao VANT onde se tem o Sistema do Sensor é a câmera; o Sistema do Corpo está atrelado ao próprio VANT e as coordenadas serão transformadas até a

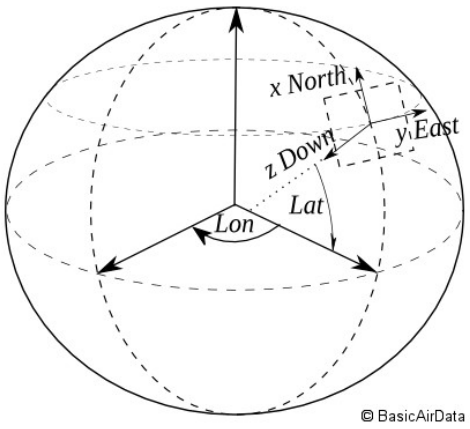

Figura 2.20: Sistema de Referência NED [53].

representação no GPS - no datum *World Geodetic System* (WGS84).

2.4.3 Descrição do Algoritmo de Georreferenciamento Direto de Pixels

Dado os fundamentos para compreensão do algoritmo, irão ser apresentados abaixo cada passo do processo de georreferenciamento direto de imagens, o que basicamente consiste em rotações, ou transformações, sucessivas entre os sistemas de referência, desde a imagem até o plano geográfico.

2.4.3.1 Sistema de Referência da Imagem para a Câmera

Câmeras fotográficas são dispositivos capazes de gravar as imagens que geralmente partem de um plano tridimensional para um superfície bidimensional, ou plano da imagem. O princípio de construção das câmeras parte de uma caixa preta com uma pequena abertura, como mostra a Figura 2.21, por onde os fótons, os quais refletem no objeto no mundo tridimensional, passam e formam a imagem no lado oposto ao da abertura na caixa, onde se encontra um sensor óptico.

Este sistema pode ser modelado matematicamente introduzindo um sistema de coordenadas de referência onde se encontra a abertura da câmera, chamado Sistema da Câmera (C), no qual a posição do objeto de interesse é plotado nas coordenadas x_C, y_C, z_C. Outro sistema de referência que irá ser considerado é o Sistema da Imagem (i), o qual fica no plano da imagem, ou plano focal, onde a posição da imagem é plotada em termos das coordenadas (x_i, y_i). Este modelo é conhecido como Câmera Pinhole e está representado da Figura 2.22.

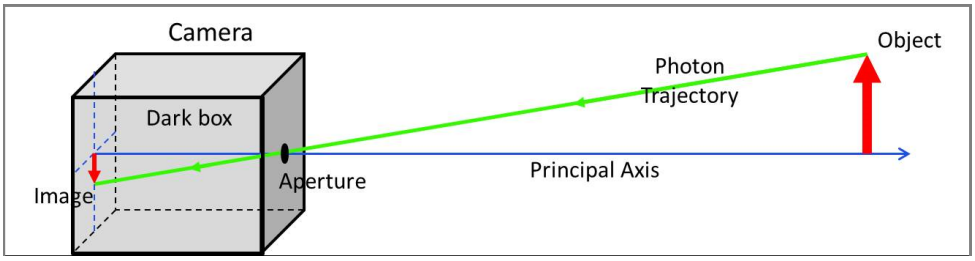

Figura 2.21: Modelo da câmera fotográfica [15].

Para facilitar as operações matemáticas, uma construção geometricamente equivalente pode ser obtida espelhando o plano da imagem, ou o plano da projeção, a fim de posicioná-lo do lado oposto no Sistema da Câmera, permitindo assim trabalhar apenas com os valores positivos dos eixos, como mostra o esquema na Figura 2.23.

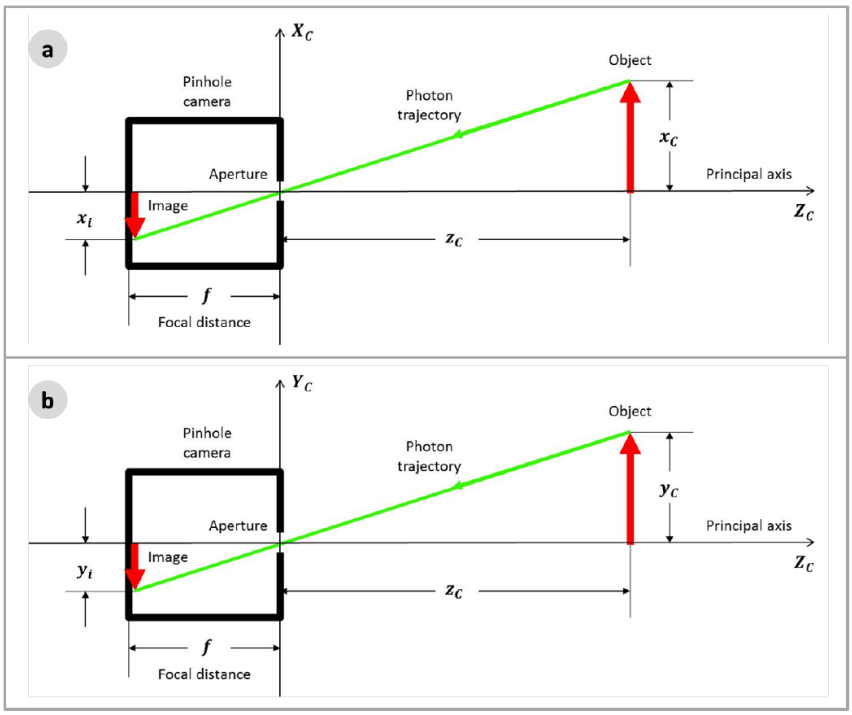

Figura 2.22: Modelo de Câmera Pinhole: a) Vista pelo eixo X; b) Vista pelo eixo Y [15].

A partir da Figura 2.23, é possível obter a relação entre as coordenadas x_i, no Sistema da Imagem (i), e x_C, no Sistema da Câmera (C),

$$\frac{x_i}{f} = \frac{x_C}{z_C} \therefore x_i = f\frac{x_C}{z_C}, \tag{2.22}$$

onde f é a distância focal.

Também, da Figura 2.23b obtém-se:

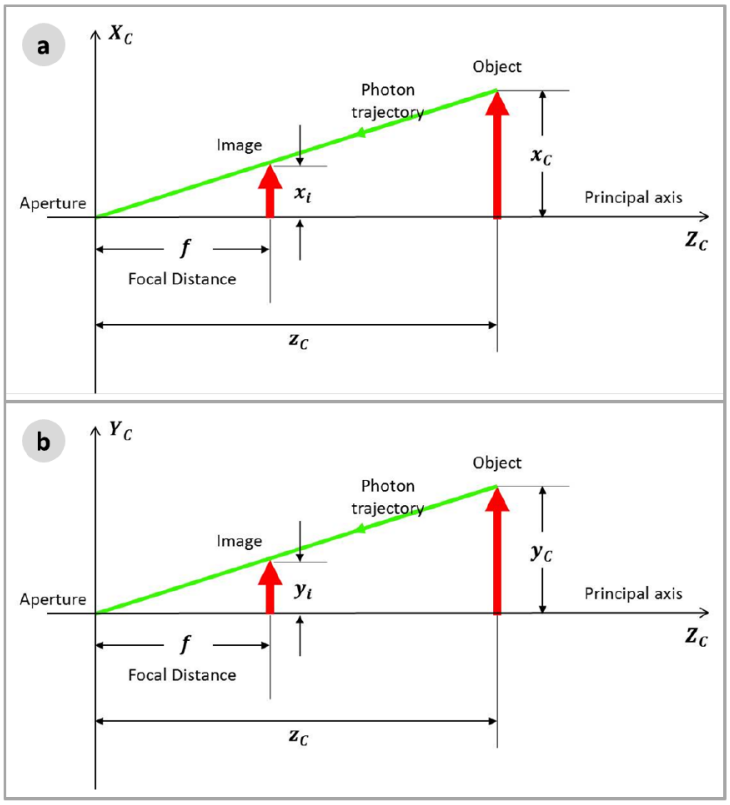

Figura 2.23: Modelo de Câmera equivalente: a) Vista pelo eixo X; b) Vista pelo eixo Y [15].

$$\frac{y_i}{f} = \frac{y_C}{z_C} \therefore y_i = f\frac{y_C}{z_C} \tag{2.23}$$

Essas equações são válidas apenas para imagens retificadas.

A Figura 2.24 mostra o mesmo esquema, mas de uma perspectiva mais detalhada, onde o Sistema da Imagem possui os eixos U e V:

$$u - c_x = x_i, \tag{2.24}$$

$$v - c_y = y_i, \tag{2.25}$$

onde, c_x e c_y são as coordenadas do ponto principal da imagem (ou seja, o ponto central).

Assim, combinando as equações 2.24 e 2.25 com as equações 2.22 e 2.23 obtém-se:

$$u = f\frac{x_C}{z_C} + c_x, \tag{2.26}$$

$$v = f\frac{y_C}{z_C} + c_y. \tag{2.27}$$

Para permitir os cálculos como rotações e transformações entre os sistemas de referência, as equações 2.26 e 2.27 podem ser escritas como:

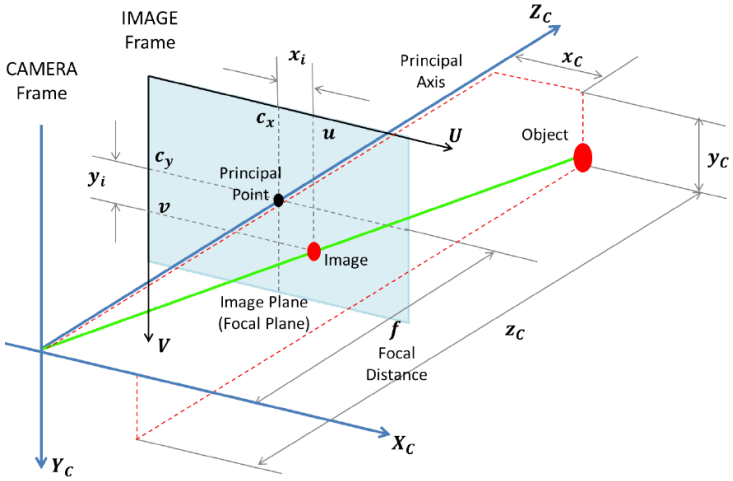

Figura 2.24: Modelo equivalente da Câmera de forma mais detalhada [15].

$$z_C \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix}. \tag{2.28}$$

As coordenadas u e v no Sistema da Imagem, i, são dadas em unidade de pixels, o que facilita o uso do algoritmo com a saída de algoritmos de inteligência artificial para detecção de objetos. Já as coordenadas do objeto de interesse no Sistema da Câmera (C) é geralmente dado em metros. Logo, a distância focal, f, e as coordenadas do ponto central da imagem (c_x e c_y) devem ser dados em pixels.

A distância focal, f, geralmente é dada em milímetros. Assim, para convertê-la para pixels, deve-se multiplicar pelo Tamanho da Imagem em pixels e dividir pelo Tamanho do Sensor, em milímetros, de forma que:

$$f[px] = f[mm] \frac{Tamanho\ da\ Imagem[px]}{Tamanho\ do\ Sensor[mm]} \tag{2.29}$$

Quando se tem diferentes resoluções de pixel para as dimensões x e y da imagem, a Equação 2.28 é escrita como:

$$z_C \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix}. \tag{2.30}$$

Assim, f_x e f_y, em pixels, podem ser calculados baseados na distância focal (f, em milímetros), nas dimensões da imagem ($Largura\_da\_Imagem$ e $Altura\_da\_Imagem$, em pixels) e do sensor ($Largura\_do\_Sensor$ e $Altura\_do\_Sensor$, em milímetros):

$$f_x = f\frac{Largura\_da\_Imagem}{Largura\_do\_Sensor}, \tag{2.31}$$

$$f_y = f\frac{Altura\_da\_Imagem}{Altura\_do\_Sensor}. \tag{2.32}$$

A Equação 2.30 também pode ser escrita como:

$$z_C Pi = KP_C, \tag{2.33}$$

onde $P_i = [u, v, 1]^T$ é o vetor com as coordenadas do objeto no Sistema da Imagem, $P_C = [x_C, y_C, z_C]^T$ é o vetor com as coordenadas do objeto no Sistema da Câmera e K é a matriz com os parâmetros intrínsecos da câmera e pode ser obtida durante o processo de calibração da mesma.

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.34}$$

Assim, as equações 2.30 e 2.33 podem ser reescritas como:

$$z_C \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix}. \tag{2.35}$$

O objetivo é obter as coordenadas do objeto em metros no Sistema da Câmera a partir das coordenadas do objeto em pixels no Sistema da Imagem (ou seja, a partir da imagem obtida pela câmera). Assim, a Equação 2.35 é reescrita como:

$$\begin{bmatrix} x_C \\ y_C \\ z_C \end{bmatrix} = K^{-1} z_C \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{2.36}$$

Considerando que z_C não é conhecido, a Equação 2.36 é dividida inicialmente por z_C para que possa ser resolvida, resultando no vetor P'_C:

$$P'_C = \begin{bmatrix} \frac{1}{z_C} x_C \\ \frac{1}{z_C} y_C \\ \frac{1}{z_C} z_C \end{bmatrix} = \begin{bmatrix} x'_C \\ y'_C \\ 1 \end{bmatrix} = K^{-1} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix}. \tag{2.37}$$

Em passos futuros, z_C irá ser determinado de forma que permita obter os valores de interesse, x_C e y_C.

2.4.3.2 Sistema de Referência da Câmera para o Gimbal

O próximo passo será a transformação das coordenadas do Sistema da Câmera (C) para o Sistema do Gimbal (G). Esse sistema tem seu eixo X, X_G, paralelo ao eixo Z do Sistema da Câmera, Z_C. O eixo Y do Sistema do Gimbal, Y_G, é paralelo ao eixo X do Sistema da Câmera, X_C, e o eixo Z do Sistema do Gimbal, Z_G, é paralelo ao eixo Y do Sistema da Câmera, Y_C. A Figura 2.25 mostra os sistemas de referência descritos.

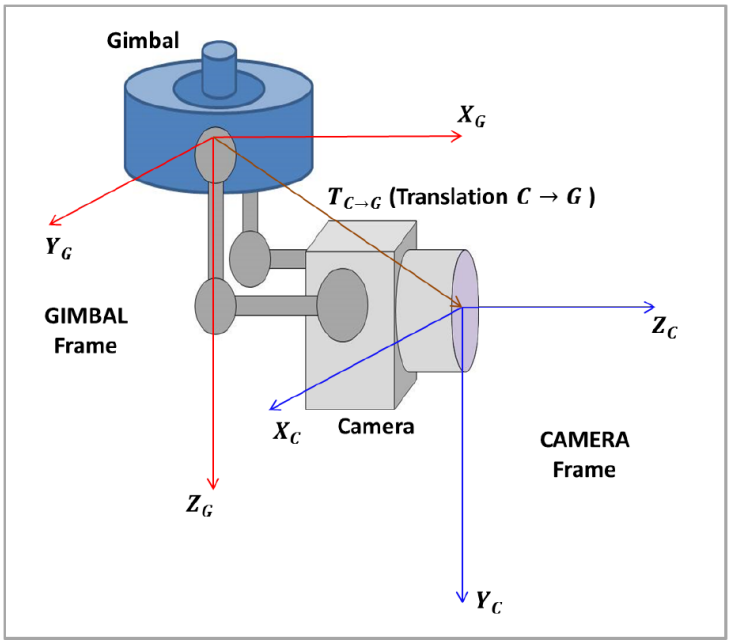

Figura 2.25: Sistemas de referência da Câmera e do Gimbal [15].

A matriz de rotação do Sistema da Câmera para o Sistema do Gimbal, $R_{C\to G}$, pode ser obtida por meio da projeção do Sistema do Gimbal no Sistema da Câmera, sendo dada por:

$$R_{C\to G} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{2.38}$$

Assim, a posição relativa do objeto no Sistema do Gimbal, P'_G, é obtida por:

$$P'_G = R_{C\to G} P'_{C'} \tag{2.39}$$

ou,

$$\begin{bmatrix} x'_G \\ y'_G \\ z'_G \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x'_C \\ y'_C \\ z'_C \end{bmatrix}. \tag{2.40}$$

Quando há diferenças na posição das origens dos sistemas, como mostra a Figura 2.25, um vetor de translação da Câmera para o Gimbal ($T_{C\to G}$) deve ser considerado. Logo, a posição relativa do objeto no Sistema do Gimbal é dada por:

$$P'_G = R_{C\to G}P'_C + T_{C\to G} \tag{2.41}$$

2.4.3.3 Sistema de Referência do Gimbal para o VANT

O próximo passo será transformar as coordenadas do objeto no Sistema do Gimbal para o Sistema do VANT, o qual tem sua origem no centro de gravidade do VANT, como mostra a Figura 2.26. Para isso, é necessário usar a Matriz dos Cossenos Diretores do Sistema do Gimbal para o Sistema do VANT, $R_{G\to VANT}$, obtida com auxílio dos ângulos de Euler: *yaw*, *pitch* e *roll*. Assim, essa transformação de coordenadas entre os sistemas de referência é dada por:

$$P'_{VANT} = R_{G\to VANT}P'_G + T_{G\to VANT}, \tag{2.42}$$

onde, $T_{G\to VANT}$ é a matriz de translação entre o Sistema do Gimbal e o Sistema do VANT.

Assim, substituindo a Equação 2.41 na Equação 2.42:

$$P'_{VANT} = R_{G\to VANT}(R_{C\to G}P'_C + T_{C\to G}) + T_{G\to VANT} \tag{2.43}$$

$$= R_{G\to VANT}R_{C\to G}P'_C + R_{G\to VANT}T_{C\to G} + T_{G\to VANT} \tag{2.44}$$

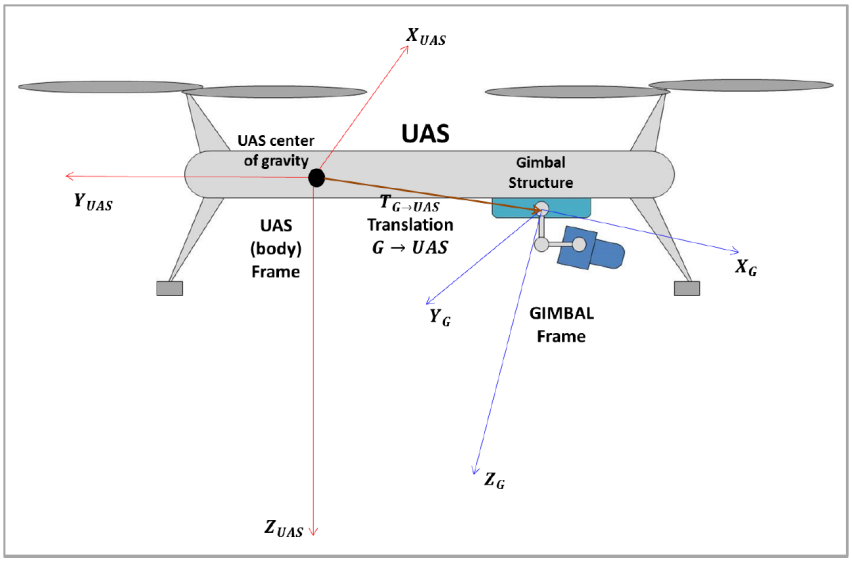

Figura 2.26: Sistemas de referência do VANT e Gimbal [15].

2.4.3.4 Sistema de Referência do VANT para o NED

Neste passo, irá ser executada a transformação das coordenadas do objeto do Sistema do VANT para o Sistema NED. A Figura 2.27 representa os sistemas de referência, também considerando o vetor de translação $T_{VANT \rightarrow NED}$, o qual representa a posição da origem do Sistema do VANT em relação ao Sistema NED.

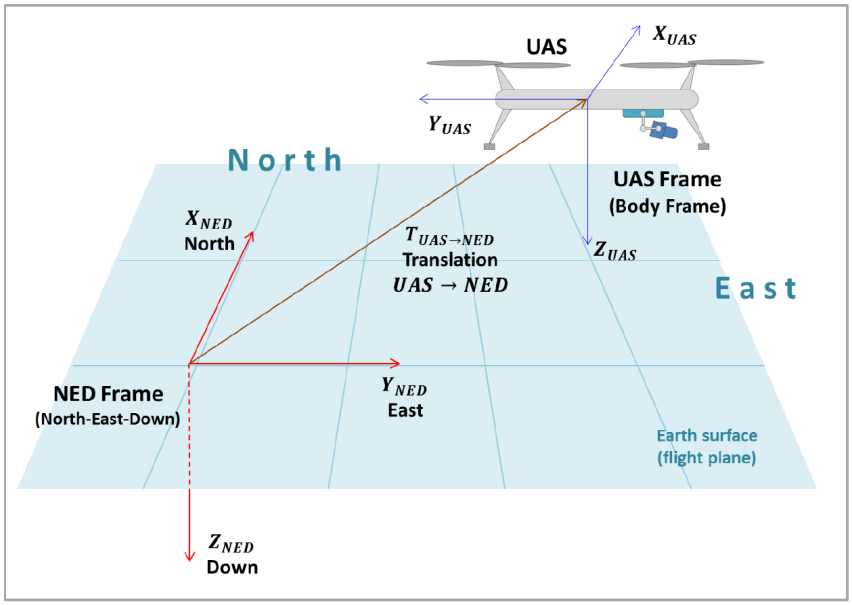

Figura 2.27: Sistemas de referência NED e do VANT [15].

Assim, para obter a transformação das coordenadas, deve-se ter a DCM entre os sistemas, $R_{VANT \rightarrow NED}$ e para isso, os ângulos de Euler (*yaw*, *pitch* e *roll*) entre os sistemas devem ser conhecidos. Dessa forma, a posição relativa do objeto de interesse no Sistema NED, P'_{NED}, é dada por:

$$P'_{NED} = R_{VANT \rightarrow NED} P'_{VANT} + T_{VANT \rightarrow NED}. \tag{2.45}$$

Substituindo a Equação 2.44 na Equação 2.45, pode-se obter de forma mais completa que:

$$P'_{NED} = R_{VANT \rightarrow NED}(R_{G \rightarrow VANT} R_{C \rightarrow G} P'_C + R_{G \rightarrow VANT} T_{C \rightarrow G} + T_{G \rightarrow VANT}) + T_{VANT \rightarrow NED} \tag{2.46}$$

$$= R_{VANT \rightarrow NED} R_{G \rightarrow VANT} R_{C \rightarrow G} P'_C + R_{VANT \rightarrow NED} R_{G \rightarrow VANT} T_{C \rightarrow G} + R_{VANT \rightarrow NED} T_{G \rightarrow VANT} + T_{VANT \rightarrow NED} \tag{2.47}$$

2.4.3.5 Sistema de Referência NED para o Sistema ENU

Para trabalhar com um sistema de coordenadas usado para georreferenciamento, ou seja, coordenadas usadas no sistema GPS, com medidas de latitude, longitude e altitude, é necessário converter as coordenadas do objeto no Sistema NED para o Sistema *East-North-UP* (ENU). Este sistema é usado como referência no sistema de coordenadas UTM, o qual está diretamente associado com o datum WGS84, usado em GPS. A Figura 2.28 mostra a relação entre os sistemas NED e ENU.

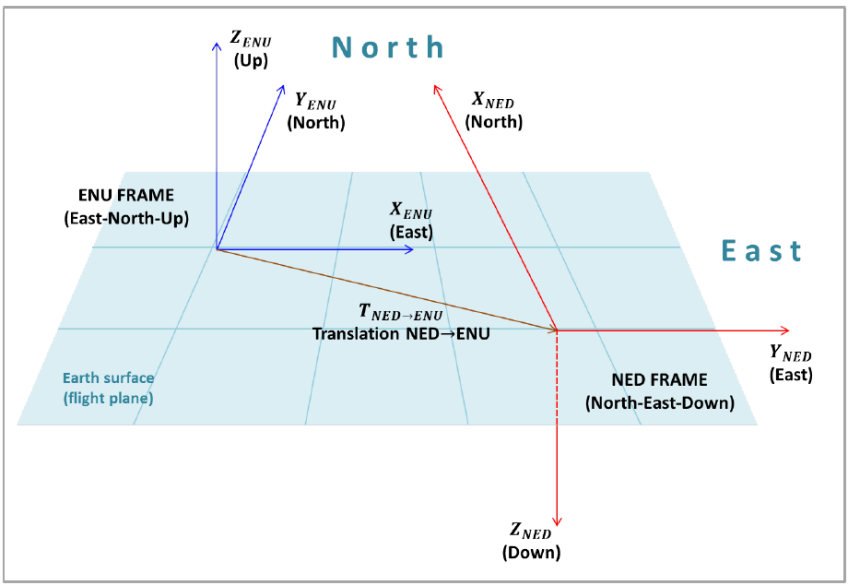

Figura 2.28: Sistemas de Referência ENU e NED [15].

A partir da Figura 2.28, a matriz de rotação do sistema NED para o ENU, $R_{NED \to ENU}$, pode ser obtida:

$$R_{NED \to ENU} = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{2.48}$$

Então, a posição relativa do objeto no sistema do ENU é dada por:

$$P'_{ENU} = R_{NED \to ENU} P'_{NED} + T_{NED \to ENU}. \tag{2.49}$$

Substituindo a Equação 2.45 na Equação 2.49, pode-se obter uma expressão geral:

$$\begin{aligned} P'_{ENU} = {} & R_{NED\rightarrow ENU}R_{VANT\rightarrow NED}R_{G\rightarrow VANT}R_{C\rightarrow G}P'_C \\ & + R_{NED\rightarrow ENU}R_{VANT\rightarrow NED}R_{G\rightarrow VANT}T_{C\rightarrow G} \\ & + R_{NED\rightarrow ENU}R_{VANT\rightarrow NED}T_{G\rightarrow VANT} \\ & + R_{NED\rightarrow ENU}T_{VANT\rightarrow NED} + T_{NED\rightarrow ENU} \end{aligned} \tag{2.50}$$

De forma simplificada,

$$P'_{ENU} = R_{C\rightarrow ENU}P'_C + T_{C\rightarrow ENU}, \tag{2.51}$$

onde a matriz $R_{C\rightarrow ENU}$ é o produto de todas as rotações (transformações) entre os sistemas de referência:

$$R_{C\rightarrow ENU} = R_{NED\rightarrow ENU}R_{VANT\rightarrow NED}R_{G\rightarrow VANT}R_{C\rightarrow G}, \tag{2.52}$$

e o vetor $T_{C\rightarrow ENU}$ é a soma de todas as translações consideradas em cada um dos passos descritos acima:

$$\begin{aligned} T_{C\rightarrow ENU} = {} & R_{NED\rightarrow ENU}R_{VANT\rightarrow NED}R_{G\rightarrow VANT}T_{C\rightarrow G} \\ & + R_{NED\rightarrow ENU}R_{VANT\rightarrow NED}T_{G\rightarrow VANT} \\ & + R_{NED\rightarrow ENU}T_{VANT\rightarrow NED} + T_{NED\rightarrow ENU} \end{aligned} \tag{2.53}$$

2.4.4 Determinação de z_C

De acordo com a Equação 2.37 a relação entre P_C com P'_C é:

$$P'_C = \frac{1}{z_C}P_C. \tag{2.54}$$

Considerando a Equação 2.51, a posição real do objeto no sistema ENU é dada por:

$$P_{ENU} = R_{C\rightarrow ENU}P_C + T_{C\rightarrow ENU}. \tag{2.55}$$

Assim, para determinar z_C, é necessário resolver um sistema formado pelas equações 2.51 e 2.55:

$$\begin{cases} P'_{ENU} = R_{C\rightarrow ENU}P'_C + T_{C\rightarrow ENU} \\ P_{ENU} = R_{C\rightarrow ENU}P_C + T_{C\rightarrow ENU} \end{cases}. \tag{2.56}$$

A segunda equação do sistema 2.56 pode ser reescrita como:

$$z_C P'_{ENU} - z_C T_{C\rightarrow ENU} = R_{C\rightarrow ENU}PC \tag{2.57}$$

A combinação das equações 2.56 e 2.57 resulta em:

$$P_{ENU} = z_C P'_{ENU} - z_C T_{C \to ENU} + T_{C \to ENU}. \tag{2.58}$$

A Equação 2.58 é uma equação matricial e tem solução na coordenada z, por causa da coordenada z_{ENU}. A coordenada z do objeto de interesse no Sistema ENU é conhecida como a altitude do objeto, considerando a origem do sistema ENU para o eixo Z_{ENU} no nível do mar. Nos casos, em que a origem do Sistema ENU esteja em outra altura de referência, a coordenada z irá corresponder a distância vertical, ou seja a altura, da origem até o nível do objeto.

Como o valor de z_{ENU} é conhecido, devido se saber qual região o VANT está sobrevoando (com auxílios dos sensores, como GPS e ultrassônico), é possível realizar os cálculos para determinar z_C como sendo:

$$z_C = \frac{z_{ENU} - z_{C \to ENU}}{z'_{ENU} - z_{C \to ENU}}, \tag{2.59}$$

onde z'_{ENU} e $z_{C \to ENU}$ são determinados com a Equação 2.51.

Assim, a posição real do objeto no Sistema ENU pode ser obtida usando a Equação 2.58:

$$P_{ENU} = z_C P'_{ENU} - z_C T_{C \to ENU} + T_{C \to ENU}.$$

Capítulo 3

Implementação da Metodologia Proposta

Após detalhar os fundamentos teóricos necessários para o entendimento da metodologia adotada para a simulação proposta, este capítulo descreve como as ferramentas e conhecimentos apresentados foram implementados a fim de ter simulações que permitam a geração de arquivos de imagens sintéticas geolocalizados (com informações de GPS gravadas nos metadados EXIF) assim como o georreferenciamento de objetos de interesse contidos nestas imagens (por meio do georreferenciamento direto de pixels) a partir de missões de detecção. Neste trabalho a aplicação se dará em detecção de pessoas, contextualizando-o no contexto de missões de busca e resgate.

Para melhor integração entre as ferramentas, a linguagem de programação utilizada foi o Python, principalmente devido ao AirSim possuir uma API com comandos para controle do drone e obtenção de imagens já implementadas nessa linguagem.

3.1 Dinâmica da Missão de Reconhecimento

A metodologia para a simulação proposta busca a ideia de permitir a geração de dados sintéticos a fim de testar algoritmos para missões de reconhecimento, inspeção, busca e resgate, entre outras aplicações. Nesse sentido, o fluxo de funcionamento da simulação segue conforme o diagrama de blocos da Figura 3.1, o qual também mostra como as ferramentas apresentadas estão integradas.

Inicialmente são definidos os parâmetros da missão no arquivo de configurações do AirSim, *settings.json*. Neste arquivo, são definidos as informações essenciais sobre a simulação, ou seja, para a missão, como:

- *SimMode:* Definição do tipo de veículo que será simulado no ambiente 3D na Unreal. Foi adotado o drone quadrirrotor;

- *OriginGeoPoint:* Posição GPS (em graus decimais) assumida pelo drone no local onde ele é instanciado inicialmente na simulação;

- *Vehicles:* Nessa seção são definidos a atitude inicial do drone (posição inicial no ambiente 3D e rotação em termos dos ângulos de Euler) e informações sobre o firmware do controlador de vôo escolhido. Neste trabalho foram usados para os resultados o *simple_flight* e *PX4*;

- *CameraDefaults:* Configurações da câmera (e gimbal associado) acoplada ao drone: Tipo de imagem (normal - 0, segmentada - 5, infravermelho - 7, etc), resolução da câmera, FOV, posição e estabilização do gimbal, quantos tipos diferentes de visualização associados aos tipos de imagens (*SubWindows*), etc.

As figuras 4.2 e 4.4 (no Capítulo 4) mostram dois exemplos de como as esses parâmetros foram colocados no arquivo de configuração do AirSim, tanto para o uso com o PX4 como para o controlador de vôo embutido no AirSim *simple_flight*.

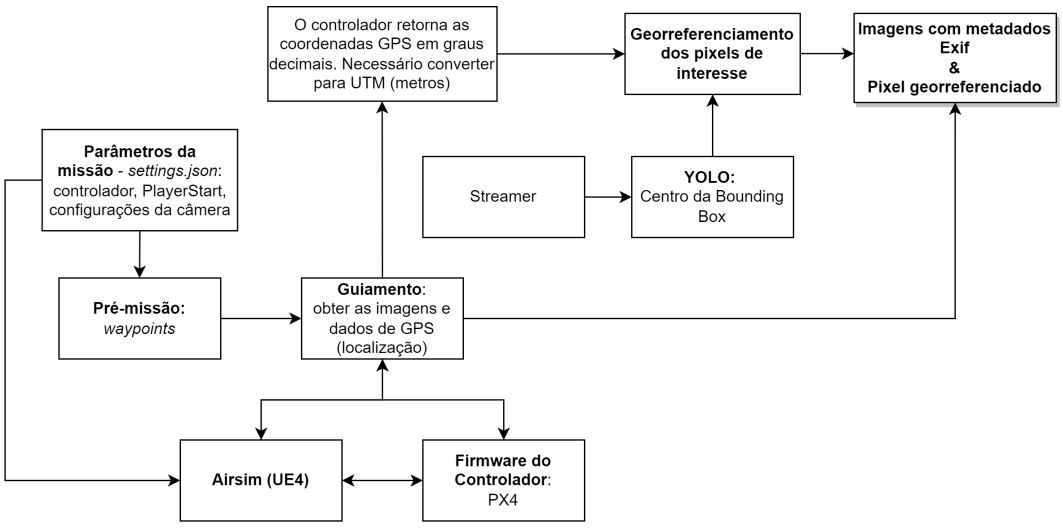

Figura 3.1: Diagrama de blocos da simulação implementada no modo Software-In-The-Loop.

Seguindo no fluxo de simulação da Figura 3.1, tem-se o bloco de Pré-missão, onde são planejados os pontos do cenário 3D que o drone irá percorrer. Os métodos na API do AirSim explorados na implementação deste trabalho são:

- *moveToPositionAsync()*: Tem como argumento as coordenadas no sistema de coordenadas do cenário virtual, ou seja, onde a referência são as próprias medidas da Unreal.

- *moveToGPSAsync()* e *moveToPositionAsyncGeo()*: Tem como argumentos coordenadas GPS (em graus decimais).

Visto isso, é planejado o caminho que o drone irá percorrer e este pode ser passado como *waypoints* ao código de Guiamento por meio de arquivo de texto ou definidos diretamente no código-fonte. O bloco de Guiamento é responsável por passar os comandos de movimentação, obter as imagens e dados de localização GPS, de acordo com os *waypoints* escolhidos. Essas informações são requisitadas e os comandos passados para o AirSim e controlador de vôo.

As subseções 3.1.1 a 3.1.4 explanam a implementação dos demais blocos (*Streamer*, YOLO e Georreferenciamento) e como eles interagem no fluxo de simulação para gerar imagens geolocalizadas e o georreferenciamento direto do objeto, ou seja, do pixel de interesse da imagem que contém o objeto detectado.

3.1.1 Código para transmissão de imagens obtidas pelo AirSim - *Streamer*

Para simular uma transmissão de vídeo da missão de reconhecimento que está sendo feita na Unreal, foi implementado um código Python baseado na *issue* em [54]. Com esta aplicação, é possível transmitir as imagens obtidas pelo drone simulado por meio da implementação de um servidor local usando o micro-framework *Flask*. O Flask permite a criação rápida de aplicações web e é considerado *micro* porque permite a criação simplificada de um sistema de rotas para as páginas criadas [55].

Nesta implementação, é criada uma página HyperText Markup Language (HTML), acessível apenas localmente, para onde estão sendo transmitidas as imagens obtidas com a API do AirSim usando o método *simGetImage()*, que retorna os bytes da imagem em formato PNG (*Portable Network Graphic*). Para geração dos *frames*, a imagem é convertida em um conjunto de dados do tipo *array* com método do *NumPy asarray()* [56], em associação ao método *bytearray()*. Com isso, é possível manipular os dados com a biblioteca *OpenCV* (*Open Source Computer Vision Library*) para gerar a imagem a partir do array no formato desejado, utilizando o método *imencode()*. Neste trabalho usou-se o formato JPEG. Além disso, é possível alterar a resolução das imagens obtidas com o método *resize()*. Nesta aplicação, é possível ajustar a resolução de transmissão para 5 opções pré-definidas (do 240p ao 1080p), dentre as resolu-

Tabela 3.1: Resoluções disponíveis na plataforma YouTube [58]

Resolução	**Nome**	**Qualidade**
3840 x 2160	2160p	4K
2560 x 1440	1440p	2K
1920 x 1080	1080p	Resolução máxima Full HD
1280 x 720	720p	Resolução mínima HD
854 x 480	480p	Definição padrão
640 x 360	360p	Resolução normal de websites
426 x 240	240p	Tamanho mínimo para vídeos no YouTube

ções usuais encontradas em aplicações de streaming e na plataforma YouTube, como mostra a Tabela 3.1 [57] [58]. Mas também é possível ajustar para outra resolução desejada usando o mesmo método, *resize()*.

3.1.2 YOLO

O modelo da família YOLO escolhido para ser utilizado nesse trabalho é a YOLOv5 (*You Only Look Once version 5*), com o código fonte localizado em [22]. Esta versão foi desenvolvida com PyTorch, que é uma biblioteca (da linguagem Python) de tensores otimizada para aprendizado profundo, incluindo visão computacional [59].

Nesta versão da YOLO, é possível passar no argumento do comando para execução do código de detecção o caminho para o diretório onde há as imagens para que a rede detecte objetos ou até mesmo a URL (*Uniform Resource Locator*) de um vídeo sendo transmitido em tempo real, ou que esteja hospedado em algum site. Valendo-se disso, foi implementado o código do *Streamer* para transmitir as imagens que estão sendo obtidas pelo AirSim do cenário 3D na Unreal. Também é possível passar o caminho até um arquivo de imagem específico para que rede realize a inferência na imagem em busca do objeto de interesse.

Com as imagens obtidas, modificou-se o código fonte da classe *Annotator* da YOLOv5, onde são definidas as configurações da *bounding box* que irá delimitar o objeto detectado. Dessa forma, foi configurado para que também fosse plotado o pixel central da *bounding box*, onde,

dado que L é a largura e H a altura da caixa delimitadora:

$$x_{centro} = L/2 \tag{3.1}$$

$$y_{centro} = H/2 \tag{3.2}$$

Assim, esse conjunto de coordenadas é passado para o algoritmo de georreferenciamento como sendo os elementos do vetor P_i (no Sistema de Referência da Imagem), onde, seguindo a notação da Equação 2.33, $u = x_{centro}$ e $v = y_{centro}$, dados em pixels.

3.1.3 Algoritmo de Georreferenciamento

Após o pixel central da *bounding box* ao redor do objeto detectado ser fornecido pelo algoritmo da YOLO, esse par de dados é usado como entrada para o algoritmo de georreferenciamento. Além disso, é passada a localização GPS em graus decimais (latitude e longitude), a altura do drone (VANT) em relação ao objeto detectado [no momento da detecção] e z_{ENU}, que é a altitude (ou altura) do objeto de interesse.

Importante ressaltar que é mais relevante para a precisão do algoritmo saber a altura em que o drone estava do objeto de interesse (no momento em que ele foi detectado) do que o próprio valor de z_{ENU}. Então, se na abordagem for considerado que z_{ENU} é a altitude do objeto de interesse, pode-se considerar também a altitude do drone no momento em que a detecção foi realizada. Porém, pode-se assumir também que z_{ENU} é sempre igual a 0 (ou seja, o objeto de interesse está no solo ou no nível do mar) e preocupar-se apenas com a altura do drone em relação ao objeto detectado. Na implementação de um sistema real, essa distância pode ser retornada manipulando os dados de um sensor ultrassônico, mas esta abordagem não será considerada na implementação deste trabalho.

A posição GPS e a altura do drone são retornadas pela API do AirSim com os métodos *getGpsData()* e *simGetVehiclePose()*, respectivamente. O segundo pode ser usado para conhecer a posição do drone no sistema NED, tendo as coordenadas nos eixos X, Y e Z. Para que isso seja coerente, o drone precisar ser instanciado na posição [0,0,0] - a qual é definida com o parâmetro *PlayerStart*. Dessa forma, não é preciso fazer a conversão de coordenadas do ambiente virtual para o sistema NED, pois as origens entre o sistema de referência do VANT e o sistema NED serão iguais. Para ter mais precisão no georreferenciamento direto, também é possível passar a atitude do drone no momento da detecção por meio do método *getMultirotorState()*, conhecendo assim os ângulos de Euler para construção da matriz $R_{VANT \rightarrow NED}$.

Em seguida, a latitude e longitude, dadas em graus decimais, são convertidas para o sistema UTM por meio do módulo *pyproj* [35], no qual é preciso inserir a zona UTM onde o drone está localizado e o SGR adotado na missão. Neste trabalho irá se usar o datum WGS84 para generalização do uso da metodologia proposta em relação a localidades no planeta Terra (já que este este é usado como sistema de referência para o GPS). Essas coordenas serão utilizadas (juntamente com a altitude do drone no momento da captura da imagem) como o deslocamento entre as origens do Sistema NED e ENU, $T_{NED \rightarrow ENU}$ (Equação 2.49). Para isso, esse conjunto de coordenadas precisa ser dado em metros, o que justifica a conversão para o sistema UTM.

Então, seguindo os passos descritos na Subseção 2.4.3, o primeiro passo é converter o vetor P_i, referente ao conjunto de coordenadas no Sistema da Imagem, para o Sistema da Câmera. Para isso, a matrix com os parâmetros intrínsecos da câmera, K, é definida com auxílio de informações definidas previamente no arquivo de configuração do AirSim (*settings.json*) e referentes um modelo de câmera escolhido (basicamente, escolher valores para os tamanhos da imagem e do sensor para compor os elementos da matriz na Equação 2.34). Usando a biblioteca do Python *NumPy* é possível facilmente realizar a inversão de matrizes a fim de implementar a Equação 2.37 e obter o vetor P'_C.

Para obter as matrizes DCM, é usada a biblioteca *NavPy* [60], onde com o método *navpy.angle2dcm()*, é possível inserir os três ângulos de Euler (que neste trabalho são usados em graus) e ter na saída da função a Matriz dos Cossenos Diretores correspondente. Na documentação da biblioteca é informado que a matriz DCM transforma um vetor do nível local de coordenadas (ou seja, do Sistema de referência NED) para o sistema de referência do corpo do VANT (em inglês, *body frame*). Mas como no algoritmo de georreferenciamento é preciso converter as coordenadas do sistema do corpo do VANT (ou seja, o Sistema do VANT) para o sistema NED, a matriz usada é a transposta da matriz DCM retornada pelo método *navpy.angle2dcm()*.

Após realizar as transformações até o sistema de referência ENU, pode-se obter z_C, como descrito na Subseção 2.4.4 e enfim obter a coordenadas de latitude e longitude do pixel de interesse. O pseudocódigo 1 descreve os passos implementados no código-fonte.

3.1.4 Inserindo metadados EXIF nas imagens

Para inserir os dados de GPS nas imagens é utilizada a biblioteca Python *Piexif*. Com ela é possível manipular os metadados EXIF de arquivos de acordo com o objetivo de interesse. No

Algorithm 1 Georreferenciamento

1: Inserir P_i ▷ composto pelos pixels detectados pela YOLOv5
2: Inserir a altura e localização GPS do drone (em graus decimais)
3: Inserir a altitude (ou altura) do objeto de interesse
4: Converter graus decimais para UTM
5: Definir: f, FOV, a resolução da câmera e dimensões do sensor da câmera ▷ associados ao *settings.json* e ao modelo de câmera escolhido
6: Definir K
7: $P'_C \leftarrow K^{-1}.P_i$ ▷ Imagem para Câmera
8: Definir $R_{C\rightarrow G}$
9: Inserir $T_{C\rightarrow G}$
10: $P'_G \leftarrow R_{C\rightarrow G}P'_C + T_{C\rightarrow G}$ ▷ Câmera para Gimbal
11: Inserir os ângulos de rotação (Euler) do Gimbal: ψ_G, θ_G e ϕ_G
12: $R_{G\rightarrow VANT} \leftarrow [DCM(\psi_G, \theta_G, \phi_G)]^T$
13: Inserir $T_{G\rightarrow VANT}$
14: $P'_{VANT} \leftarrow R_{G\rightarrow VANT}P'_G + T_{G\rightarrow VANT}$ ▷ Gimbal para o VANT
15: Inserir os ângulos de rotação (Euler) do VANT: ψ_{VANT}, θ_{VANT} e ϕ_{VANT}
16: $R_{VANT\rightarrow NED} \leftarrow [DCM(\psi_{VANT}, \theta_{VANT}, \phi_{VANT})]^T$
17: Inserir $T_{VANT\rightarrow NED}$
18: $P'_{NED} \leftarrow R_{VANT\rightarrow NED}P'_{VANT} + T_{VANT\rightarrow NED}$ ▷ VANT para NED
19: Inserir $R_{NED\rightarrow ENU}$
20: Inserir $T_{NED\rightarrow ENU}$
21: $P'_{ENU} \leftarrow R_{NED\rightarrow ENU}P'_{NED} + T_{NED\rightarrow ENU}$ ▷ NED para ENU
22: $T_{C\rightarrow ENU} \leftarrow T_{NED\rightarrow ENU} + R_{NED\rightarrow ENU}T_{VANT\rightarrow NED} + R_{NED\rightarrow ENU}R_{VANT\rightarrow NED}T_{G\rightarrow VANT} + R_{NED\rightarrow ENU}R_{VANT\rightarrow NED}R_{G\rightarrow VANT}T_{C\rightarrow G}$
23: $z_{C\rightarrow ENU} \leftarrow T_{C\rightarrow ENU}[2]$
24: $z'_{ENU} \leftarrow P'_{ENU}[2]$
25: Inserir z_{ENU} ▷ Altura do objeto de interesse
26: $z_C = \frac{z_{ENU} - z_{C\rightarrow ENU}}{z'_{ENU} - z_{C\rightarrow ENU}}$
27: $P_{ENU} \leftarrow z_C P'ENU - z_C T_{C\rightarrow ENU} + T_{C\rightarrow ENU}$ (posição georreferenciada em UTM)
28: Converter UTM para graus decimais
29: Saída: Pixel georreferenciado em coordenadas GPS (latitude e longitude)

caso desta aplicação, o foco é inserir as informações de latitude, longitude e altitude do drone no momento em que a câmera associada ao mesmo obteve cada imagem em questão.

A informação de localização é requisitada pelo script de Guiamento para o controlador de vôo, o qual fornece os dados de latitude e longitude em graus decimais. Para inserir esses dados no formato EXIF nas imagens, é necessário converter o formato para graus, minutos e segundos. Para isso, foi desenvolvido um algoritmo que recebe as informações de GPS do simulador de drones (AirSim) juntamente com o nome do arquivo de imagem em que se deseja inserir os dados de localização. Também são definidos os caminhos dos diretórios de onde se encontram as imagens sem metadados EXIF e onde se deseja salvá-las após o processo de inserção dessas informações. O processo para converter graus decimais para graus, minutos e segundos é descrito no pseudocódigo 2, onde irá ser considerado a latitude como exemplo, mas os mesmos passos são válidos para a longitude. Exceto que os marcadores para valores positivos e negativos são *N* (North) e *S* (South), respectivamente, para latitude e *E* (East) e *W* (West), respectivamente, para longitude.

Algorithm 2 Transformação de graus decimais para graus, minutos e segundos

1: Inserir latitude
2: **if** $latitde > 0$ **then**
3: Marcador *N*
4: **else**
5: Marcador *S*
6: $latitute \leftarrow abs(latitude)$ ▷ Tirar o módulo do valor da coordenada
7: $Graus \leftarrow int(latitude)$ ▷ pega-se apenas a parte inteira do valor
8: $Resto\_minutos \leftarrow (latitude\%1) \times 60$ ▷ pega-se apenas a parte decimal da latitude e multiplica-se por 60
9: $Minutos \leftarrow int(Resto\_minutos)$
10: $Resto\_segundos \leftarrow (Resto\_minutos\%1) \times 60$
11: $Segundos = int(Resto\_segundos)$

Após calcular as novas informações de latitude e longitude, estas são unidas com a altitude em um dicionário de informações o qual corresponde justamente ao campo onde são encontradas as informações de GPS quando observa-se as propriedades de um arquivo de imagem, como mostra a Figura 3.2. As informações são transformadas em bytes com o método *piexif.dump()* para depois assim poderem ser escritas no cabeçalho do arquivo de imagem.

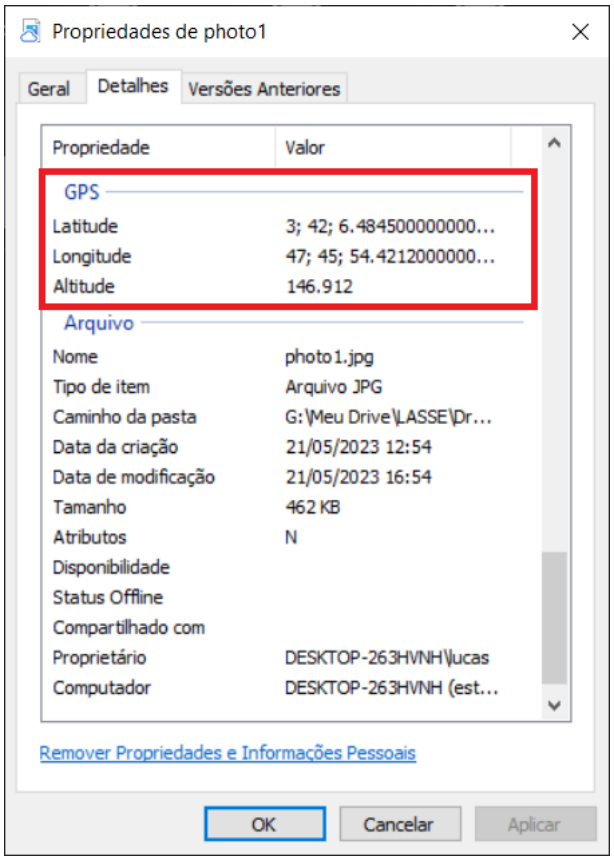

Figura 3.2: Dados de GPS gravados nos metadados de um arquivo de imagem.

Capítulo 4

Resultados

Com o objetivo de gerar resultados para testar a dinâmica do script de guiamento, obtendo imagens com metadados EXIF, juntamente com o teste do algoritmo de georreferenciamento do pixel central da bounding box que envolve a pessoa detectada por meio da YOLOv5, foram planejados cenários de teste na Unreal Engine, em conjunto com as configurações do AirSim. A integração das ferramentas descritas neste trabalho para a geração desses cenários de teste pode ser visualizada na Figura 4.1, a qual oferece uma visão mais específica e detalhada do que foi implementado durante a pesquisa para o desenvolvimento desta metodologia. Para melhor entendimento em relação a como as grandezas são retornadas pelas ferramentas utilizadas, visando facilitar até mesmo a reprodução deste trabalho, o separador decimal adotado nas seções a seguir será o ponto.

4.1 Resultados com Simulações no Modo SITL

De acordo com a Figura 4.1, a definição dos waypoints e monitoramento da missão pode ser dada pelo *QGroundControl*, no modo de simulação *Software-In-The-Loop*. O modo de SITL foi muito relevante no desenvolvimento desta pesquisa pois a implementação do mesmo auxiliou no entendimento da dinâmica de funcionamento de um sistema UAS (*Unmanned Aircraft System*). Com isso, pôde-se trabalhar com uma Estação de Controle Terrestre e firmware do controlador de vôo usados em sistemas reais, entendendo os princípios de integração por meio do link de comunicação entre os blocos (que acontece com o protocolo MAVLINK). Além disso, métodos do AirSim para lidar com coordenadas GPS, como o *moveToGPSAsync()*, funcionavam inicialmente apenas no modo de simulação SITL pois, necessitavam ter um firmware

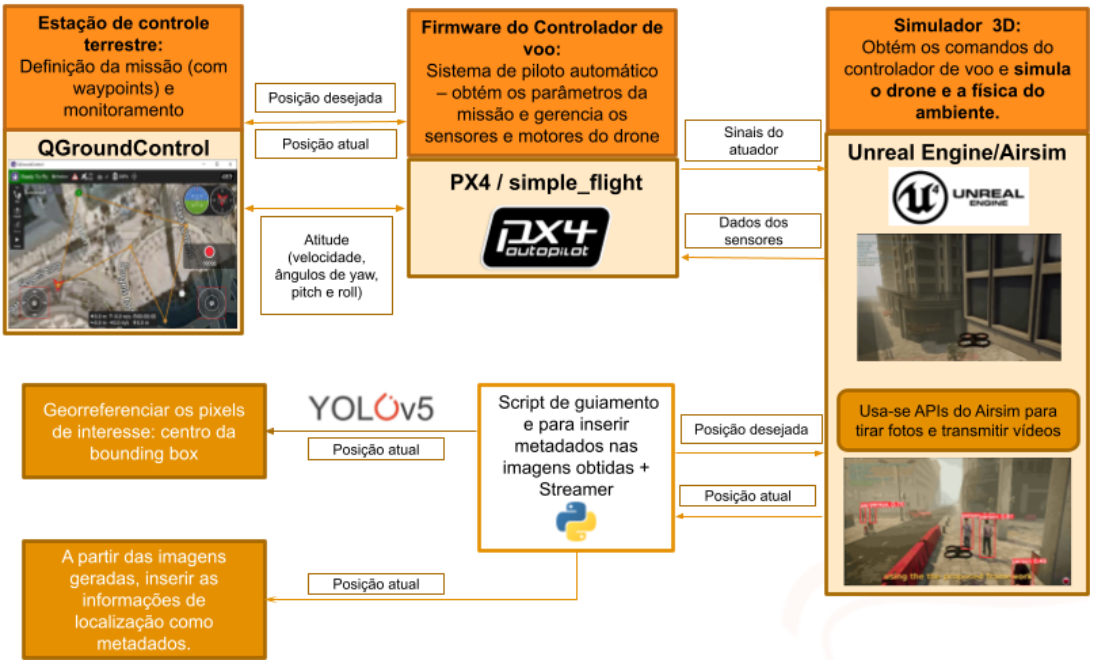

Figura 4.1: Diagrama de blocos da simulação implementada no modo Software-In-The-Loop e com o script de guiamento.

de controlador de vôo como o *PX4* ou *ArduPilot*.

Porém, com atualizações da API do AirSim, esses métodos ficaram mais ajustados para funcionar com o controlador de vôo *simple_flight*. Então, para geração de resultados de forma mais otimizada, a fim de facilitar a integração dos algoritmos e resultados desenvolvidos durante este trabalho com as demais aplicações que estão sendo desenvolvidas no Núcleo de Pesquisa e Desenvolvimento em Telecomunicações, Automação e Eletrônica (LASSE) - envolvendo simulação do canal de comunicação e outros métodos de Inteligência Artificial - não investiu-se tanto no modo de simulação SITL para compilar os resultados apresentados neste trabalho. Entretanto, resultados envolvendo o modo SITL podem ser encontrados em [61] e [62]. As configurações do AirSim usadas para a integração das ferramentas utilizadas podem ser vistos na Figura 4.2. As coordenadas GPS referentes ao *OriginGeoPoint* de [61] correspondem a um ambiente rural localizado no estado do Pará. Já no vídeo [62], as coordenadas correspondem a um ambiente urbano, na Suécia, com as coordenadas do *OriginGeoPoint* sendo: $Latitude : 59.3325446$ e $Longitude : 18.0640958$. Os cenários estão com essas coordenadas apenas para se ter uma referência com relação ao GPS, mas o ambiente 3D mostrado não é uma réplica dessas localidades.

```
{
    "SettingsVersion": 1.2,
    "SimMode": "Multirotor",
    "ClockType": "SteppableClock",
    "OriginGeopoint": {
        "Latitude": -3.701955,
        "Longitude": -47.764951,
        "Altitude": 132
    },
    "Vehicles": {
        "PX4": {
            "VehicleType": "PX4Multirotor",
            "UseSerial": false,
            "LockStep": true,
            "UseUdp": true,
            "UseTcp": true,
            "TcpPort": 4560,
            "UdpPort": 14560,
            "ControlPortLocal": 14540,
            "ControlPortRemote": 14580,
            "Sensors":{
                "Barometer":{
                    "SensorType": 1,
                    "Enabled": true,
                    "PressureFactorSigma": 0.0001825
                }
            },
            "Parameters": {
                "NAV_RCL_ACT": 0,
                "NAV_DLL_ACT": 0,
                "COM_OBL_ACT": 1,
                "LPE_LAT": -3.701955,
                "LPE_LON": -47.764951
            }
        }
    },
    "CameraDefaults": {
      "CaptureSettings": [
          {
              "ImageType": 0,
              "Width": 854,
              "Height": 480,
              "FOV_Degrees": 90,
              "MotionBlurAmount": 0
          },
          {
              "ImageType": 5,
              "Width": 854,
              "Height": 480,
              "FOV_Degrees": 90,
              "MotionBlurAmount": 0
          }
      ],
      "Gimbal": {
        "Stabilization": 1,
        "Pitch": -90, "Roll": 0, "Yaw": 0
      },
      "X": 0, "Y": 0, "Z": 0
    },
    "SubWindows": [
      {
          "WindowID": 0,
          "CameraName": "3",
          "ImageType": 0,
          "VehicleName": "PX4",
          "Visible": true,
          "External": false
      },
      {
          "WindowID": 1,
          "CameraName": "3",
          "ImageType": 5,
          "VehicleName": "PX4",
          "Visible": true,
          "External": false
      }
    ]
}
```


Figura 4.2: Configurações do AirSim para o modo de simulação SITL.

4.2 Resultados com o Controlador de Vôo *simple_flight*

Os demais cenários utilizados para teste do funcionamento e integração entre os algoritmos desenvolvidos estão divididos entre os diferentes ambientes 3D simulados na Unreal representando diferentes contextos e aplicações nos quais a metodologia pode ser útil para geração de dados sintéticos e testes de redes neurais em missões de reconhecimento, monitoramento, busca e resgate. Dessa forma, para a geração dos próximos resultados, as ferramentas foram integradas tal qual mostra a Figura 4.3, onde o controlador de vôo utilizado é o *simple_flight*. Nesta configuração, a dinâmica da missão ocorre conforme mostra o diagrama da Figura 3.1 e o arquivo de configuração do AirSim pode ser visto na Figura 4.4.

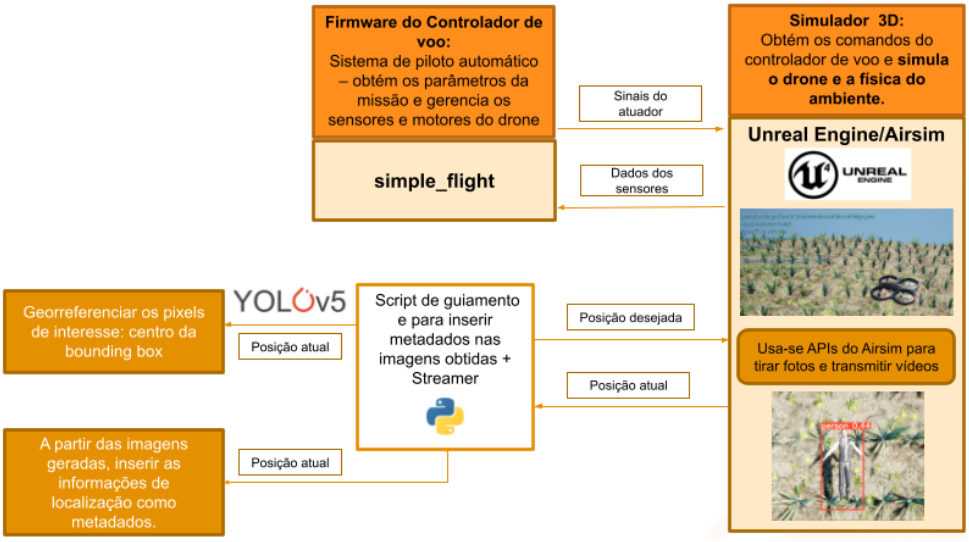

Figura 4.3: Diagrama de blocos da simulação implementada com o controlador de vôo *simple_flight*.

Dentro do arquivo *settings.json* várias configurações importantes são definidas, as quais são relevantes para o algoritmo de georreferenciamento. Essas configurações incluem a resolução da imagem, FOV, ângulos de Euler com relação ao Sistema de Referência do Drone e do Gimbal, a posição do Gimbal com relação ao drone e a resolução da imagem. A Tabela 4.1 apresenta a relação desses parâmetros entre a configuração do AirSim e o algoritmo de georreferenciamento.

O vetor de translação entre o Sistema do drone para o Sistema NED, $T_{drone \rightarrow NED}$, tem relação direta com onde o drone é instanciado no ambiente 3D da Unreal. Esse parâmetro também pode ser definido na Unreal e é conhecido como *PlayerStart*, que é justamente onde o elemento principal da simulação será instanciado no ambiente 3D. Fazendo o *PlayerStart* igual

```
{
    "SettingsVersion": 1.2,
    "SimMode": "Multirotor",
    "OriginGeopoint": {
      "Latitude": -3.701955,
      "Longitude": -47.764951,
      "Altitude": 132
    },
    "Vehicles": {
        "D1": {
            "VehicleType": "SimpleFlight",
            "DefaultVehicleState": "Disarmed",
            "X": 0,
            "Y": 0,
            "Z": 0,
            "Pitch": 0,
            "Roll": 0,
            "Yaw": 0
        }
    },
    "CameraDefaults": {
        "CaptureSettings": [
            {
                "ImageType": 0,
                "Width": 2048,
                "Height": 1024,
                "FOV_Degrees": 90,
                "MotionBlurAmount": 0
            },
            {
                "ImageType": 5,
                "Width": 2048,
                "Height": 1024,
                "FOV_Degrees": 90,
                "MotionBlurAmount": 0
            }
        ],
        "Gimbal": {
          "Stabilization": 1,
          "Pitch": -90, "Roll": 0, "Yaw": 0
        },
        "X": 0, "Y": 0, "Z": 0
    },
    "SubWindows": [
        {
            "WindowID": 0,
            "CameraName": "3",
            "ImageType": 0,
            "VehicleName": "D1",
            "Visible": true,
            "External": false
        },
        {
            "WindowID": 1,
            "CameraName": "3",
            "ImageType": 5,
            "VehicleName": "D1",
            "Visible": true,
            "External": false
        }
    ]
}
```


Figura 4.4: Configurações do AirSim usando o *simple_flight*.

Tabela 4.1: Parâmetros de configuração do AirSim relevantes para o georreferenciamento

Nome do Parâmetro	**Parâmetro (settings.json)**	**Parâmetro (georreferenciamento)**	**Valor**
Posição do drone	Bloco "Vehicles": X, Y, Z	$T_{drone \rightarrow NED}$	[0, 0, 0]
Ângulos de Euler do drone	Bloco "Vehicles": Pitch, Roll e Yaw	$[\psi_{drone}, \theta_{drone}, \phi_{drone}]$	[0,0,0]
Resolução da Câmera	Bloco "CameraDefaults": Width e Height	Largura_da_Imagem Altura_da_Imagem	[2048, 1024]
Field of View (FOV)	FOV_Degrees	FOV	90
Ângulos de Euler do gimbal	Bloco "Gimbal": Pitch, Roll e Yaw	$[\psi_G, \theta_G, \phi_G]$	[-90, 0, 0]
Posição do gimbal em relação ao drone	Bloco "Gimbal": X, Y, Z	$T_{G \rightarrow drone}$	[0,0,0]

a coordenada [0,0,0], a conversão entre as coordenadas fica mais direta. Já os ângulos de Euler do drone indicam a atitude inicial do mesmo.

Com relação aos ângulos de Euler do gimbal, configurou-se para que ele posicione a câmera virada para baixo, com um ângulo de 90° em relação ao drone e com estabilização do gimbal ativada (bloco "Gimbal", parâmetro "Stabilization"no arquivo de configurações do AirSim). Dessa forma, é possível configurar a inclinação desejada para a câmera. Quanto à posição da câmera em relação ao gimbal e do gimbal em relação ao drone, assume-se que os sistemas de referência desses 3 elementos têm a mesma origem. Por isso, é definido que $T_{G \rightarrow drone} = T_{C \rightarrow G} = [0, 0, 0]$.

Para os resultados propostos, escolheu-se que as imagens do AirSim seriam transmitidas para a YOLOv5 com na resolução HD (1280 x 720, de acordo com a Tabela 3.1). Por isso, mesmo que no arquivo de configuração do AirSim as imagens estejam sendo obtidas em uma resolução de 2048 x 1024, é possível redimensionar a resolução para o valor desejado como descrito na Subseção 3.1.1. Então, a resolução da imagem que será considerado pelo algoritmo de georreferenciamento não será a que foi definido na configuração inicial do AirSim, mas sim a escolhida no código do *Streamer*.

Como a campo de visão da câmera (FOV) já é definido no arquivo *settings.json*, pode-se manipular a Equação 2.29 e encontrar a largura do sensor, como:

$$Largura\_do\_Sensor = \text{tg}(FOV/2) \times 2f. \tag{4.1}$$

Com isso, é possível definir a matriz com os parâmetros intrínsecos da câmera de acordo com a Equação 2.34, onde, considerando $Altura\_do\_Sensor = 8mm$ e $f = 12mm$,

$$K = \begin{bmatrix} 640 & 0 & 640 \\ 0 & 1080 & 360 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.2}$$

O computador utilizado para realizar as simulações possui uma GPU NVIDIA RTX 3050 versão laptop (com 4 GB de memória dedicada), um processador Ryzen 5 5600H, 16 GB memória RAM DDR4, 512 GB de SSD e executando o sistema operacional Windows 10.

4.2.1 Resultados em um ambiente 3D fictício

O cenário com ambiente 3D fictício representa a aplicação da simulação proposta em ambientes virtuais genéricos, ou seja, que não se propõem a ser réplicas de um localidade no mundo real. Desta forma, é possível também testar a metodologia proposta em ambientes fictícios, usados no desenvolvimento de jogos ou com elementos 3D customizados de acordo com a necessidade de uma determinada aplicação. Por exemplo, o cenário proposto representa um ambiente rural e ilustra como a simulação pode ser usada em contextos de monitoramento para situações de busca e resgaste de pessoas. Mas também, tendo um algoritmo para detecção de ervas daninhas [20], pode-se, com os pixels referentes a localização das daninhas (ou outro elemento de interesse na plantação), usar o algoritmo de georreferenciamento direto e assim ter a localização em GPS de uma localidade onde se tenha maior densidade dessas ervas para testar um sistema automatizado de pulverização de herbicidas com drones. A Figura 4.5 mostra o ambiente 3D na UE4, onde os testes foram realizados. Neste ambiente, uma pessoa se encontra paralela ao chão, simulando uma pessoa deitada, a fim de ilustrar um contexto de busca e viabilização do resgate, por meio da detecção e geolocalização de onde essa pessoa se encontra.

Assim, segundo o fluxo de simulação descrito na Figura 3.1, foram definidos os parâmetros da missão conforme mostra a Figura 4.4. A localização se refere a uma fazenda no interior do Pará, na zona UTM 23, onde as coordenadas UTM de latitude e longitude do *OriginGeoPoint* são: *9590338.235376593 m S* e *192848.83382590482 m E*. A missão consiste em percorrer a

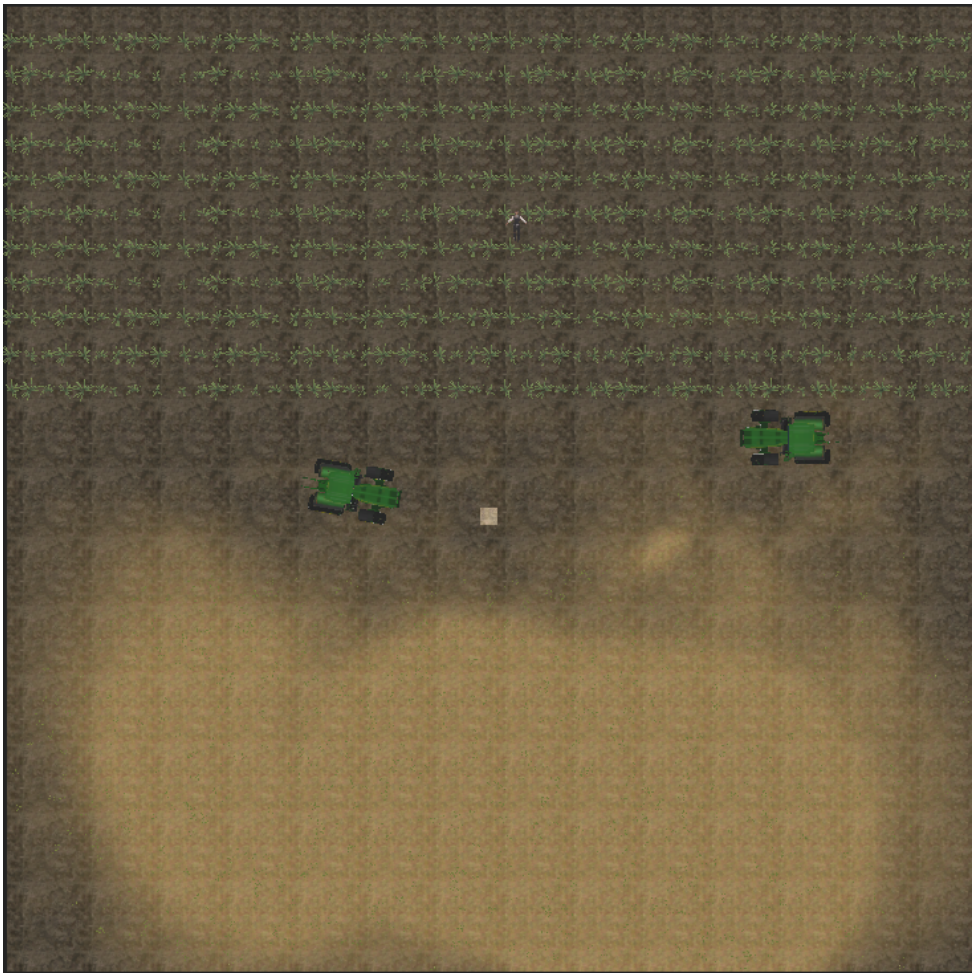

Figura 4.5: Ambiente 3D na Unreal representando um cenário rural.

plantação, tirando 15 fotos, das quais a cada a foto obtida, os dados de GPS (latitude, longitude e altitude) são gravados nos metadados EXIF da imagem. Enquanto isso, o código de *Streamer* está transmitindo as imagens para a YOLOv5 que é executada com o comando *python detect.py –weights yolov5s.pt –class 0 –source http://localhost:5000/video_feed*, onde:

- *weights yolov5s.pt*: conjunto de pesos que serão utilizados no processo de inferência (dos quais pode-se utilizar os pesos pré-treinados da YOLOv5);
- *class 0*: indica a classe de interesse dentro do dataset COCO que se deseja detectar. No caso "0" indica pessoas;
- *source http://localhost:5000/video_feed*: é fonte da onde estão sendo transmitidas as imagens utilizadas na tarefa de detecção da rede. O endereço *http://localhost:5000/video_feed* indica que o vídeo está sendo transmitido em um servidor local criado com a biblioteca *Flask* e que pode ser acessado na porta 5000, utilizando o endereço http indicado.

4.2.1.1 Validação dos resultados referentes às imagens geolocalizadas

Com as 15 fotos geolocalizadas, foi possível montar uma ortofoto no software *WebODM*, conforme mostra a Figura 4.6. O *WebODM* é uma interface gráfica web do software *OpenDroneMap* (ODM), o qual é uma solução de código aberto para processamento de imagens aéreas e geração de ortofotos, nuvens de pontos, modelos de elevação e modelos 3D [63]. Com esta ferramenta é possível obter um relatório de qualidade onde pode-se observar o nível de reconstrução do ambiente fotografado a partir das imagens obtidas.

A Tabela 4.2 destaca algumas informações contidas no relatório de qualidade do WebODM. Dentre elas estão o número e a porcentagem das imagens que foram reconstruídas para montar a ortofoto da Figura 4.6. O ODM procura detectar características distintas nas imagens, como cantos ou arestas, e associar essas características a pontos correspondentes nas imagens de entrada. Refinando esse processo por meio de triangulação entre os pontos chaves (pontos em comum entre as imagens) e as posições conhecidas da câmera, é formada a nuvem de pontos reconstruídos (esparsos), a qual fornece uma representação tridimensional aproximada da geometria da cena, capturando as principais características estruturais . Para formação da nuvem de pontos densa, o software preenche espaços vazios por meio de interpolação e informações de imagens adjacentes, incluindo detalhes texturais [64] [65]. Também encontra-se na tabela a informação referente ao *Ground Sampling Distance* (GSD), que é definido como a distância

entre os centros de dois pixels adjacentes medidos no solo. Essa métrica está relacionada à distância focal da câmera, à resolução do sensor da câmera e à distância da câmera ao objeto. Assim, quanto maior o valor do GSD da imagem, menor a resolução espacial da imagem e menos detalhes visíveis [66] [67].

Outro resultado interessante gerado pelo WebODM para verificar a qualidade das informações contidas nas imagens geolocalizadas fornecidas ao software, é mostrado na Figura 4.7. Nela, é apresentada uma estimativa das dimensões do ambiente reconstruído a partir das imagens processadas pelo WebODM. A linha azul representa o trajeto percorrido pelo drone, com os pontos azuis correspondendo às coordenadas GPS contidas nas imagens (*GPS EXIF Data*), enquanto a posição reconstruída pelo software é destacada em vermelho (*Reconstructed GPS position*). Observa-se que, em alguns pontos, houve um erro entre as informações nas imagens e a reconstrução das informações, o que pode ser mensurado com o parâmetro referente aos erros de GPS da Tabela 4.2. Além disso, os vários pontos em tonalidade amarela (*Reconstructed Points*) na Figura 4.7 formam a nuvem de pontos reconstruídos [65].

Embora o software tenha errado ao dimensionar o eixo X, mostrando apenas a distância a partir do centro do drone até 3 metros à frente do que foi capturado pela câmera, as dimensões no eixo Y ficaram coerentes com a linha de *waypoints* configurada para o VANT percorrer. O comprimento de $53\ m$ é consistente, visto que, no planejamento da missão, escolheu-se o ponto $-25\ m$ (em relação ao eixo Y, que corresponde ao eixo horizontal) como ponto inicial para que o drone começasse a capturar as imagens. E o ponto final do percurso é correspondente a $25\ m$ (também ao longo do eixo Y).

Porém, como a câmera embutida no drone captura uma visão lateral a partir do ponto central (o próprio drone), isso resultou em alguns metros adicionais além dos $50\ m$ definidos para o percurso, devido ao campo de visão da câmera (FOV). Como resultado, na Figura 4.7, aparece uma área escura, que é referente ao fim do cenário 3D na UE4. Nessa região, não havia mais elementos para reconstrução de pontos das imagens, resultando na escuridão à direita. Portanto, a medida aproximada de $53\ m$ é coerente com a trajetória percorrida pelo drone.

4.2.1.2 Validação do algoritmo de georreferenciamento direto de pixels

Dados os resultados relacionados à qualidade dos arquivos de imagem geolocalizados, compatíveis para o uso com softwares comerciais, como o WebODM, o próximo passo é verificar o funcionamento do algoritmo de georreferenciamento associado a detecção da pessoa

Figura 4.6: Ortofoto gerada a partir das fotos obtidas do ambiente 3D no cenário rural.

Tabela 4.2: Relatório gerado pelo WebODM para o cenário rural

Imagens reconstruídas	15 de 15 fotos (100.0%)
Pontos reconstruídos (esparsos)	26035 de 30933 pontos (84.2%)
Pontos reconstruídos (densos)	856,168 pontos
Ground Sampling Distance (GSD) - média	1.4 cm
Características detectadas	24,769 características
Características reconstruídas	5,155 caraterísticas
Referencia geográfica	GPS
Erros de GPS	0.59 metros

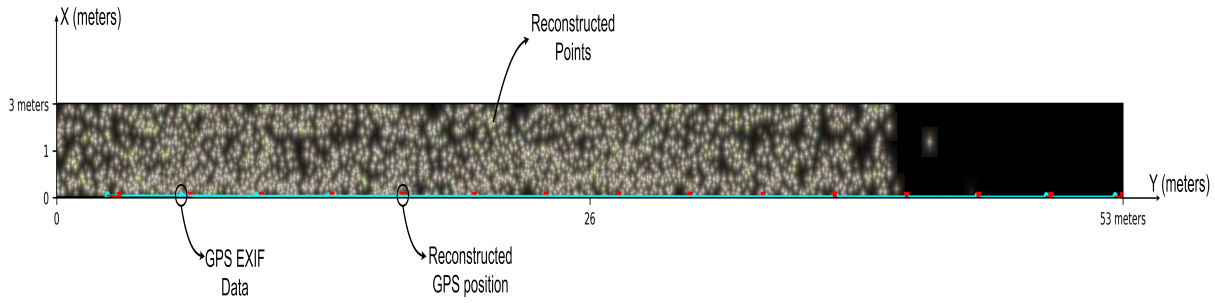

Figura 4.7: Nuvem de pontos com os waypoints estimados que o drone percorreu no cenário rural reconstruído pelo WebODM.

que está deitada no meio da plantação. Para isso, entre alguns testes, notou-se a necessidade de fazer uma verificação a parte para validar essa funcionalidade da metodologia proposta visto a dificuldade da YOLOv5 detectar pessoas nesse ambiente simulado a alturas mais elevadas (como a de $15\ m$) utilizadas para capturar as imagens geolocalizadas.

Por isso, realizou-se uma missão similar a anterior, porém com o drone voando a 5 metros de altura e assim a rede YOLOv5 conseguiu detectar a pessoa deitada. Como a proposta de detecção em tempo real (com o método de streaming) fica incompatível com a apresentação escrita deste trabalho, o vídeo da missão pode ser visto em [68]. Aqui será colocado uma imagem que o drone capturou com a câmera associada, na resolução de 2048 x 1024, para validar o algoritmo de georreferenciamento. A imagem com a *bounding box* resultante da detecção da YOLOv5 pode ser vista na Figura 4.8. Para esta resolução, a matriz com os parâmetros intrínsecos da câmera é:

$$K = \begin{bmatrix} 1024 & 0 & 1024 \\ 0 & 1536 & 512 \\ 0 & 0 & 1 \end{bmatrix}. \tag{4.3}$$

O vetor com o P_i para essa imagem é $[218, 598]$ (correspondente a x_{centro} e y_{centro}, respectivamente) conforme mostra a Figura 4.9, com a saída no prompt de comando do Windows ao se executar o código da de detecção da YOLOv5. Com esses parâmetros definidos e a posição do drone no momento da foto sendo correspondente a $T_{NED \rightarrow ENU}$ = *[-3.701801293303139, -47.764904969220275, 137.01708984375]* (latitude, longitude e altitude do drone no momento da foto), o valor das coordenadas geográficas do pixel georreferenciado em graus decimais é *pixel_georeferrenciado = [-3.70180371, -47.76494038]*. Em coordenadas UTM, o valor da latitude e longitude, em metros, é de *9590354.981307276 m S* e *192849.96228310984 m E*, respectivamente.

A posição da pessoa em coordenadas UTM pode ser obtida somando o deslocamento da mesma em relação a coordenada conhecida do *OriginGeoPoint*. Como a Unreal trabalha com as distâncias em centímetros, convertendo para metros, verificou-se que a pessoa está deslocada 1.6 metros na longitude e 15.7 metros na latitude. Logo, a posição da pessoa em coordenadas UTM é de *9590353.935376592 m S* e *192850.43382590482 m E*. Em graus decimais, essa posição corresponde as coordenadas [-3.7018131746639136, -47.764936167875284]. Assim, considerando até a 4ª casa decimal, a localização retornada pelo algoritmo de georreferenciamento direto não obteve erros em relação a posição real do objeto no ambiente simulado. Realizando a diferença entre as coordenadas UTM, obtém-se que o erro em metros é de $1.05\ m$ na latitude

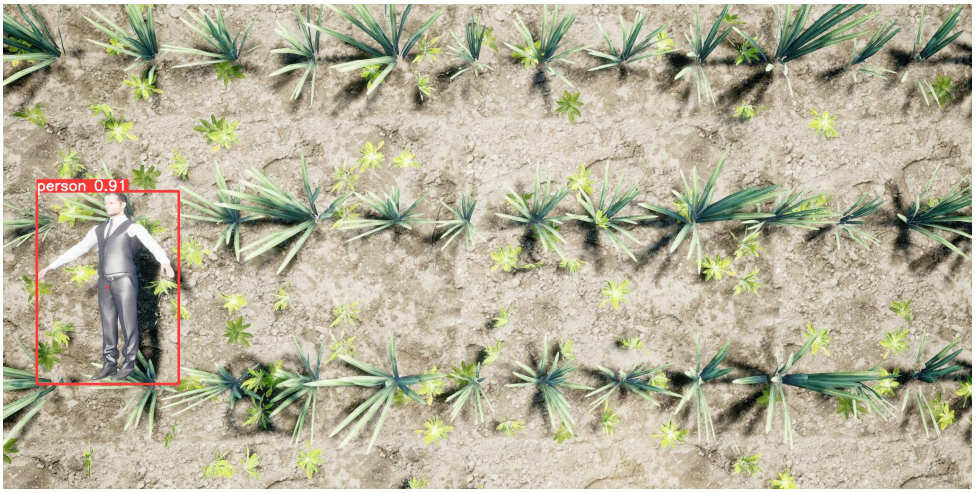

Figura 4.8: Imagem obtida com o drone voando a 5 metros de altura e com a detecção de pessoas realizada pela YOLOv5 no cenário rural.

```
G:\Meu Drive\LASSE\Drone_project\projetos\Yolo\yolov5>python detect.py --weights yolov5s.pt --class 0 --source ..\..\scripts_developed\images_guidance\2023-05-17_test5_5m_eric\imagens_geolocalizadas\photo2.jpg
C:\Users\lucas\AppData\Local\Programs\Python\Python310\lib\site-packages\pyproj\crs\crs.py:141: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>' is the preferred initialization method. When making the
change, be mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-in-proj-6
  in_crs_string = _prepare_from_proj_string(in_crs_string)
detect: weights=['yolov5s.pt'], source=..\..\scripts_developed\images_guidance\2023-05-17_test5_5m_eric\imagens_geolocalizadas\photo2.jpg, data=data\coco128.yaml, imgsz=[640, 640], conf_thres=0.25, iou_thres=0.45, max_det=1000, device=,
view_img=False, save_txt=False, save_conf=False, save_crop=False, nosave=False, classes=[0], agnostic_nms=False, augment=False, visualize=False, update=False, project=runs\detect, name=exp, exist_ok=False, line_thickness=3, hide_labels=F
alse, hide_conf=False, half=False, dnn=False
fatal: cannot change to 'G:\Meu': No such file or directory
YOLOv5  2022-5-12 torch 1.13.0+cpu CPU

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients

x_center = 218
y_center = 598

image 1/1 G:\Meu Drive\LASSE\Drone_project\projetos\scripts_developed\images_guidance\2023-05-17_test5_5m_eric\imagens_geolocalizadas\photo2.jpg: 320x640 1 person, Done. (0.112s)
Speed: 0.0ms pre-process, 112.0ms inference, 2.0ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs\detect\exp66
```

Figura 4.9: Saída no prompt de comandos do Windows ao se executar o código de detecção da YOLOv5 em imagem no cenário rural.

e $0.472\ m$ na longitude.

Para testar de forma prática se os erros acima são aceitáveis para a missão, pode-se assumir o contexto onde, dado que um primeiro drone estava realizando a missão de monitoramento e/ou busca e resgate, quando a rede embarcada detecta uma pessoa, um sinal sonoro é emitido ao operador e a posição em GPS é guardada em arquivo de texto. Com as imagens geolocalizadas, o vídeo com a detecção produzido pela YOLOv5 em associação com o código *Streamer*, mas principalmente com a posição do pixel georreferenciado, um segundo drone pode ser enviado para guiar o operador para onde a pessoa em situação de risco se encontra. Com as coordenadas obtidas nesse teste, é possível enviar o drone para a localização do pixel georreferenciado com o comando *moveToGPSAsync(-3.70180371, -47.76494038, 142, 5).join()* - onde o terceiro parâmetro é uma altitude arbitrada como suficiente para que a câmera do drone capte a pessoa no campo de visão. Com isso, obteve-se o posicionamento do drone bem acima da pessoa deitada, como mostra a Figura 4.10, onde a sub-janela mais a esquerda mostra a captura da câmera do drone (as *bounding boxes* foram inseridas manualmente, apenas para facilitar a visualização da pessoa deitada). A sub-janela ao centro é apenas para demonstração da visualização de imagem do tipo segmentada que o AirSim também permite obter.

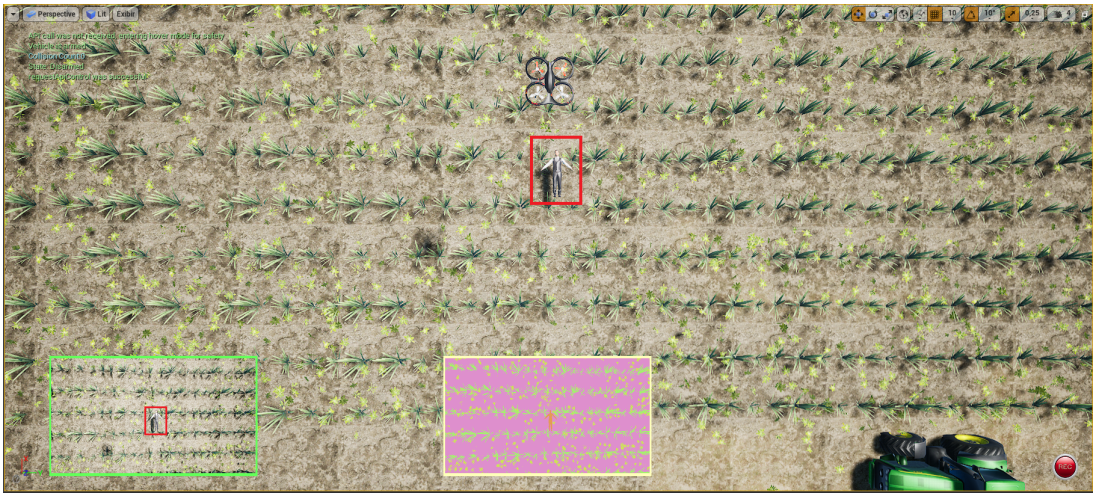

Figura 4.10: Posicionamento do drone acima da pessoa deitada usando a posição GPS retornada pelo algoritmo de georreferenciamento.

4.2.2 Resultados em ambiente 3D representando um gêmeo digital

O outro cenário implementado para testes dos algoritmos foi obtido com auxílio da ferramenta Blender e os plugins associados, *blender-osm* e *bpyproj* (descritos na Subseção 2.1.5).

O ambiente 3D representado na Figura 4.11 é uma cópia digital de um quarteirão na cidade de Witten, Alemanha. Usando o plugin *blender-osm* foram extraídos os prédios referentes a essa localização específica, onde é possível observar a semelhança com o mundo real comparando com uma imagem capturada do aplicativo Google Earth, como mostra a Figura 4.12. Também foi obtida uma imagem 2D do mapa do local, a qual foi utilizado como sobreposição no solo do ambiente 3D, para dar uma melhor referência de localização ao cenário. O sistema de coordenadas projetadas utilizado, com auxílio do plugin *bpyproj*, foi referente ao EPSG: 3857, o qual é usado para renderizar mapas em aplicações como o Google Maps e OpenStreetMap [69]. Com auxílio do módulo *AirsimGeo* é possível movimentar o drone, usando o sistema de coordenadas escolhido, com o método *moveToPositionAsyncGeo()* e obter as coordenadas GPS com o método *getGpsLocation()*.

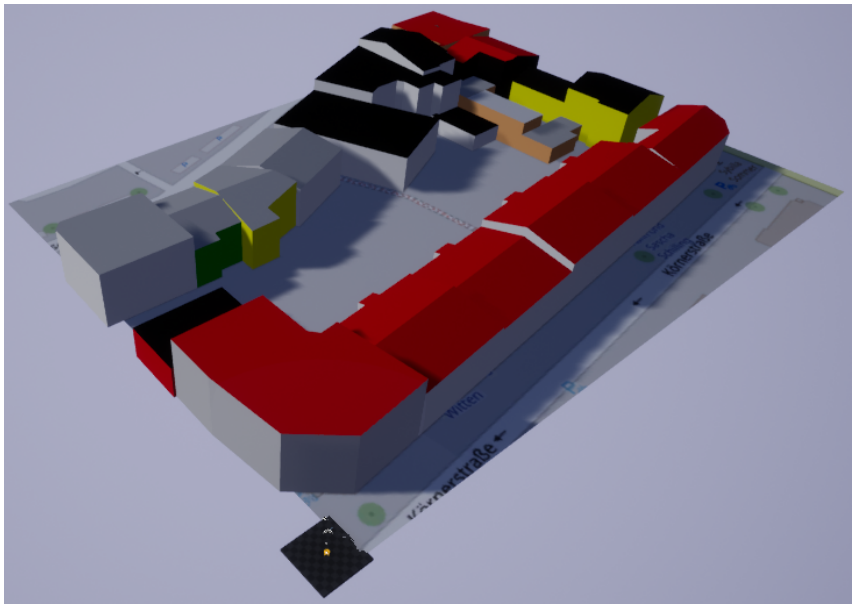

Figura 4.11: Ambiente 3D na unreal representando uma localidade da cidade de Witten, Alemanha.

Para este cenário, o arquivo *settings.json* foi configurado conforme apresenta a Figura 4.4, mudando apenas a latitude, longitude e altitude do *OriginGeoPoint* para a coordenada: [51.43672, 7.33577, 94] (correspondente a zona UTM 32).

4.2.2.1 Validação dos resultados referentes às imagens geolocalizadas

Nesta missão, foram obtidas 12 fotos com as localizações GPS do drone no momento em que cada uma foi capturada. Utilizando o software WebODM, foi possível realizar o processamento das mesmas a fim de validar a geração de arquivos de imagens geolocalizadas. A

Figura 4.12: Localidade da cidade de Witten, Alemanha, utilizada como base para o ambiente 3D.

Figura 4.13 apresenta a ortofoto gerada pela software sobreposta ao mapa, encaixando corretamente ao quarteirão na cidade de Witten referente aos dados GPS. Apesar de que a porcentagem de reconstrução dos pontos foi inferior nesse caso, assim como o número de características detectadas a partir das imagens fornecidas ao WebODM, nota-se que o software ainda conseguiu reconstruir o cenário de forma aproximada ao mapa real.

Figura 4.13: Ortofoto gerada a partir das fotos obtidas do ambiente 3D no cenário urbano.

Tabela 4.3: Relatório gerado pelo WebODM para o cenário urbano

Imagens reconstruídas	12 de 12 fotos (100.0%)
Pontos reconstruídos (esparsos)	3091 de 4031 pontos (76.7%)
Ground Sampling Distance (GSD) - média	6.4 cm
Pontos reconstruídos (densos)	235,620 pontos
Características detectadas	7,135 características
Características reconstruídas	793 caraterísticas
Referencia geográfica	GPS
Erros de GPS	0.36 metros

4.2.2.2 Validação do algoritmo de georreferenciamento direto de pixels

Para validar o georreferenciamento direto de pixel, uma pessoa foi posicionada no ambiente 3D nas coordenadas GPS: [51.437347, 7.337117]. Ou, *5699772.99 m N* e *384420.17 m E*, em coordenadas UTM.

Com isso, voando o drone até a posição GPS: [51.437362, 7.337106, 113] com o método *moveToPositionAsyncGeo*, foram capturadas as Figuras 4.14 e 4.16. Na primeira, a câmera foi configurada para estar a 90° em relação ao drone (definindo $\psi_G = 90°$). O vetor P_i para essa imagem é [1442, 210] conforme mostra a Figura 4.15, com a saída no prompt de comando do Windows ao se executar o código da de detecção da YOLOv5. A posição do drone no momento da foto é representado pelo vetor $T_{NED \rightarrow ENU}$ e como a pessoa está a uma altura $13.36m$ (já que a pessoa está em cima de um prédio), pode-se obter a altura do drone com o método $simGetVehiclePose()$ e assim calcular z_{ENU} como a diferença entre as alturas (aproximadamente $5.4\ m$). Logo, a posição retornada pelo algoritmo de georreferenciamento direto em graus decimais é $pixel\_georeferrenciado = [51.437372, 7.337137]$. Realizando a diferença entre as coordenadas reais e calculada pelo algoritmo, o erro é de $2.74\ m$ na latitude e $1.42\ m$ na longitude.

A Figura 4.16 foi capturada movimentando o drone para a mesma posição do cenário que na Figura 4.14, porém, foi configurado para que a câmera estivesse inclinada em 60° em relação ao drone ($\psi_G = 60°$), a fim de demonstrar a generalidade do algoritmo de georreferenciamento direto de pixels em diferentes situações. Assim, com a inclinação do gimbal (logo, da câmera) diferente, a pessoa naturalmente está posicionada em um local diferente em relação a situação

Figura 4.14: Imagem obtida com o drone voando a 5.4 metros de altura em relação à pessoa detectada pela YOLOv5 ($\psi_G = 90°$).

```
G:\Meu Drive\LASSE\Drone_project\projetos\Yolo\yolov5>python detect.py --weights yolov5s.pt --source ..\..\airsimgeo\examples\images_guidance_witten\photo0.jpg
C:\Users\lucas\AppData\Local\Programs\Python\Python310\lib\site-packages\pyproj\crs\crs.py:141: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>' is the preferred initialization method. When making the
change, be mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-in-proj-6
  in_crs_string = _prepare_from_proj_string(in_crs_string)
detect: weights=['yolov5s.pt'], source=..\..\airsimgeo\examples\images_guidance_witten\photo0.jpg, data=data\coco128.yaml, imgsz=[640, 640], conf_thres=0.25, iou_thres=0.45, max_det=1000, device=, view_img=False, save_txt=False, save_con
f=False, save_crop=False, nosave=False, classes=None, agnostic_nms=False, augment=False, visualize=False, update=False, project=runs\detect, name=exp, exist_ok=False, line_thickness=3, hide_labels=False, hide_conf=False, half=False, dnn=
False
fatal: cannot change to 'G:\Meu': No such file or directory
YOLOv5  2022-5-12 torch 1.13.0+cpu CPU

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients

x_center = 1442
y_center = 210

image 1/1 G:\Meu Drive\LASSE\Drone_project\projetos\airsimgeo\examples\images_guidance_witten\photo0.jpg: 320x640 1 airplane, Done. (0.107s)
Speed: 2.0ms pre-process, 107.0ms inference, 8.0ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs\detect\exp69
```

Figura 4.15: Saída no prompt de comandos do Windows ao se executar o código de detecção da YOLOv5 em imagem no cenário urbano ($\psi_G = 90°$).

anterior. O vetor P_i para essa imagem é [1407, 729] conforme mostra a Figura 4.17, com a saída no prompt de comando do Windows ao se executar o código da de detecção da YOLOv5. Com isso, a posição, em graus decimais, da pessoa detectada de acordo com o algoritmo de georreferenciamento direto é $pixel\_georreferenciado = [51.437382, 7.337136]$. O erro encontrado foi de $3.84\ m$ na latitude e $1.43\ m$ na longitude.

Neste exemplo de detecção de pessoas com a YOLOv5, nota-se que a rede erra na inferência e classifica o objeto detectado como *airplane*. Porém, como o foco do trabalho é a integração do algoritmo de detecção de objetos com o de georreferenciamento, considerou-se apenas a detecção realizada (mesmo com o erro) a fim de se obter o pixel de interesse.

Figura 4.16: Imagem obtida com o drone voando a 5.4 metros de altura em relação à pessoa detectada pela YOLOv5 ($\psi_G = 60°$).

```
G:\Meu Drive\LASSE\Drone_project\projetos\Yolo\yolov5>python detect.py --weights yolov5s.pt --source ..\..\airsimgeo\examples\images_guidance_witten\photo1.jpg
C:\Users\lucas\AppData\Local\Programs\Python\Python310\lib\site-packages\pyproj\crs\crs.py:141: FutureWarning: '+init=<authority>:<code>' syntax is deprecated. '<authority>:<code>' is the preferred initialization method. When making the
change, be mindful of axis order changes: https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-in-proj-6
  in_crs_string = _prepare_from_proj_string(in_crs_string)
detect: weights=['yolov5s.pt'], source=..\..\airsimgeo\examples\images_guidance_witten\photo1.jpg, data=data\coco128.yaml, imgsz=[640, 640], conf_thres=0.25, iou_thres=0.45, max_det=1000, device=, view_img=False, save_txt=False, save_con
f=False, save_crop=False, nosave=False, classes=None, agnostic_nms=False, augment=False, visualize=False, update=False, project=runs\detect, name=exp, exist_ok=False, line_thickness=3, hide_labels=False, hide_conf=False, half=False, dnn=
False
fatal: cannot change to 'G:\Meu': No such file or directory
YOLOv5  2022-5-12 torch 1.13.0+cpu CPU

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients

x_center = 1407
y_center = 729

image 1/1 G:\Meu Drive\LASSE\Drone_project\projetos\airsimgeo\examples\images_guidance_witten\photo1.jpg: 320x640 1 airplane, Done. (0.103s)
Speed: 2.0ms pre-process, 103.0ms inference, 8.0ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs\detect\exp71
```

Figura 4.17: Saída no prompt de comandos do Windows ao se executar o código de detecção da YOLOv5 em imagem no cenário urbano ($\psi_G = 60°$).

Capítulo 5

Considerações Finais

Neste trabalho foi possível observar os passos para implementação de uma metodologia de simulação com VANTs capaz de gerar imagens geolocalizadas e pixels georreferenciados, associados a detecção de objetos em arquivos de imagens. Apesar do trabalho não focar no treinamento de algoritmos de detecção de objetos, como a utilizada YOLO, a solução proposta traz a possibilidade de gerar dados sintéticos para treinar essas redes neurais de forma menos custosa. Além disso, é apresentada uma proposta de como manipular o código da YOLOv5 a fim de integrá-lo de forma automatizada ao algoritmo de georreferenciamento direto. O código-fonte desenvolvido assim como informações sobre documentação e onde encontrar os cenários 3D usados, podem ser encontrados juntamente com o repositório LaTeX na URL (acessível para colaboradores do LASSE): https://gitlab.lasse.ufpa.br/latex/master-dissertation/2022-lucasconde-master. Informações também podem ser encontradas em repositório público no GitHub.

Quanto ao algoritmo de georreferenciamento direto, ele se apresenta como tendência em missões onde busca-se cada vez mais a automatização dos UAS, pois em situações, como as de reconhecimento, busca e resgate, não é possível posicionar PCTs, sendo necessário outras soluções para geolocalização de objetos de interesse no cenário de acordo com as imagens capturadas pelo VANT em associação com seus sensores que auxiliam na localização do drone. Dessa forma, com a metodologia proposta é possível estimar a posição do objeto de interesse por meio da própria posição GPS do drone, que fica gravada nos metadados EXIF dos arquivos de imagem no momento em que a câmera associada ao veículo captura a imagem com o objeto em questão. E com o algoritmo de georreferenciamento direto do pixel central da bounding box gerada pela YOLO em torno do objeto de interesse, é possível se aproximar ainda mais da localização exata buscada, reduzindo o erro a menos de 5 metros da posição real, como

demonstrado nos resultados apresentados. Esses erros podem ser minimizados ainda mais com a calibração dos sensores os quais retornam a atitude do drone e o posicionamento do mesmo, seja no sistema de posicionamento global (GPS), quanto em relação ao próprio objeto de interesse. Como para demonstração da metodologia levou-se em conta a atitude configurada no arquivo *settings.json* e a posição do drone retornada pelos sensores simulados pelo Airsim, os erros relacionados a estimação retornada por esses sensores, pode culminar em erros maiores do algoritmo de georreferenciamento, como foi visto no cenário da Subseção 4.2.2.

Apesar disso, o trabalho obteve êxito em demonstrar a metodologia de geração de arquivos de imagens geolocalizadas, compatíveis com softwares comerciais, em ambientes 3D fictícios e gêmeos digitais, agregando ainda a técnica de georreferenciamento direto de pixels. Com isso, essa dissertação agrupou várias referências associadas ao estado da arte no uso de motores gráficos e simuladores de veículos autônomos, em aplicações voltadas ao uso de IA aplicada a detecção de objetos e georreferenciamento dos mesmos.

5.1 Trabalhos Futuros

A fim de melhorar a metodologia proposta, um dos esforços que podem ser feitos em trabalhos futuros é a geração de um conjunto de dados sintéticos para treino da YOLOv5 e assim, melhorar a acurácia da mesma para detectar pessoas, por exemplo. Além disso, buscar automatizar ainda mais a integração entre o código de guiamento e o de georreferenciamento é uma abordagem para minimizar os erros quanto ao georreferenciamento de pixels, obtendo os ângulos referente a atitude exata do drone no momento da captura das imagens.

Outro passo para aumentar o realismo da simulação seria usar o modo HITL, onde, usando os mesmos métodos implementados neste trabalho (para o controlador *Pixhawk* associado ao PX4), é possível movimentar o drone e capturar sua posição em coordenadas GPS, usando sensores reais e adicionando funcionalidades como a medição do consumo de bateria em uma missão. Tal aplicação necessitaria de uma calibração dos sensores associados, mas traria ainda mais vantagens quanto a otimização dos algoritmos antes que esses fossem implementados em um sistema real. Entretanto, uma dificuldade que pode ser encontrada, é a compatibilidade entre as diferentes versões dos firmwares e APIs envolvidos e até mesmo o funcionamento deste com o sistema operacional. Por exemplo, no decorrer do desenvolvimento desta pesquisa, não obteve-se êxito em integrar uma simulação SITL no Windows 11. E devido à mudança de

máquinas durante o período onde esse trabalho foi desenvolvido, o método *moveToGPSAsync()* começou a não funcionar adequadamente com o firmware PX4.

Referências Bibliográficas

[1] I. Nnoli, "Ultra-Realism Made Accessible with NVIDIA AI and Path Tracing Technologies," https://developer.nvidia.com/blog/ultra-realism-made-accessible-with-ai-and-path-tracing-technologies/, 2023, acesso em 31/05/2023.

[2] "NVIDIA DLSS," https://www.nvidia.com/en-us/geforce/technologies/dlss/, acesso em 31/05/2023.

[3] "5 Real World Examples of Digital Twins," https://www.palamir.com/news/5-real-world-examples-of-digital-twins#:~:text=Singapore%20and%20Shanghai%20both%20have,increase%20the%20wellbeing%20of%20residents., acesso em 31/05/2023.

[4] "Ford Motor Company: Dearborn Campus Uses a Digital Twin Tool for Energy Plant Management," https://betterbuildingssolutioncenter.energy.gov/implementation-models/ford-motor-company-dearborn-campus-uses-a-digital-twin-tool-energy-plant, acesso em 31/05/2023.

[5] M. GEYER, "BMW Group Starts Global Rollout of NVIDIA Omniverse," https://blogs.nvidia.com/blog/2023/03/21/bmw-group-nvidia-omniverse/, 2023, acesso em 31/05/2023.

[6] "CAD Center creates Tokyo-sized digital twins with Twinmotion and Unreal Engine," https://www.unrealengine.com/en-US/spotlights/cad-center-creates-tokyo-sized-digital-twins-with-twinmotion-and-unreal-engine, 2021, acesso em 31/05/2023.

[7] "NVIDIA Omniverse," https://www.nvidia.com/pt-br/omniverse/#, acesso em 30/05/2023.

[8] S. Shah, D. Dey, C. Lovett, and A. Kapoor, “Airsim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles,” in *Field and Service Robotics*, 2017. [Online]. Available: https://arxiv.org/abs/1705.05065

[9] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “ CARLA: An Open Urban Driving Simulator,” in *Proceedings of the 1st Annual Conference on Robot Learning*, 2017, pp. 1–16.

[10] “Categorias – Competição Brasileira de Robótica,” https://www.cbrobotica.org/index.php/categorias/, acesso em 31/05/2023.

[11] “Gazebo,” https://gazebosim.org/home, acesso em 31/05/2023.

[12] B. Mishra, D. Garg, P. Narang, and V. Mishra, “Drone-surveillance for search and rescue in natural disaster,” *Computer Communications*, vol. 156, pp. 1–10, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0140366419318602

[13] I. Martinez-Alpiste, G. Golcarenarenji, Q. Wang, and J. M. Alcaraz-Calero, “Search and rescue operation using UAVs: A case study,” *Expert Systems with Applications*, vol. 178, p. 114937, 2021.

[14] L. Xing, X. Fan, Y. Dong, Z. Xiong, L. Xing, Y. Yang, H. Bai, and C. Zhou, “Multi-UAV cooperative system for search and rescue based on YOLOv5,” *International Journal of Disaster Risk Reduction*, vol. 76, p. 102972, 2022.

[15] C. A. Correia, F. A. Andrade, A. Sivertsen, I. P. Guedes, M. F. Pinto, A. G. Manhães, and D. B. Haddad, “Comprehensive Direct Georeferencing of Aerial Images for Unmanned Aerial Systems Applications,” *Sensors*, vol. 22, no. 2, p. 604, 2022.

[16] X. Liu, X. Lian, W. Yang, F. Wang, Y. Han, and Y. Zhang, “Accuracy Assessment of a UAV Direct Georeferencing Method and Impact of the Configuration of Ground Control Points,” *Drones*, vol. 6, no. 2, 2022. [Online]. Available: https://www.mdpi.com/2504-446X/6/2/30

[17] F. A. D. A. Andrade, A. H. Sivertsen, C. A. M. Correia, N. Belbachir, L. C. De Sousa, V. M. P. Rufino, E. P. Petrópolis, E. R. e Silva, and V. H. R. F. Henriques, “Virtual Reality Simulation of Autonomous Solar Plants Inspections with Unmanned Aerial Systems,” in

2021 Aerial Robotic Systems Physically Interacting with the Environment (AIRPHARO). IEEE, 2021, pp. 1–8.

[18] L. Bommes, C. Buerhop-Lutz, T. Pickel, J. Hauch, C. Brabec, and I. Marius Peters, "Georeferencing of photovoltaic modules from aerial infrared videos using structure-from-motion," *Progress in Photovoltaics: Research and Applications*, vol. 30, no. 9, pp. 1122–1135, 2022.

[19] R. Chang, S. Zhou, Y. Zhang, N. Zhang, C. Zhou, and M. Li, "Research on Insulator Defect Detection Based on Improved YOLOv7 and Multi-UAV Cooperative System," *Coatings*, vol. 13, no. 5, 2023. [Online]. Available: https://www.mdpi.com/2079-6412/13/5/880

[20] E. S. D. O. Junior, "Geração Procedural de Dados Sintéticos Realistas para Treinamento de Modelos de Aprendizado de Máquina: Aplicação sobre agricultura de precisão com base em imagens de drones," Master's thesis, Universidade Federal do Pará, 2022.

[21] Y. Hu, X. Wu, G. Zheng, and X. Liu, "Object Detection of UAV for Anti-UAV Based on Improved YOLO v3," in *2019 Chinese Control Conference (CCC)*, 2019, pp. 8386–8390.

[22] G. Jocher, "YOLOv5 by Ultralytics," May 2020. [Online]. Available: https://github.com/ultralytics/yolov5

[23] L. Conde, G. Souza, Y. Souza, F. Nunes, M. Takeda, A. Castro, and A. Klautau, "SoC FPGA-based Beacon Emulator Platform for a Three-Axis Satellite Antenna," in *2020 International Conference on Computing, Electronics & Communications Engineering (iC-CECE)*. IEEE, 2020, pp. 217–223.

[24] F. Garrett, "O que é Unreal Engine? Entenda tecnologia de gráficos de jogos e consoles," https://www.techtudo.com.br/noticias/2020/07/o-que-e-unreal-engine-entenda-tecnologia-de-graficos-de-jogos-e-consoles.ghtml, 2020, acesso em 30/05/2023.

[25] "51WORLD - Leading Digital Twin Solution Provider," https://www.51vr.com.au/, acesso em 30/05/2023.

[26] D. Weir-McCall, "51World creates digital twin of the entire city of Shanghai," https://www.unrealengine.com/en-US/spotlights/

51world-creates-digital-twin-of-the-entire-city-of-shanghai, 2020, acesso em 30/05/2023.

[27] R. Kerris, "Ericsson Builds Digital Twins for 5G Networks in NVIDIA Omniverse," https://blogs.nvidia.com/blog/2021/11/09/ericsson-digital-twins-omniverse/, 2021, acesso em 31/05/2023.

[28] J. P. F. Guimarães, "Controle de Atitude e Altitude Para um Veículo Aéreo Não Tripulado do Tipo Quadrirrotor," Master's thesis, Universidade Federal do Rio Grande do Norte, 2012.

[29] "About – blender.org," https://www.blender.org/about/, acesso em 31/05/2023.

[30] "blender-osm: OpenStreetMap and terrain for Blender," https://github.com/vvoovv/blender-osm, acesso em 31/05/2023.

[31] "OpenStreetMap," https://www.openstreetmap.org/, acesso em 31/05/2023.

[32] "Blender Map Projection Plugin," https://github.com/JeremyBYU/bpyproj, acesso em 31/05/2023.

[33] J. Castagno, "AirSim Georeferenced World - Part 1," https://robosim.dev/posts/airsim-geo/, 2019, acesso em 31/05/2023.

[34] "AirSimGeo," https://github.com/JeremyBYU/airsimgeo, acesso em 31/05/2023.

[35] "pyproj 3.5.0," https://pypi.org/project/pyproj/, acesso em 14/05/2023.

[36] "What Is Object Detection?" https://www.mathworks.com/discovery/object-detection.htmll, acesso em 07/05/2023.

[37] "What Is a Convolutional Neural Network?" https://www.mathworks.com/discovery/convolutional-neural-network-matlab.html, acesso em 07/05/2023.

[38] G. Alves, "Understanding ConvNets (CNN)," https://medium.com/neuronio/understanding-convnets-cnn-712f2afe4dd3, 2018, acesso em 07/05/2023.

[39] S. Saha, "A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way," https://towardsdatascience.com/

a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53, 2018, acesso em 07/05/2023.

[40] Z. J. Wang, R. Turko, O. Shaikh, H. Park, N. Das, F. Hohman, M. Kahng, and D. H. P. Chau, "CNN Explainer: Learning Convolutional Neural Networks with Interactive Visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 27, no. 2, pp. 1396–1406, feb 2021. [Online]. Available: https://doi.org/10.1109%2Ftvcg.2020.3030418

[41] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *2017 International Conference on Engineering and Technology (ICET)*, 2017, pp. 1–6.

[42] J. deVilla, "How to Read and Remove Metadata from Your Photos With Python," https://auth0.com/blog/read-edit-exif-metadata-in-photos-with-python/, 2021.

[43] V. T., "Understanding Geotagging In Photos," https://medium.com/hd-pro/understanding-geotagging-in-photos-d67097b5bd44, 2019, acesso em 07/05/2023.

[44] "exif 1.6.0," https://pypi.org/project/exif/, acesso em 07/05/2023.

[45] "piexif 1.1.3," https://pypi.org/project/piexif/, acesso em 07/05/2023.

[46] H. Fradi, L. Bracco, F. Canino, and J.-L. Dugelay, "Autonomous Person Detection and Tracking Framework Using Unmanned Aerial Vehicles (UAVs)," in *2018 26th European Signal Processing Conference (EUSIPCO)*, 2018, pp. 1047–1051.

[47] "Sistemas de Coordenadas," https://docs.qgis.org/3.28/pt_PT/docs/gentle_gis_introduction/coordinate_reference_systems.html, acesso em 31/05/2023.

[48] "Coordenadas topográficas X Coordenadas UTM," https://mundogeo.com/2013/06/05/coordenadas-topograficas-x-coordenadas-utm/#:~:text=UTM%20%C3%A9%20um%20sistema%20de,a%20posi%C3%A7%C3%A3o%20de%20um%20objeto., acesso em 31/05/2023.

[49] "Sirgas 2000 e WGS84: o que são e como converter?" https://mappa.ag/blog/sirgas-2000-e-wgs84-o-que-sao/, 2023.

[50] "2.2 DATUMS & WORLD MODELS," https://www.vectornav.com/resources/inertial-navigation-primer/math-fundamentals/math-datums, acesso em 31/05/2023.

[51] "EPSG.io: Coordinate Systems Worldwide," https://epsg.io/, acesso em 31/05/2023.

[52] "2.1 REFERENCE FRAMES," https://www.vectornav.com/resources/inertial-navigation-primer/math-fundamentals/math-refframes, acesso em 31/05/2023.

[53] "Coordinate System," https://www.basicairdata.eu/knowledge-center/background-topics/coordinate-system/, acesso em 31/05/2023.

[54] "Create a Video-Stream," https://github.com/Microsoft/AirSim/issues/892, acesso em 14/05/2023.

[55] "Flask," https://flask.palletsprojects.com/en/2.3.x/, acesso em 14/05/2023.

[56] "Numpy," https://numpy.org/, acesso em 14/05/2023.

[57] E. Krings, "Decoding 720p: The Best Streaming Resolution Settings for Live Quality Video," https://www.dacast.com/blog/best-h-264-encoder-settings-for-live-streaming/, 2022, acesso em 14/05/2023.

[58] "The Best Resolution For YouTube: A Complete Guide," https://offeo.com/learn/best-resolution-for-youtube#:~:text=The%20most%20common%20resolutions%20for,)%2C%20known%20as%204K%20resolution., 2022.

[59] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[60] "Welcome to navpy's documentation!" https://navpy.readthedocs.io/en/latest/, acesso em 14/05/2023.

[61] "Demo Agro v1.2," https://youtu.be/o_qqH2Ka0-M, acesso em 12/12/2021.

[62] “Integration of Airsim in SITL simulation with Yolov5,” https://youtu.be/SPGJN546Qsk, acesso em 10/05/2022.

[63] “WebODM Drone Software - OpenDroneMap,” https://www.opendronemap.org/webodm/, acesso em 31/05/2023.

[64] “Quality report details,” https://community.opendronemap.org/t/quality-report-details/11451, 2022.

[65] “Statistics and Quality Report,” https://opensfm.org/docs/quality_report.html, 2021.

[66] “Ground sampling distance (gsd) in photogrammetry,” https://support.pix4d.com/hc/en-us/articles/202559809-Ground-sampling-distance-GSD-in-photogrammetry, acesso em 23/08/2023.

[67] “What is ground sample distance (gsd)?” https://visionaerial.com/what-is-ground-sample-distance/#:~:text=The%20range%20for%20UAV%20photogrammetry,surveys%2C%20which%20is%20very%20low., 2021.

[68] “Missão com Airsim transmitindo imagens para YOLOv5,” https://youtu.be/vnCsrG5JJz4, acesso em 21/05/2022.

[69] “EPSG:3857: WGS 84 / Pseudo-Mercator – Mercator esférico, Google Maps, OpenStreetMap, Bing, ArcGIS, ESRI,” https://epsg.io/3857, acesso em 11/05/2023.