FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

A Distributed and Scalable Approach for Drone Navigation Based on State-of-the-Art Computer Vision Techniques

Henrique Reis Pedroso Munhoz Rubio

Mestrado em Engenharia Eletrotécnica e de Computadores

Supervisor: Professor António Pedro Rodrigues Aguiar

November 5, 2025

Resumo

A presente dissertação explora a navegação autónoma dos Veículos Aéreos Não Tripulados (VANTs), comumente denominados drones, baseada exclusivamente em visão computacional e com um enfoque distribuído e escalável. O trabalho desenvolvido enfrentou os problemas referentes à substituição de sensores sofisticados pelo uso apenas de câmaras para a função de navegação autónoma. A necessidade por soluções com flexibilidade e economia para a operação de drones de forma autónoma em ambientes dinâmicos serviu de motivação para a construção de sistemas de navegação igualmente eficientes e robustos. Desta forma, este estudo foi implementado dentro de uma estrutura modular usando contêineres Docker em conjunto com ROS2 e simulações no ambiente Gazebo, garantindo assim a flexibilidade para serem usados em plataformas que possuam um hardware mais restrito, além do processamento distribuído em nuvem.

O principal foco desta pesquisa foi o desenvolvimento e a validação de um sistema de navegação autónomo capaz de detetar e acompanhar uma pessoa, independente de sua orientação em relação ao drone e com segurança, precisão e confiabilidade. Tal abordagem foi desenvolvida baseada no YOLOv8 para a deteção e no Supervision/ByteTrack para o rastreio de objetos em tempo real. Com o objetivo de controlar a distância relativa entre o drone e o alvo, controladores PID foram usados, baseados em medições corrigidas pela calibração da câmara, sendo esta uma abordagem bastante prática e eficiente.

O sistema ainda foi validado em um ambiente de simulação bastante realista: Gazebo. As contribuições desta dissertação abrangem, inclusive, criar e validar a arquitetura escalável e distribuída para uso na navegação de drones, bem como a aplicação de modelos avançados de visão computacional para a deteção, rastreamento e monitoramento de alvos. Este estudo proporciona uma base sólida para melhorias futuras, a serem aplicadas em cenários mais complexos e desafiadores, incluindo contextos reais.

Abstract

This dissertation tackles the autonomous navigation of Unmanned Aerial Vehicles (UAVs) from a distributed and scalable approach using computer vision techniques to shed light on the challenges posed by using cameras instead of more complex sensors and devices. The need for cost-effective and adaptive solutions for the autonomous operation of drones in dynamic scenarios has pushed the development of robust and efficient navigation systems. This study was conducted in the framework of a modular architecture making use of Docker containers together with the middleware ROS2 to achieve a flexible implementation for limited hardware platforms and cloud based distributed processing.

The main purpose of this research was to design and test an autonomous navigation system capable of detecting, tracking, and following a human being safely. The system was developed based on the YOLOv8 model for real-time object detection and ByteTrack/Supervision for object tracking algorithms. The control of the relative distance between the drone and the target was carried out through PID controllers that make use of measurements derived from camera calibration, a method proved effective and convenient.

The system was validated inside a realistic simulation environment: Gazebo. Contributions of this work include the creation and validation of a distributed and scalable architecture for drone navigation, integration of advanced computer vision models for target detection, tracking, and following. This study lays the foundation for future developments and optimizations in more challenging scenarios, including real-world applications.

UN Sustainable Development Goals

This dissertation, developing and implementing an autonomous distributed scalable drone navigation system based on advanced computer vision techniques, aligns with many UN Sustainable Development Goals. Even though the system was experimented with in a simulation environment, the modular structure and use of accessible technologies grant room for further actual implementation that carries social, environmental, and educational impact.

By allowing drones to behave autonomously in difficult and dynamic environments with reduced computational power and without GPS, it is a step toward democratizing access to technology in hard-hit areas. Some possibilities include rescue operations in remote areas or places affected by the aftermath of a disaster, environmental monitoring in real-time, logistics in hard-to-reach locations, and education in under-resourced communities.

This analysis relates the set of SDGs directly attached to this work; highlights the specific goals; presents the contributions of the project; and proposes indicators and metrics to be used in further implementations to assess impact.

SDG	**Target**	**Contribution**	**Performance Indicators and Metrics**
2 - Zero Hunger	2.4	With minor modifications, the lightweight architecture can be distributed and adapted to low-cost drones that family farmers may wish to use for crop monitoring, pest detection, and irrigation optimization.	Amount of agricultural scenarios simulated/tested; accuracy in detecting critical areas; average response time to anomalies.
4 - Quality Education	4.4	This system is built with open-source technologies and may be implemented as a teaching platform for robotics, AI, and computer vision.	Number of students using the system for their training; number of tutorials and documentation developed; average installation and configuration time.
9 - Industry, Innovation, and Infrastructure	9.1	The distributed infrastructure based on Docker and ROS2 can be applied to infrastructure inspection with drones, to promote preventative and inexpensive maintenance.	Number of reusable modules; system response time; tests simulated in urban or industrial scenarios.

SDG	Target	Contribution	Performance Indicators and Metrics
9 - Industry, Innovation, and Infrastructure	9.5	This project contributes to the development of research into autonomous drones with computer vision proposing scalable solutions for low-resource environments.	Number of resulting publications; derivative projects; reuse of the architecture in other theses.
10 - Reducing Inequalities	10.2	One particular system could be tailored for contexts and communities in the rural areas who do not have cutting edge technology and can rather make use of inexpensive hardware such as Raspberry Pi.	Estimated deployment cost on cheap hardware; number of regions/countries with potential use for the solution; accessibility of the tools.
11 - Sustainable Cities and Communities	11.5	Future deployments of the system will be concerned with rescue and inspection of areas affected by the occurrence of a natural disaster.	Time to detect people in remote areas; estimated average autonomy of drones used in the simulated missions; inter-container communication robustness.
13 - Climate Action	13.1	Autonomous drones for environmental surveillance can be put into operation at remote sites with docking stations to help climate mitigation activities at a reduced cost.	Number of simulated missions with continuous detection; estimated energy consumption per mission; adaptability to different environmental conditions.
15 - Life on Land	15.1	The architecture can be adapted to environmental monitoring for drones in the collection of data on vegetation, habitats, and biodiversity.	Number of environmental surveillance simulations carried out; system scalability to several drones; and captured data quality.
15 - Life on Land	15.5	The system may be adapted to detect illegal activities within nature reserves (deforestation, hunting).	Average reduced time to identify suspicious activities; number of alerts generated by environmental anomalies.

Table 1: Alignment of the project with the Sustainable Development Goals (SDGs)

Although not yet validated in a physical environment, this work provides a solid and open foundation for future applications with real and measurable impact on several SDGs. The modular and accessible nature of the tools allow the architecture to be easily adapted to various contexts toward technological inclusion, distributed innovation, and sustainable solutions to high-risk environments, rural areas, and educational processes. Thus, this project is more significantly involved in their collective effort towards a world more sustainable, equitable, and more resilient.

Acknowledgements

This work was made possible by the contributions of and companionship of some extraordinary persons to whom I remain deeply grateful.

I sincerely wish to thank my advisor, Professor António Pedro Rodrigues Aguiar, for his patience and understanding, and above all, for the technical freedom that he accorded me to choose how to approach and shape this work. On a level between support and independence, his guidance allowed me to mature technically and personally throughout this process.

I want to deeply thank my family. They are always my solid foundation and the greatest inspiration to overcome all of life's challenges.

I thank my parents for all their unconditional support, love, and faith in me. They taught me to be strong, humble, and to fight for my dreams. I will carry their teachings with me, along with life lessons.

To my brother, I am grateful for being my companion, my advisor, and my true friend on this journey. His ideas, dedication, and presence throughout this journey were essential and made all the difference in getting here. Sharing each step with my brother made the path easier and truly valuable. I am deeply grateful for everything we have built together.

To my girlfriend, I want to thank her for her profound patience and encouragement, for her support every step of this journey. She was there to calm me in moments of doubt and to cheer me up in moments of exhaustion. I am grateful to have her by my side and for believing in me when I didn't even believe in myself.

Without the support of each of them, none of this would have been possible. I owe this achievement to each of them.

Henrique Reis Pedroso Munhoz Rubio

"It is not the strength, but the perseverance of long effort that achieves great results."

Samuel Johnson

Contents

List of Figures

List of Tables

List of Acronyms

AI	Artificial Intelligence
AMQP	Advanced Message Queuing Protocol
API	Application Programming Interface
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CV	Computer Vision
DDS	Data Distribution Service
FPS	Frames Per Second
GPU	Graphics Processing Unit
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
JSON	JavaScript Object Notation
LiDAR	Light Detection and Ranging
MAVLink	Micro Air Vehicle Communication Link
MQTT	Message Queuing Telemetry Transport
PID	Proportional–Integral–Derivative
RCLCPP	ROS 2 C++ Client Library
RCLPY	ROS 2 Python Client Library
RL	Reinforcement Learning
RMSE	Root Mean Squared Error
ROS	Robot Operating System
ROS 2	Robot Operating System 2
RT	Real Time
SDK	Software Development Kit
SDG	Sustainable Development Goal
SLAM	Simultaneous Localization and Mapping
SSL	Secure Sockets Layer
TCP	Transmission Control Protocol
TCP/IP	Transmission Control Protocol / Internet Protocol
UDP	User Datagram Protocol
UAV	Unmanned Aerial Vehicle
VLA	Visual Leader–Follower Architecture
VPS	Visual Positioning System
YAML	YAML Ain't Markup Language
YOLO	You Only Look Once
YOLOv8	You Only Look Once version 8
Gazebo	Simulation Environment for Robotics
RabbitMQ	Message Broker based on AMQP Protocol
Supervision	Python Toolkit for Multi-Object Tracking
ByteTrack	Multi-Object Tracking Algorithm based on Detection Association

Chapter 1

Introduction

This chapter presents the main topics dealt with by this dissertation under the title "A Distributed and Scalable Approach for Drone Navigation Based on State-of-the-Art Computer Vision Techniques." Section 1.1 presents the context of the work, while Section 1.2 presents the motivation for the research. The main problem discussed is developed in Section 1.3, while the general objectives of the dissertation are presented in Section 1.4. The methodology under which the work has been realized is set out in Section 1.5, followed by the methodology overview in Section 1.6. The main contributions of this work are outlined in Section 1.7, and the structure of the document is presented in Section 1.8.

1.1 Context of the Work

In recent years, drones or Unmanned Aerial Vehicles (UAVs) have attracted growing attention from various sectors such as surveillance, logistics, agriculture, infrastructure inspection, and academia. Relevant capabilities, such as the freedom to access distant or hazardous environments, and executing repetitive, autonomous, and precise tasks, have propelled the fast growth in UAVs technologies [1], [2].

Traditional drone (UAV) navigation relies on the Global Positioning System (GPS) or premade maps. However, such traditional solutions work under certain limitations: they don't work well when signal reception is poor or the scenario is too dynamic for planned algorithms [1], [3]. To tackle these limitations, this research proposes an autonomous navigation approach that is both scalable and resilient, supported by advanced computer vision algorithms within a distributed architecture.

The motivating factors for pursuing this research area are the increasing demand for fully autonomous drones that can be able to work in real-world complex scenarios with limited hardware cost. This calls for environmental perception in real time and fast decision-making [4], [5], [6].

The visual perception system has become another keystone complement to the usual sensors like GPS and IMUs (Inertial Measurement Unit) for better precision. Thus, the ability of allowing a drone to detect, track, and possibly maintain a safety distance from a moving target like a human

person on its own without having continuous human interventions has always posed a challenge in practice. Factors such as lighting variations, visual occlusions, unpredictable behaviors from the target, and heavy demanding computational nature of vision on resource-restricted platforms contribute to this difficulty. [5], [7], [8].

This thesis aims to develop and validate an autonomous drone navigation system capable of robustly detecting and tracking a person, estimating relative distance, and autonomously following the target while maintaining a safe distance.

The major contributions of this dissertation were, among others, the design and empirical simulation-based validation of a highly modular and scalable distributed architecture for autonomous drone navigation. In addition, it provided an integration and performance analysis of state-of-the-art computer vision methods for target detection and tracking in real-time and considered the conception of a practical method for relative distance estimation, indispensable for leader-follower drone behavior. The research thus pushes the boundaries of autonomous drones by shedding some light on deploying highly complex AI systems on resource-constrained platforms.

1.2 Background and Motivation

The more the airspace has changed its operations, there have been more usages for drones, starting from logistics, surveillance, agriculture, infrastructure inspection, search and rescue, and army operations. The ability of these tools to get into the remotest, most dangerous, and inaccessible environments brings to fore the need to have advanced autonomous navigation methods [9]. Traditionally, navigation systems rely on devices such as GPS, compasses, LiDAR (Light Detection and Ranging), IMUs, and pre-mapped environments, to name a few.

Unlike GPS and compasses systems, that can suffer from signal loss and magnetic interference, sensors like LiDAR and IMU do not depend on external signals. Instead, they measure distance and motion internally, making them less affected by disturbances from the environment. However, these sensors tend to be more expensive and consume a lot of power, which limits their use and deployment in lightweight drones. These limitations could make this approach useless for navigation in dynamic or unknown environments [10]. These drawbacks are pronounced indoors, wherein the altitude of the drone is heavily constrained and the signals stand either weak or completely absent, especially for GPS.

The main motivation for this project is the need for UAVs working autonomously in the real and complex environment that implies real-time perception and decision-making [5]. In particular, autonomous target tracking and following at a safe distance from a moving target i.e., a person, without any constant human intervention, will be a giant step towards full autonomy [11].

1.3 Problem Statement

The task of autonomous human-following by drones poses inherent challenges given the changing illumination conditions, occlusions, the dynamic nature of the target, and intensive computation

for high-accuracy vision processing on resource-scarce drone platforms [5], [6]. Approaches today usually find difficulty in dealing with the scalability and responsiveness requirements for practically deploying these systems in diverse operational scenarios [12]. On the other hand, when the target is a person viewed from different angles, one needs cutting-edge computer vision techniques that perform with reliability under varying environmental conditions [13]. While deep learning approaches can undoubtedly constitute a potential solution to a perception problem [14], the embedding of such computationally intensive models into a drone control system that is real-time, responsive, and scalable stays a great engineering challenge in its own right.

1.4 Thesis Objectives

This thesis, "A Distributed and Scalable Approach for Drone Navigation Based on State-of-the-Art Computer Vision Techniques," attempts the development and verification of fully autonomous drone navigation systems designed for reliable person detection and tracking, relative distance estimation, and autonomous target following while maintaining a safe distance. The main objectives include:

1. Designing and implementing an actual distributed systems architecture based on Docker containers, which decouples video capture from heavy computer vision processing and drone control; this design achieves great scalability, fault tolerance, and flexibility in deployment [15].

2. Development and use of Docker images/modules of the latest Gazebo simulator and ROS2 to provide a standard and reproducible environment for development, simulation, and testing. This environment makes it possible for developers, ideally, to integrate and optimize state-of-the-art computer vision models, particularly YOLOv8 for person detection in real time [16] and ByteTrack (via Supervision) for robust object tracking and ID assignment [17], and to evaluate their real-world suitability in simulations.

3. Establishing a robust inter-module communication setup where ROS topics grant messages for exchange between the drone world/environment containers and controller modules, while message queue solutions (such as RabbitMQ) allow a virtually infinite scalable solution for distributed processing (using docker containers) [18]. This configuration permits flexibility in enabling Docker units to be deployed onto different compute resources that might reside in geographically distant locations or might perhaps be cloud-based, depending upon processing requirements, mostly for CV/YOLO being more compute-intensive.

4. To develop a relative distance control by employing PID controllers given dynamic camera calibration [19] that enables accurate leader/follower behavior.

5. To have a ROS-based control method for autonomous navigation, allowing drones to operate safely and efficiently independently to be simulated or real-world appliances [20].

1.5 Research Methodology

The research procedure in this thesis is characterized by a strictly sequential methodology, typically informed by a Systematic Literature Review (SLR) [21]. Thus, in the first phase, an exhaustive SLR was carried out to delineate the existing state of autonomous drone navigation, computer vision techniques for object detection and tracking, distributed robotics system architecture, and relative methods for distance estimation. An exhaustive study was done through academic databases and renowned conferences for solutions that have been discovered so far, examining their strengths and weaknesses, to offer alternatives. Since the selection of the technologies was directly consequent to the SLR study (YOLOv8, Supervision, Docker, ROS2, RabbitMQ), the SLR also lay in the justification for the distributed architecture and, consequently, the particular objectives of this thesis.

Thus, after the SLR, the focus is on the design and implementation of the distributed system established on modular Docker containers, the integration of computer vision models, and the setup of a robust communication protocol. The next step consisted of experimental validation within a realistic simulation environment with Gazebo and ROS2, assuring iterative testing, parameter tuning, and performance evaluation in somewhat controlled settings. This iterative design, plus simulation-based validation allowed for continuous improvement of different parts of the system, ultimately providing confidence behind the functional correctness and scalability.

1.6 Methodology Overview

The entire methodology of operations is concentrated on the integration of technologies within these compounded containers. Video streams, say, recorded through a camera on a drone, would then have to be efficiently transmitted to another server running Docker for processing purposes [22]. Inter-module communication, as well as the propagation of commands, acts as the infrastructural backbone for orchestrating the distributed components, done through pub/sub message passing to Dockerized services [23]. Python will be the primary language of the entire system; among various libraries, it will use OpenCV and PyTorch, while interfacing with ROS2 to achieve complete drone control as well as inter-container messaging [24]. The validations take place within the Dockerized Gazebo simulation environment [25]. Structurally, these systems were designed for deployment on real hardware, such as physical drones and Raspberry Pi modules, however, these validations never came to materialization due to time constraints and remained a prospect for future work. Still, this Docker-centric core provides an open canvas for future exploration, laying the groundwork for venturing into newer, more complicated environments, such as those with multiple dynamic targets, increased occlusion, and different environmental conditions.

1.7 Contributions of the Thesis

The main contributions of this dissertation can be listed as follows.

1. Designing and empirically validating, through simulation, a modular scalable distributed architecture to navigate drones autonomously. The architecture makes the best use of the features provided by Docker containers and ROS2 to achieve flexibility and good resource management [15].

2. Integrating computer-vision tools at the cutting edge (here, YOLOv8 for detection and ByteTrack/Supervision for tracking) in the distributed environment and performing an evaluation of the performance showing that this computer-vision pipeline is useful for real-time perception of objectives in dynamic environments [7].

3. Developing a practical method for relatively estimating distances, which is the basis for leader-follower drone interaction, by applying PID control based on dynamic camera calibration instead of using traditional methods of depth sensing [26].

4. Gaining the insight from challenges and solutions toward deploying complex AI systems on resource-constrained platforms and setting up a viable simulation environment, which stands at a vantage point for further investigations into complex scenarios [5].

1.8 Thesis Structure

The structure of the thesis is as follows: Chapter 2 undertakes a comprehensive literature review on autonomous drone navigation, computer vision for object detection and tracking, and distributed systems, based on the systematic literature review. Chapter 3 describes the system design, where the architectural elements are described and their modularity and interactions within the Dockerized and ROS2 environment are presented. Chapter 4 considers the experimental setup and simulation results. In the end, Chapter 5 discusses the conclusions and limitations of the current implementation and offers several possibilities for future work.

Chapter 2

Background and Literature Review

This chapter offers an overview of the existing literature about autonomous drone navigation, computer vision for real-time perception, and distributed system architectures. It begins with an exploration of the systematic literature review method used to extract recognized trends, state-of-the-art, and research gaps. This is followed by an overview of the evolution and current status of autonomous drone navigation and then by a thorough study of computer vision techniques for object detection, object tracking, and distance estimation. Finally, the chapter covers distributed computing paradigms and their application within robotic systems, thus demonstrating the motivation for the proposed Docker-based methodology.

2.1 Systematic Literature Review Methodology

A Systematic Literature Review was conducted to establish a comprehensive understanding of the landscape of research relevant to this thesis. Its main aim was to identify, select, and compile the most pertinent academic works in accordance with established methodological guidelines that address the defined research questions [27].

2.1.1 Research Questions

The research questions formulated for the initial SLR were:

1. RQ1: What computer vision algorithms are considered to be the state-of-the-art for the real-time operations of object detection and tracking from drones?
2. RQ2: How are relative distance and pose estimation achieved in the vision-based navigation system of drones, particularly in the leader-follower paradigm?
3. RQ3: What architectural paradigms are more used to deploy heavy perception and control algorithms onboard, or for, autonomous drones?
4. RQ4: What are the benefits that distributed system architectures bring to drone navigation in terms of scalability and robustness and the respective challenges?

5. RQ5: What are the main limitations of GPS-denied environments for drone navigation, and how can those limitations be circumvented by vision-based methods?
6. RQ6: Under varying environmental conditions (lighting, occlusion, clutter), how do computer vision-based systems perform for drone navigation?
7. RQ7: What methods are currently used to reduce computational requirements in real-time visual processing onboard a drone, especially since drones have limited onboard processing capabilities?
8. RQ8: How can simulation environments (e.g., Gazebo, AirSim) be used to effectively support the development and validation of systems for vision-based drone navigation?
9. RQ9: What are the trade-offs between centralized control and decentralized control in systems of drones using vision for navigation?

2.1.2 Search Strategy and Selection Criteria

The search was executed via a multi-database approach involving the major academic search engines like Scopus, ACM Digital Library, and IEEE Xplore. Keywords joined with their explicit or implicit synonyms via Boolean operators were put together for search queries, an example being:

```
("drone" OR "UAV") AND ("autonomous navigation" OR "human following")
("computer vision" OR "object detection" OR "object tracking") AND ("YOLO" OR
    "Supervision" OR "deep learning") AND ("drone" OR "UAV")
("distance estimation" OR "relative pose") AND ("drone" OR "UAV") AND ("camera
    calibration" OR "PID control")
("distributed system" OR "edge computing" OR "cloud robotics") AND ("drone" OR
    "UAV") AND ("ROS" OR "Docker" OR "RabbitMQ")
```

Listing 2.1: Boolean search queries used in the SLR

From initial screening, studies were filtered through title and abstract review for relevancy. Full-text articles were then retrieved and filtered by exclusion criteria, consisting of journal articles, conference papers, and relevant dissertations of the last five to ten years (2015-2025). We have excluded non-English articles, short papers, and other studies not directly concerned with autonomous drone navigation or relevant computer vision/distributed system aspects.

2.1.3 Data Extraction and Synthesis

The data from selected studies were systematically extracted, including publication year, authors, methodology, key findings, technologies implemented, performance measures, and limitations noted. This data was synthesized along thematic lines. The synthesis process further brought about research gaps and areas of further inquiry, feeding straight into the aims of this thesis.

2.2 Autonomous Drone Navigation: Evolution and Challenges

The field of autonomous drone navigation has seen massive growth. Initially, the majority of systems undertook simple landmark-based navigation, while now many systems are capable of the evolution of adaptive behaviors in dynamic environments. Early systems relied mainly on the GPS for localization and navigation [3]. While they were very good in the open outdoor environment, GPS-based systems pose various limitations. For example, in signal-denied areas such as indoors, canyons formed by tall buildings in the city, etc., GPS-dependent systems become useless and produce inaccurated results [3]. And so were introduced SLAM or Simultaneous Localization and Mapping techniques which enable the drones to map the environment on one hand and locate themselves in those maps on the other hand, usually by either visual, LiDAR, or inertial sensors [28]. Figure 2.1 shows how the investigation in the field of UAVs has evolved in the last few years, demonstrating the relevance of the area and applications.

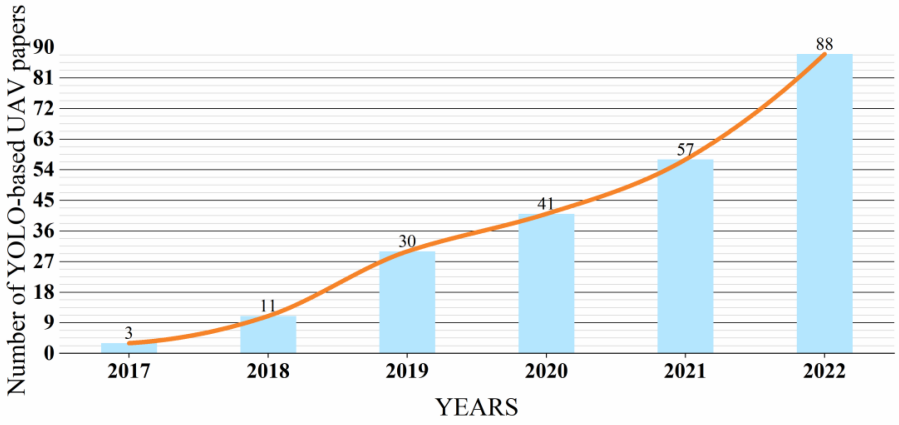

Figure 2.1: Evolution of published articles on UAV research according to [29]

This dynamic navigation environment poses many challenges, especially in path setting for moving targets. Some of these are:

- Real-time and precise perception: A split-second, accurate appreciation of all moves in the environment, including object identification, tracking, and motion prediction.
- Robustness Against Variability in Environment: The changes with which systems would be degraded, including illumination, weather, occlusions, and background clutter, amongst others.
- Computational Constraints: Given that the drone faces limitations with onboard processing, efficient algorithms are a must, so as to consider reducing the computational offload.
- Safety and Collision Avoidance: Ensuring that the drone remains safe to humans and obstacles in its neighborhood, requiring very precise control and prediction skills.

With the growth of drone technology appearing to slow, the increased demand for autonomy will be sufficient to maintain a global market for drones at 50 billion dollars by 2030 [30]. Growing

interest in autonomous navigation, especially in real-world scenarios outside controlled settings, poses a need for solutions that are robust and scalable.

2.3 Computer Vision for Drone Perception

Computer vision constitutes an essential pillar with several enabling functions, equipping drones to perceive, interpret, and interact on an autonomous basis. In this respect, drones receiving intelligent instructions and behavior changes in real time from image reconnaissance may be required when an external positioning system like GPS is either unavailable or unreliable. Thus, in this section, some of the techniques that constitute basic concepts of the field will be reviewed, including object detection, tracking, and relative distance estimation, which form the mechanism whereby a human-following system is implemented in this dissertation so that the drone has mechanisms for locating, identifying, and maintaining the right distance to a moving target with respect to its formation.

2.3.1 Object Detection

Object detection: is the identification and localization of instances objects of a certain class — for example, a person or vehicle — in an image or video stream. Traditionally, such methods have included the use of hand-crafted features and typical machine learning, often with domain knowledge and tuning. With the growth of deep learning and, more importantly, Convolutional Neural Networks (CNNs), there have been improvements in detection on two fronts: accuracy and speed [14].

You Only Look Once (YOLO): is considered by many as the most revolutionary method since the introduction of deep learning for object detection. Unlike region-proposal-based methods, which divide the image up into a multitude of sections and analyze each individually, YOLO takes the image in one shot and simultaneously detects objects in it. This synoptic structure gives YOLO the advantage of operating in real-time.

YOLOv8: is the most used and stable version at the present time of writing. Latest generation improvements in the neural network architecture of YOLOv8 as well as in the loss functions and training methods have given it a higher degree of accuracy and a faster inference speed than all versions that preceded it [16]. The benchmark results suggest that an mAP greater than 50% can be achieved on the COCO dataset using YOLOv8 while maintaining an inference speed of more than 100 FPS on modern GPUs [31]. In comparison to the detection accuracy and inference of the main YOLO variants, Table 2.1 provides a summary of the same.

Table 2.1: Performance of YOLO Variants for Object Detection [31]

Model	**mAP@0.5: 0.95 (COCO)**	**Inference Speed (FPS, V100 GPU)**	**Key Features**
YOLOv3	33.0	~20	Anchor boxes, multi-scale detection
YOLOv4	43.5	~50	CSPDarknet53, PANet, Mish activation
YOLOv5	55.8	~200	AutoAnchor, augmentations and improved grid sensitivity
YOLOv8	63.0	~280	Anchor-free, decoupled head, improved backbone
YOLO-NAS	52.2	110	Neural Architecture Search optimized

More recent versions of the YOLO family like YOLOv11 are often considered experimental since there is not widespread documentation, adoption, or compatibility, also requiring more computational power for training and inference. Given the real-time constraints and limited computational power available on embedded platforms such as the Raspberry Pi or low-end CPUs, YOLOv8 was chosen as a balance between performance, maturity, and feasibility of deployment.

Due to the implications for human-following mechanisms using drones, YOLO becomes a promising alternative for real-time applications. Provided the concept can be off-board on an edge server, it maintains reliability in its detection with respect to latency. This is why the present work chooses YOLO as its main detection method. Figure 2.2 shows the state-of-the-art landscape in vision-based object detection, which paves the way for the realization of real-time perception capabilities on embedded or distributed platforms via algorithms such as YOLO [32].

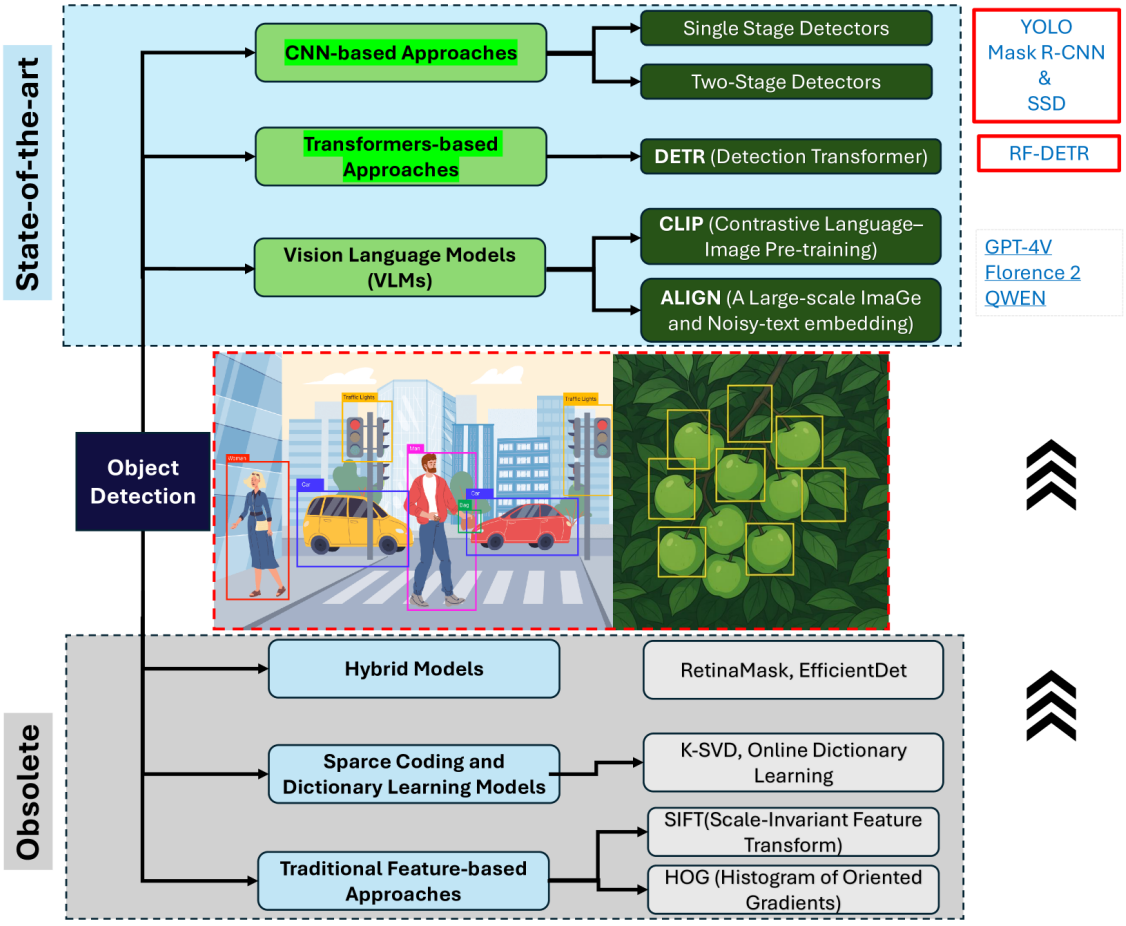

Figure 2.2: State of the Art in Object Detection as per [32]

2.3.2 Object Tracking

While detecting objects identifies objects in each frame, **object tracking** corresponds to the identity of the object in a sequence of frames. This is essential for determining the trajectory of the object-adjacent future position-or interaction along the way, such as following a human.

Some of the methods commonly used for object tracking include:

- **Detection-based Tracking (Tracking-by-Detection):** This paradigm consists first of detecting objects in each frame and then associating detections over time. This approach bets on the accuracy of current detectors.
  - **Supervision** exemplifies a tracking-by-detection paradigm that uniquely tries to assimilate detections with low confidence. It thus keeps both high-score and low-score detections and tries to associate the low-score detections with existing tracks so that the association becomes more robust against occlusion or motion blur [17]. Its underlying technique (ByteTrack) has exhibited notable MOTA (Multiple Object Tracking Accuracy) improvements over previous methods such as DeepSORT, thereby setting new records on a number of benchmarks. Table 2.2 presents a general comparative

overview for the major object-tracking algorithms, emphasizing their core principles, occlusion treatment, and real-time applicability.

Table 2.2: Comparison of Object Tracking Algorithms [33]

Algorithm	**Core Principle**	**Handling Occlusions**	**ID Switches (Lower is Better)**	**Real-time Performance**	**Complexity**
Supervision (ByteTrack)	Associates all detections (high & low score)	Uses low-confidence detections to bridge gaps	Very Low	Excellent	Moderate
DeepSORT	Extends SORT with deep appearance features	Relies on appearance similarity, struggles with heavy/prolonged occlusion	Moderate	Good	High (due to NN)
SORT	Simple IOU-based association	Prone to ID switches during occlusion	High	Excellent	Low

- **Filter-based Tracking:** These sort of methods use probabilistic filters, such as the Kalman Filter or Particle Filter, to guess the future state of an object based on the history of motion of the object. After observing the object, the model revises its estimates, thus creating a buffer to noise and uncertainty for dynamic environments. It is mainly useful for tracking continuously moving objects.
- **Correlation Filter-based Tracking:** These algorithms learn a discriminative correlation filter which is applied directly onto the image regions for the localization of the object of interest. These algorithms have a reputation for being very fast and performing well in real-time scenarios with at least partial occlusions or moderate appearance changes.

For drone applications, combining the already powerful YOLO detector with a tracker such as Supervision/ByteTrack provides a strong synergy, which allows for more reliable and continuous target tracking.

2.3.3 Relative Distance Estimation

Concerning relative distance, the drone has to estimate several distances so that it keeps at a safe but constant distance following its target. Under static environments, distances may be pre-mapped, while in dynamic human-following, it has been required to infer distances on the fly.

Table 2.3: Distance Estimation Methods [34], [35]

Method	**Principle**	**Advantages**	**Disadvantages**	**Suitability for Drone Following**
Stereo Vision	Triangulation from two cameras	High accuracy, direct depth map	Computational cost, calibration sensitive, hardware weight	Good, but complex for small drones
Lidar	Time-of-flight measurements	Very accurate, robust to lighting	High cost, weight, limited range/FOV	Excellent, but costly/heavy
Monocular Depth (DL-based)	Deep learning inference from single image	Low hardware cost, real-time potential	Generalization issues, accuracy varies with scene	Promising, but can lack robustness
Homography + Bounding Box Scaling (This Thesis)	Relates object size in image to real-world size via camera parameters	Cost-effective, simple, uses existing detection	Requires known object size or calibration, sensitive to camera pose changes	High, with proper calibration and PID control

This thesis follows the approach of dynamic camera calibration and PID (Proportional-Integral-Derivative) control for regulate the relative distance between the drone and the target. By obtaining the camera's point of view and knowing the dimensions of the target or a calibrated reference, the observed bounding box size and position can be associated with the real distance to the object.

- **Camera Calibration:** Dynamic calibration is of prime importance in aerial applications so that it can compensate for any changes arising due to vibrations or tiny relative shifts in the mounting of a camera during flight, thereby holding perception accuracy over time [36].
- **PID (Proportional-Integral-Derivative) Controllers:** PID controllers are used widely in robotics and provide a simple and robust way to control the system behavior. In this case, they are used to modify the drone position or velocity as a function of the error between the desired following distance (the distance measured from the calibrated bounding box size) and the distance estimated by the perception module. This guarantees that the feedback loop will be stable and fast enough to maintain an appropriate distance between the drone and the moving target [19]. See Figure 2.3 that presents the feedback-based structure for the PID controller used for drone distance control. In this figure, the error signal (that is, the difference between the desired distance and the measured distance) is processed through the Proportional (P), Integral (I), and Derivative (D) blocks. This control output actuates the quadcopter. The continuous update through the feedback loop maintains the desired distance.

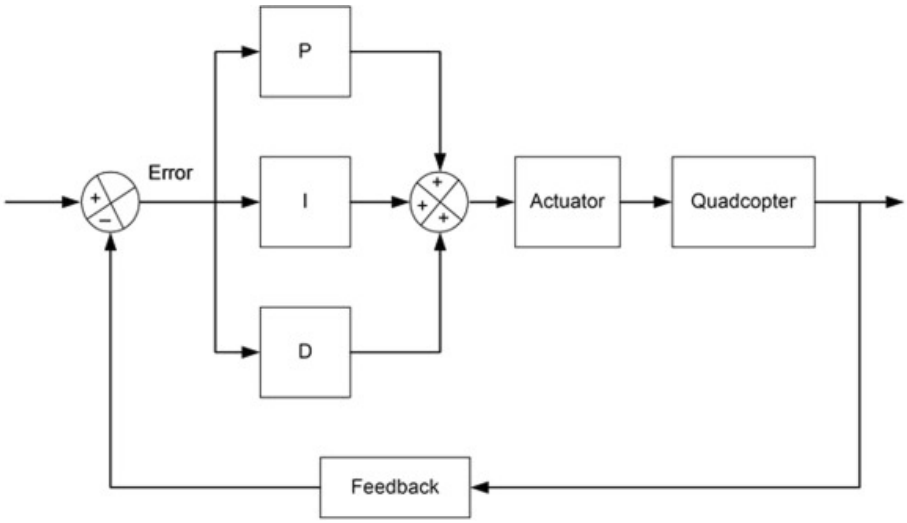

Figure 2.3: PID control loop for distance regulation in a drone system.

2.3.4 Related Work on Vision-Based UAV Autonomous Navigation

There is a growing reliance on autonomous aerial systems to perform tasks that used to be manual, risky, or logistically difficult. A field that has perhaps received much attention in both early and recent research is the use of fiducial markers (ancestral artificial visual patterns placed on identified surfaces) - as used in [37] - toward aided methods of autonomous drone-based localization and landing. These markers enhance the accuracy of reference points, which drastically ease the determination of a relative position and orientation of the drone, especially in environments lacking GPS, such as indoors or under dense vegetation cover. They have been widely used because of their ease of setup and accuracy under controlled conditions in prototyping, academic experiments, and even some industrial environments.

In the MSc dissertation, "Integrated Vision-Based Navigation and Sliding Mode Control for Robust Autonomous Quadcopter Landing," Bruno Daniel Pereira da Silva describes the development of a robust ArUco-marker detection algorithm for precision landing based on vision [37]. The deterministic nature of the marker was exploited in order to position the vehicle with very high accuracy, especially in an artificial landing scenario. Most of the validation for the system was done in a Gazebo simulation environment, with an implementation of PX4-Autopilot integrated with ROS2 for control and data exchange.

While marker-based methods such as the one developed by Silva have proven successful in specific applications, for instance, in precision landing on a predefined pad, the need for a predetermined visual pattern considerably restricts their applicability in dynamic, unconstrained environments where the target may not be displaying such markers.

Marker-dependent approaches are rapidly becoming niche and nearing obsolescence for a majority of general navigation tasks. In contrast, the advantages of modern vision techniques such as those using YOLO lie in detecting and tracking arbitrary objects (e.g., a person walking from behind) without any prior setup or environmental modifications, thus offering unmatched levels of

flexibility, scalability, and robustness across a diverse set of operational conditions.

2.4 Distributed Systems for Autonomous Drones

In many practical applications, the computational load involved in real-time computer vision and highly complex control algorithms is often beyond the capacities of small-sized light drones in their onboard processing. This forces the adoption of distributed system architectures, where challenging computations are transferred to more powerful computation engines. In addition to reducing the problems caused by low-quality hardware, it also greatly improves the system's scalability, flexibility, and robustness.

2.4.1 Rationale for Distributed Processing

The motivation for adopting a distributed approach in drone navigation is indeed manifold:

- **Computational Offloading:** Heavy video processing, deep learning inferences, such as YOLOv8, and complex path planning may happen at the station, server in the cloud, or a better edge device. Hence, the onboard processors are freed to perform crucial flight control [38].
- **Scalability:** In the distributed setting, scaling means including more computing nodes to the environment or distributing the algorithms into more modules and nodes. New servers can just be plugged in, as per how many drones are desired to be enabled or for the occurrence of some latest vision tasks- without any stoppage.
- **Modularity and Flexibility:** Breaking down each component into modules (Docker containers) will facilitate independent development, testing, and deployment of those modules, and thus they can be updated or replaced without affecting the rest of the larger system.
- **Fault-Tolerance:** In an auditing perspective, if one of the units, either computational or otherwise, becomes faulty or unusable, then the task could be redirected to another unit, or the backup could be kicked into handling-it's a clear advantage for building resilience.
- **Resource Optimization:** In a distributed architecture, hardware resources (for instance: GPUs for AI inference or CPUs for control tasks) can be deployed strategically to maximize their effectiveness. This allows for the best possible usage of computational resources, reduces the amount of resources that are not used, and makes it sure that every node is doing the work for which it is most qualified. Therefore, the total efficiency and performance of the system are enhanced.

.

2.4.2 Key Technologies in Distributed UAV Systems

Some technologies are essential for constructing distributed drone systems:

- **Docker:** Docker is an application development platform inside a **container**, where building, shipping, and running applications happen. Docker containers essentially package an application with its dependency into something that would run faithfully in any supporting environment. In the robotics realm, Docker provides a very light and portable reproducible environment for deploying nodes or any other part of the robotic applications [15], [38]. Being able to create **Docker images/modules** with a pre-configured environment (for example, Gazebo, ROS2) greatly simplifies development and deployment.

- **Robot Operating System (ROS / ROS2):** A truly customizable tool acting as middleware to write robot software. It gives libraries and tools that aid software developers to create robot applications. The successor to ROS, ROS2, brought real-time capabilities, improving multi-robot support, and having a robust communication backbone-best suited for distributed systems where different components must communicate using **ROS topics** [39], even if the nodes are set in different Docker containers or distributed across different machines. ROS topics are the most common means of exchanging messages; they smoothly transmit data (camera feeds, recognition results, control commands) between nodes loosely coupled with each other.

- **RabbitMQ:** An Open source application working according to AMQP standards enabling asynchronous communication between distributed applications. It can be used within the modules to send custom messages and distribute data processing between the Dockerized modules, offering a strongly backed and fault-tolerant message backbone. This publish/subscribe model allows the decoupling of components that communicate without being aware of each other.

Furthering the selection of the appropriate mechanism of communication for different distributed drone situations, Table 2.4 presents a comparison of these popularly used protocols - ROS topics and RabbitMQ. They are classified depending on their communication paradigms, merits, latency, reliability, and complexity, giving a comparative assessment that helps in making an informed decision on their use case.

Figure 2.4 presents the conceptual architecture of a distributed drone system based on Docker containers, illustrating the separation between the onboard autopilot and the backend cloud infrastructure. This separation allows computationally intensive tasks, such as computer vision inference, to be processed remotely, while the drone maintains a lightweight embedded system focused on flight. This architecture serves as the basis for the modularity and scalability of the system developed in this work, facilitating the distribution of components across different geographic locations or cloud environments, as illustrated in the figure adapted from [42].

By combining these applications, a modular and scalable pipeline can be built wherein computationally intensive processes such as CV inference with YOLOv8 can be executed on strong

Table 2.4: Distributed Communication Protocols for Robotics [40], [41]

Protocol / Broker	**Paradigm**	**Key Feature / Advantage**	**Typical Use-Case in Robotics**	**Latency**	**Reliability**	**Complexity**
ROS Topics	Publish / Subscribe	Real-time, low-latency, integrated with ROS ecosystem	Onboard inter-process communication, tightly coupled modules	Very Low	High	Moderate
RabbitMQ	Message Broker	Advanced queuing, message persistence, robust routing	Asynchronous command and control, logging, data offloading	Moderate	Very High	Moderate

remote servers, thereby theoretically keeping the drone light for efficient flight. Such a distributed version of the system allows it to dynamically adapt to changing workload schedules by allowing the CV/YOLO stage to be run on processing units ever distant or in the cloud, resulting in better performance and resource optimization activities. Because of such flexibility, a solid ground is laid for scaling the system into more complex scenarios, like multi-drone coordination, multi-target tracking, or the fast-paced environment with massive occlusion.

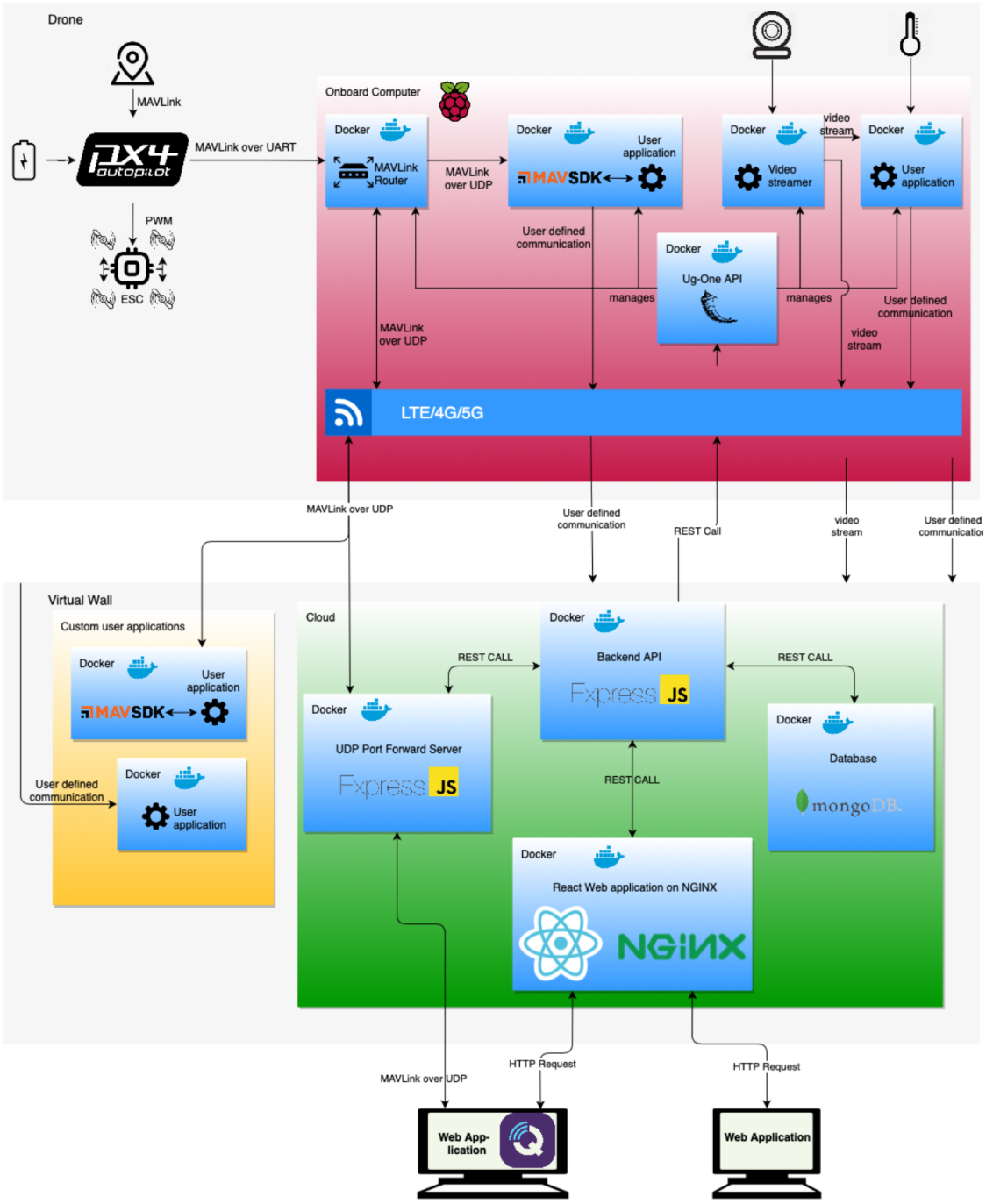

Fig. 1. UG-One system components overview

Figure 2.4: Conceptual distributed drone system architecture with Docker [42]

Chapter 3

System Design and Architecture

This chapter illustrates the architecture of the proposed system for autonomous drone navigation. A detailed overview is offered about the modular components and their interconnections on the technology stack that enables perception in real-time, intelligent control, and communication. The architecture itself is consciously designed with the distributed infrastructure in view, ensuring that the infrastructure maximizes scalability, flexibility, and robustness, which are all deemed requirements for advanced-level autonomous drone operations [9].

3.1 Overall System Architecture

Built with a modular and decoupled architecture in mind, the system operates distributing the computing loads to separate processing units. Consequently, from their inherent resource-limited drone platform, we have selected intensive tasks to be executed on remote computers, which could be also running in virtual machines and Cloud. Being a class of architecture, it not only solves hardware constraints but also builds a scalable and flexible autonomous system [43].

In this way each functional block is wrapped into a **Docker container**. Containerization is the key to portability: ensuring the possibility of reproducing the code in different developing and deploying environments, while also easing the issues related with complex software dependencies [44]. In this scenario, communication is crucial and done mainly via **ROS2 topics** that deal with high-frequency control messages and raw data streams in real-time. This could be complimented by **RabbitMQ**, the advanced message broker that enables asynchronous communication and robust orchestration of less time-critical information and commands [45].

Figure 3.1 presents the overall architecture of the system developed in this work, depicting the interactions among the different Docker containers that encompass the algorithms, navigation controllers, simulations and communication modules.

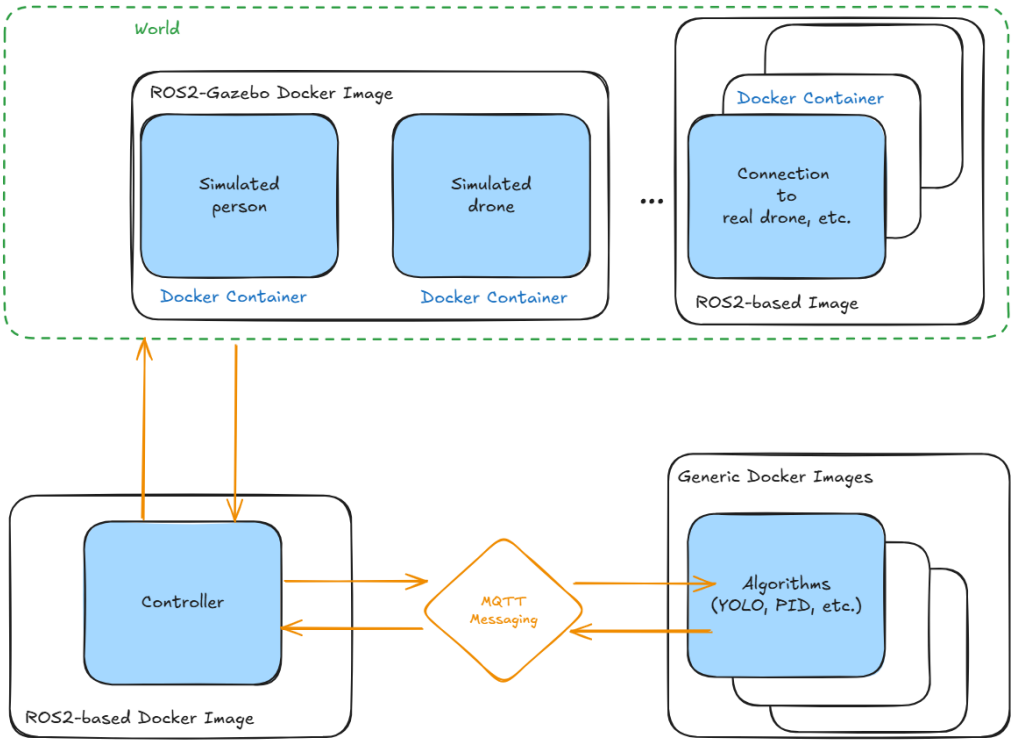

Figure 3.1: Distributed System Architecture for Drone Navigation

The main functional blocks involved, as shown in Figure 3.1, are:

1. **ROS2-based Image:** Docker container responsible for connections to the real/simulated drones and other actors of the world. Based on this image we can have multiple different modules that use ROS2 as message communication mechanism, such as the Controller module, which controls the drone and can use MQTT messaging client to facilitate asynchronous data exchange.

2. **ROS2-Gazebo Image:** Docker containers that allow the introduction of simulated models into the environment, allowing a hybrid world where real and simulated actors can co-exist.

3. **Generic Docker Images:** Docker containers which host the core algorithms such as YOLO for vision processing, PID controllers for navigation, etc.

4. **Generic Messaging Queue Server:** Manages the communication between modules using RabbitMQ/MQTT for message passing and synchronization.

This intended modularity offers several advantages, dynamic scaling being the major one: the scaling of computation resources toward particular jobs. The Application of Vision Processing demands intensive GPU acceleration and is thus deployed on a high-end server, while the Drone Platform Module is meant to be kept light and hence deprived of heavy computational load to enhance agility in flight.

In the system, every component is packaged into a Docker container that is meant to be considered self-contained and independent. This scenario follows microservices architecture where systems maintain loose coupling and are independently deployable to become more modular and easier to make changes or maintenance without requiring the entire system to be shut down.

3.2 Simulated Environment Entities

On our quest to build a hybrid environment where multiple elements can co-exist whether they are simulated or real hardware, we can start depicting the simulated entities. A proper simulation model is usually a must for system building and training, which also helps validating the approach without the burnden of the real hardware.

A virtual environment allowed us to iterate on the algorithms to improve them; to test their abilities in controlled situations; and to carry out experiments under safe conditions before implementation in the field.

By using **Gazebo** acting as the physics simulator and **ROS2** as the robotic middleware, we have ensured the system architecture work very similar to the real world, with all the appropriate physical characteristics, such as gravity, inertia, friction, lighting and shadows; and integration with real hardware becomes only a matter of technical implementation.

For consistency, easy reproducibility, and straightforward setup from one developer workstation to another, a **Docker image/module** for the simulation environment was established. It contains the containerised environment comprising the following necessary components:

- **Gazebo Simulation Environment:** It develops a simulated 3D world with high-fidelity physical characteristics from the real world and a high-fidelity drone model, such as the PX4 quadcopter simulation [46], as seen in Figure 3.2. In Figure 3.3 we see a representation of the drone perspective throught a virtual camera implementation, designed to precisely replicate the features of the actual camera module usually put onboard the drone.

  Initially, the environment had a moving person model to act as the target along with realistic ambiance and obstacles (as in Figure 3.4. But mainly due to limitations in computation and graphics power of existing hardware, it was therefore decided that the environment and the models (of drone, person, obstacles, walls and environmental colors) were kept as simple as possible to ensure the feasibility of the project and undertake all the tests required.

  While the world was reduced to the bare minimum in terms of simulation - as seen in Figure 3.4, removing complex indoor environments and substituting a PX4 model with a simple drone, all of the important real-world physics properties are still intact: inertia, gravity, friction, lighting, etc. Also, the new simplified drone model still had retained realistic quadcopter flight dynamics such as four motor dynamics capable of full direct movements along x, y, z axes and directional rotations consisting of yaw, pitch, and roll.

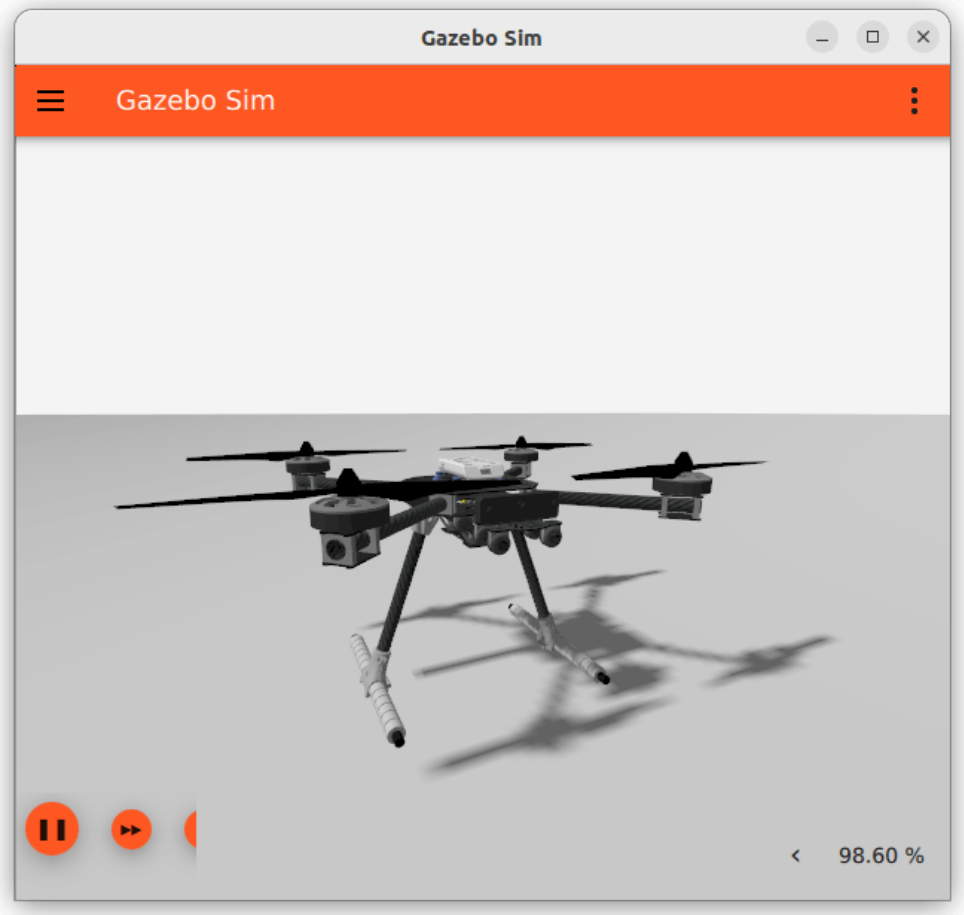

Figure 3.2: Realistic Drone Model in Gazebo Simulator

That way, the Dockerized Simulation Environment developed with all the necessary components yielding greater realism of the real world and objects to be simulated offers some distinct advantages such as:

- **Reproducible Testing:** It guarantees that anyone in research or development can carry out the simulation environment with the same dependencies and configurations, giving the possibility of confirming and replicating experimental results.
- **Scalable Development and Testing:** Many instances of the simulation can start to carry out mass testing of algorithms in various situations without actually involving many physical drones.
- **Safe Experimentation and Rapid-Prototyping:** New algorithms and control strategies might be iteratively developed and tested in a safe environment, offering the possibility for fast prototyping, which may reduce the risks to the hardware.

The Gazebo simulated drone publishes virtual camera feeds that approximate those that would be generated by a genuine camera, and its virtual flight controller consumes commands with ROS2

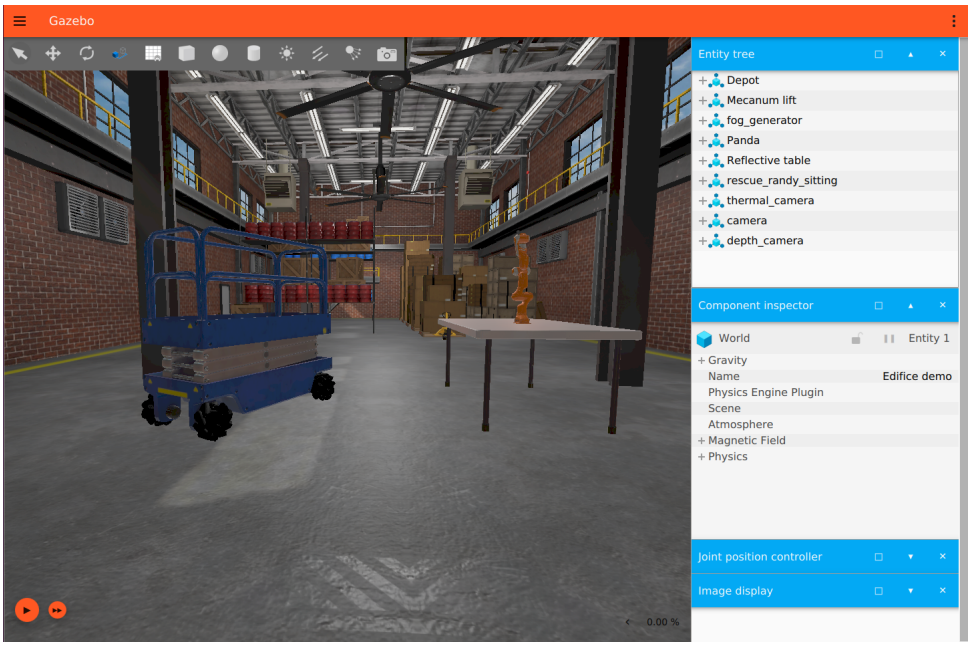

Figure 3.3: Image of the simulated camera mounted on the drone

topics, interfacing with the Vision Processing and Navigation/Control Modules just as a real drone would.

3.3 Real Hardware Modules

The proposed hardware consisted of the Raspberry Pi on top of the drone, which could use a flight controller running open-source firmware such as PX4 or ArduPilot where the communication is offered through the MAVLink protocol [15]. This Module was thought to conduct basic onboard operations:

- **Video Capture:** responsible to handle camera video frames in real time (e.g. using Raspberry Pi Camera Module v3) with dimensions and frame rates requested by the caller for further processing [47].
- **Telemetry Data Acquisition:** intended to receive live flight data such as IMU and GPS (if available) from the flight controller or real sensors.
- **Command Execution**: designed to receive commands from the ROS2 Controller module for the intended navigation are converted into flight controller inputs, e.g., desired altitudes, velocities, or positions.

While this module was designed to use minimal onboard resources, the constraints by both temporal limitations and the inherent difficulties in securing access to physical hardware within a suitably safe and controlled environment, has predominantly focused on a simulated operational

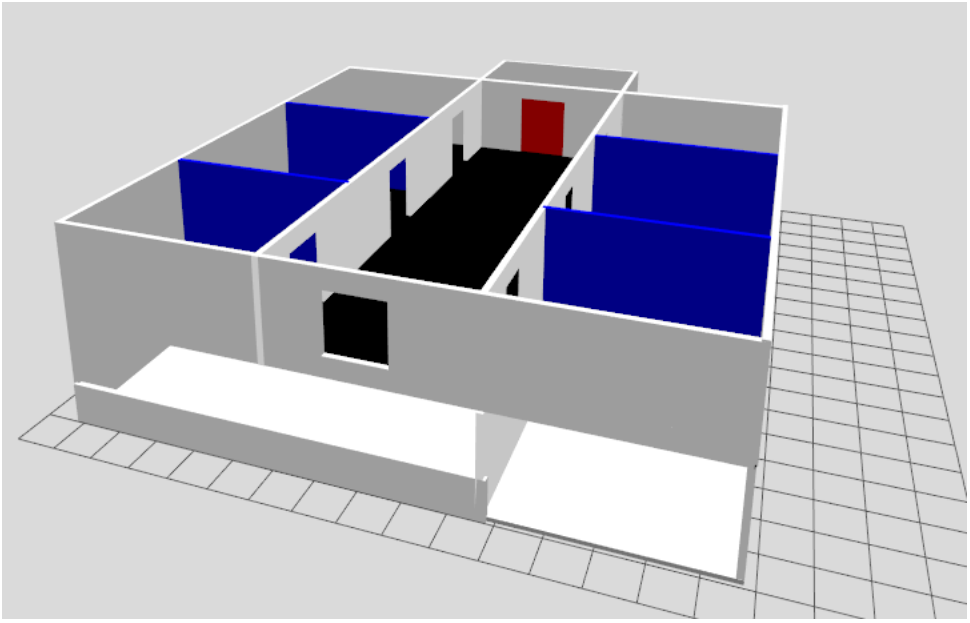

Figure 3.4: Full and complete scenario on Gazebo simulator

setting. This approach allowed for iterative testing and refinement of the proposed methodologies without the logistical complexities and safety protocols necessitated by real-world drone deployment. Consequently, all experimental validation and performance evaluations have been conducted exclusively within this simulated framework.

Whereas this module was intended to use the least onboard resources, it is temporally limited and because of the inherent difficulties of gaining access to real hardware under a suitably safe and controlled environment, has centered mainly in the simulated operational setup. This has made it possible for the iterative testing and refinement of proposed methodologies to proceed without the logistic complications and safety protocols associated with actual drone deployment. Thus, there has not been any experimental validation and performance evaluation performed outside this simulated domain.

3.4 Vision Processing Module

The Vision Processing Module, primarily developed in Python within a specialized Docker container, carries out real-time object detection and tracking. Being a computationally expensive component, this module is mostly implemented in some ground-based server or in a cloud server with GPU accelerations, it mainly includes:

- **Object Detection using YOLOv8:** Video frames are obtained from the drone camera through a ROS2-based communication topic and the message is converted to a readable image as in the code below

```
cv_image = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')
```

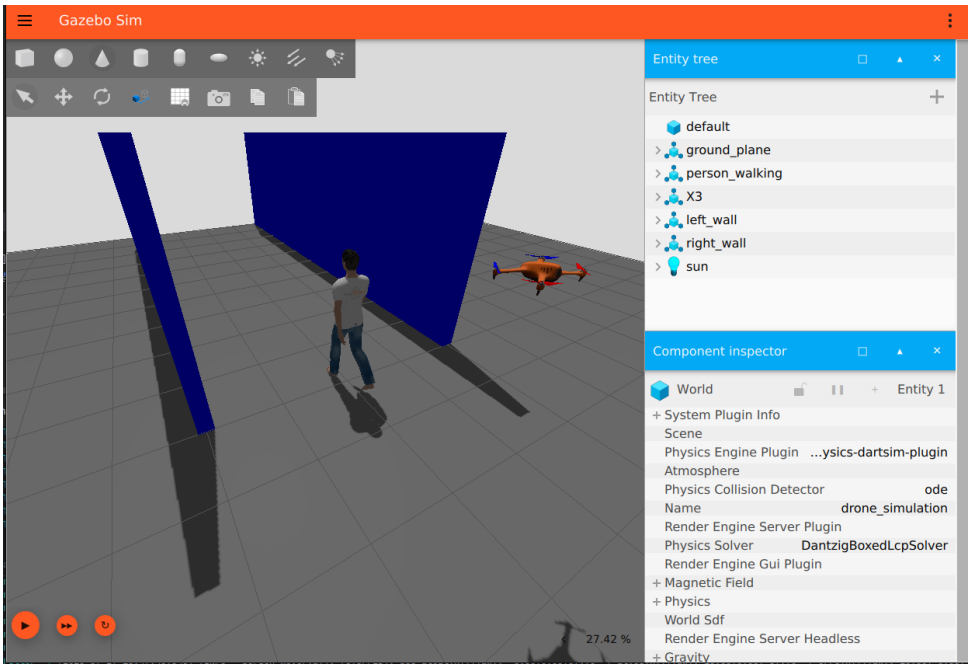

Figure 3.5: Simplified scenario on Gazebo simulator

The `cv2` function of `OpenCV` was used to grab frames and pass the image through the detection algorithm to identify persons in real-time through a GPU-optimized *YOLOv8* model [31] done by resizing, normalizing, and, at times, batching frames for effective GPU utilization.

YOLOv8 Inference Pipeline:

1. **Model Loading**: Pre-trained YOLOv8 model weights are loaded through the `ultralytics` library (e.g., weights `yolov8n.pt` or `yolov8s.pt` for a good mix of speed and accuracy). The model is set up to run on the GPU if one is available, to allow for faster inference.
2. **Preprocessing**: Each incoming video frame is resized to the input resolution expected by YOLOv8, e.g., 640×640 pixels, and normalized. This step is performed efficiently with OpenCV functions.
3. **Inferencing**: The preprocessed frame passes through the YOLOv8 network. This step produces raw detection outputs, which consist of bounding box coordinates, confidence scores, and class predictions.
4. **Post-processing**: Non-Maximum Suppression (NMS) is then applied to eliminate overlapping bounding boxes and create the final list of detected objects. Detections with confidence scores less than a certain threshold (say, 0.6) and which are not 'person' class are disregarded.

- **Object Tracking using ByteTrack:** The following piece is the tracking module, The detected targets are sent to the Supervision/ByteTrack algorithm to generate consistent unique IDs over subsequent frames. Things like temporary occlusion and motion blur confuse most trackers in maintaining a continuous identity and trajectory awareness of those targets. These features smooth the tracking process much better than simpler prevailing IoU-based tracking algorithms, a particular need in an dynamic environment [17].

  Extensive use was made of Roboflow's `supervision` library to exploit its powerful tracking abilities and efficient visualization mechanisms. These include those that allow bounding boxes, labels, and tracking IDs to be drawn directly onto video frames for real-time debugging and qualitative analysis, which greatly facilitated both the development and the verification of the vision pipeline. Figure 3.6 shows the result of the YOLO detection with the emphasis on the drawed bounding box lines over the person.

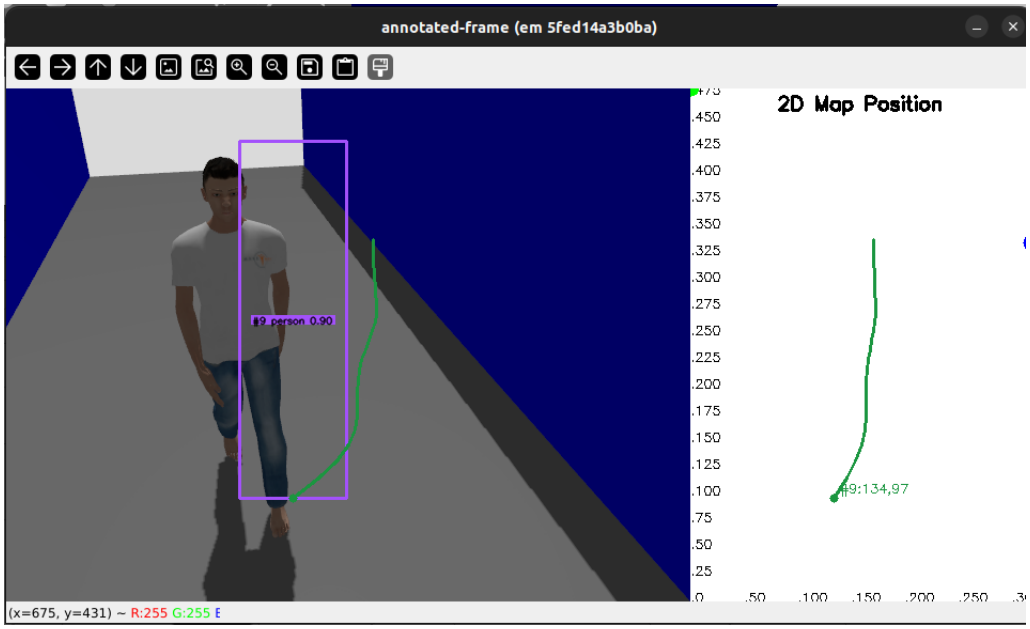

Figure 3.6: Yolo detection from drone camera perspective (bounding boxes)

- **Publishing Detection Data:** The resulting detection and tracked information consists of bounding box coordinates (e.g., pixel locations, width, height), class labels (e.g., person), unique tracking IDs, etc., all formatted as structured ROS2 messages and published on the defined topics (e.g., `/vision/detections`, `/vision/tracked_objects`).

  The data structure used to represent the full set of information processed by the vision module can be depicted as below:

```
detections_dict[f"object_{tracker_id}"] = {
    "tracker_id": int(tracker_id),
    "class_name": detections.data.get("class_name", [])[i],
    "class_id": int(detections.class_id[i]),
```

```
            "confidence": round(float(detections.confidence[i]), 2),
            "xyxy_bbox": [... detections.xyxy[i].tolist() ...],
            "movement": {
                "x1": round(...),
                "y1": round(...),
                "x0": round(...),
                "y0": round(...),
                "xy_direction": [...]
            }
        }
```

The Docker container of this module is prepared with all deep learning dependencies, including PyTorch, CUDA Toolkit, and the YOLOv8 models and weights, to guarantee that the execution environment is standardized, optimized, and reproducible (meaning it works the same everywhere) and to ensure that the inference is GPU-accelerated, essential in real time applications, as in drone systems.

3.5 Navigation Module

The **Navigation-and-Control Module** acts as the intelligent center that receives the output of visual perception and so generates the drone commands to realize the leader-follower autonomous behaviors.

- **Target State Estimation:** The module subscribes to ROS2 topics where tracked object information is published by the Vision Processing Module. The module estimates relative distance to the tracked target and relative horizontal shift of this target within the drone field-of-view by invoking the principles of **dynamic camera calibration** [11], which includes inferring from observed bounding box dimensions (e.g., pixel height/width) to real-world distances relating to the current pose of drone and camera parameters.
- **Relative Distance and Pose Estimation**:
    1. **Camera Calibration Parameters**: Pre-calibrated camera intrinsic parameters (focal length, principal point) and distortion coefficients are loaded into the program. These parameters are essential for important geometric calculations [48].
    2. **Position Estimation Using Homographies**: Assuming the target person to be on the ground plane, homography principles are applied to relate the bases of the target bounding box (position of feet in the image plane) to the ground plane position as projected with respect to the drone. This includes considering of extrinsic parameters

of the drone (height, pitch, and roll), and the fixed mounting angle of the camera with respect to the drone.

3. **Scaling the Bounding Box for Distance**: From a rear view, either the bounding box height or width is used for distance estimation. Given the focal length of the camera, f, and an estimate of the actual height of the target, H_{real}, the distance D can be approximated by $D = (H_{real} \cdot f)/H_{pixel}$, where H_{pixel} is the bounding box height in pixels. This may be further fine-tuned depending on where in the frame the person appears, as a correction for perspective distortion is necessary [49].
4. **Horizontal Offset**: The horizontal offset of the target from the image center is calculated in meters and is thus made the basis for commanding the drone to center the target in its field of view.

- **Implementing PID control:** The estimation of relative distance and offset is fed back into **Proportional-Integral-Derivative (PID) controllers**, which then yield commands on linear and angular velocities (forward/backward, left/right, up/down, yaw rate) to minimize the error between desired following distances or offset and the current estimated values. The parameters of PID are well tuned for stable, smooth, and responsive drone movements [6].

  Three independent PID controllers are written in Python to control the drone movements along the X (forward/backward), Y (left/right), and Z (up/down) axes or control its velocity components.

  - **Depth Control (X-axis):** A PID controller sets the forward/backward velocity according to the error between the desired following distance (e.g., 5 meters) and the estimated present distance to the target.
  - **Lateral Control (Y-axis):** Another PID-controller sets the left/right velocity based on the horizontal offset error, attempting to keep the target centered in the frame.
  - **Vertical Control (Z-axis):** A PID controller sets the up/down velocity to either hold the target at a desired vertical position in the frame or at a desired altitude with respect to the estimated base of the target.

  The tuning of the PID parameters (K_p, K_i, K_d) is a critical and iterative process, done initially in simulation and thus further refined in actual tests so that the controller remains stable, responsive, and free from oscillations.

- **Publishing Commands:** These calculated drone control commands (e.g., `geometry_msgs/Twist` messages) are published as ROS2 topics (e.g., `/drone/cmd_vel`) to be consumed by the Drone Platform Module (via the Communication Hub).

This module closes the control loop, enabling the drone to follow tracked individuals with basic autonomy.

3.6 Data Flow and Communication Protocols

Figure 3.7 shows the logical structure of the data flow on the project. Firts, the camera mounted on the drone generates images and they are sent to the controller, which has a subscriber for the image topic. This subscriber is responsible for sending it to the YOLO module, and subsequently the aforementioned processed data is sent back to the controller so it can be published back to the ROS2 topics. The controller acts then as a two-way bridge between the robotic/real world and the algorithms environment.

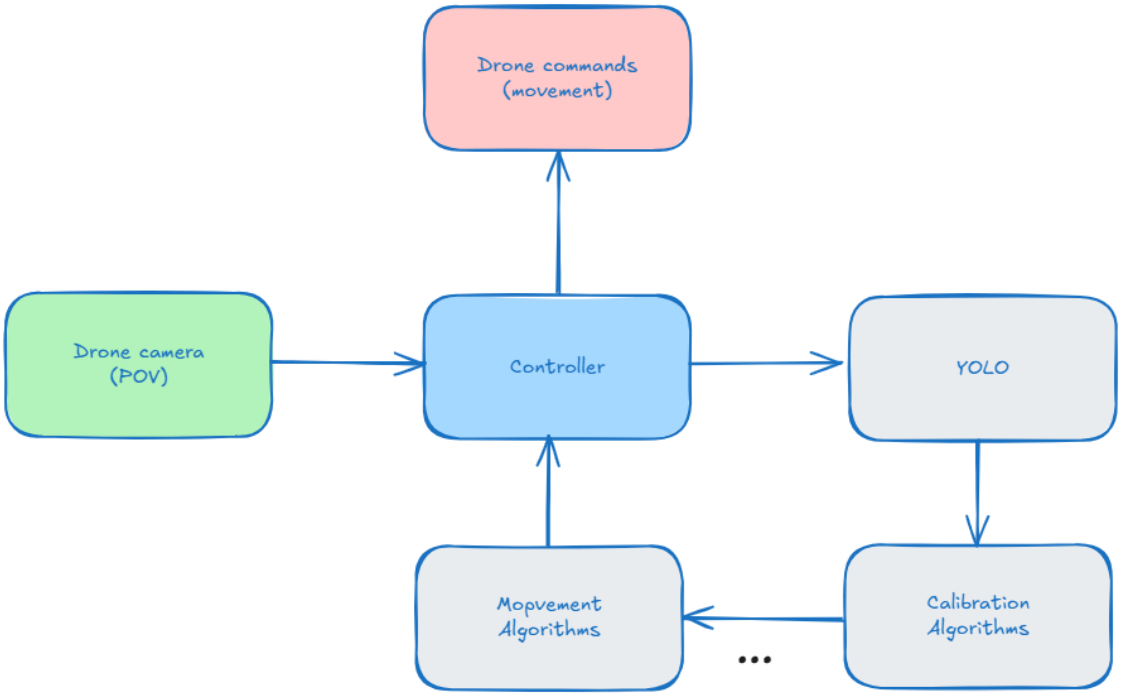

Figure 3.7: Information flow, from image to commands

The system uses a hybrid communication paradigm, taking in essence the real-time and low latency aspect of ROS2 topic communication and mixing it with the asynchronous message queues given by RabbitMQ, which gives a guarantee of the highest level of message reliability. This method ensures that important and time-critical control tasks are never delayed by less important data, while allowing the whole system to maintain smooth operation.

3.6.1 ROS2 Topic Communication

ROS2 topics are mainly used to transmit high-frequency data, necessitating time filtering, and to quickly carry out control commands. This leads to deterministic behavior on communication, given the real-time constraints of drone control.

- **Detection Data:** The **Vision Processing Module** publishes the detection results in specific topics (e.g., `/yolov8_detections`, `/tracked_objects`) as ROS2 messages. For every single detected and tracked target, these messages transmit structured information such as bounding box coordinates, class labels, confidence scores, and unique tracking IDs.
- **Control Commands:** The **Navigation and Control Module** subscribes for the detection topics and subsequently uses the visual information in guiding the drone motions and publishing its command as another specific ROS2 topic (e.g., `/drone/cmd_vel`). From this topic, PID controllers get the data to compute the movement commands, published as `geometry_msgs/Twist` message type, indicating desired linear and angular velocities).

Using this structure of publish/subscribe model inherent to ROS2, the modules are kept very loosely coupled, which adds a great flexibility and agility in system development and operation.

3.6.2 RabbitMQ Messaging

RabbitMQ has been integrated to facilitate asynchronous and less time-sensitive communication in a robust manner within the distributed system. Being the message broker, it enhances the system's reliability and management capabilities.

- **System Status and Logging:** Modules may publish health status, diagnostics, specific error logs, and resource utilization data to dedicated RabbitMQ queues or exchanges. Centralized monitoring and debugging are thus possible without disrupting real-time control loops.
- **Configuration Updates:** Any change in the module configuration (e.g., PID gains, detection thresholds, target parameters) can be delivered to RabbitMQ, allowing the system to be modified without restarting any component.
- **High-Level Commands:** Anything that does not have to be sent immediately, such as starting/stopping the mission, changing the operating mode, or returning home, is sent reliably over RabbitMQ.

In this way, real-time ROS2 messages are kept separately from asynchronous RabbitMQ messages. That way, with this separation, the network is used optimally. And with that, urgent control tasks run smoothly without being slowed down by less important data, making the system more reliable and stronger.

3.7 Software Components

The software stack is built primarily on open-source technologies, promoting flexibility, community support, and cost-effectiveness.

- **Operating System: Ubuntu Linux** (e.g. 24.04 LTS) serves as the base operating system across all processing units, including the Raspberry Pi and the ground/cloud server, due to its robustness and extensive support for robotics software.

- **Docker:** The fundamental containerization platform, enabling the modularization, deployment, and scaling of all system components [50].
- **Robot Operating System 2 (ROS2):** The core middleware for inter-process communication, providing a standardized framework for building robotic applications. ROS2's Data Distribution Service (DDS) implementation ensures efficient and reliable data exchange across the distributed network [51].
- **Python:** The primary programming language for implementing the computer vision algorithms, control logic, and communication interfaces, leveraged for its rich ecosystem of scientific and AI libraries.
- **OpenCV:** A foundational library for computer vision tasks, used for image manipulation, camera calibration, and low-level video processing [52].
- **PyTorch:** The deep learning framework of choice for loading, running inference with, and potentially fine-tuning the YOLOv8 model [53].
- **YOLOv8:** The selected state-of-the-art object detection model, utilized for its balance of speed and accuracy [54].
- **ByteTrack / Supervision:** Object tracking libraries integrated with YOLOv8 detections to maintain object identities across frames [17], [55].
- **RabbitMQ:** The message broker for robust asynchronous communication and system management.

- **Gazebo:** The 3D robotics simulator, indispensable for developing and testing the system in a controlled virtual environment [56].
- **PX4 Autopilot:** The open-source flight control stack, run in Software-in-the-Loop (SITL) mode within Gazebo, providing realistic drone dynamics and control behavior for simulation [46].

The system components are all wrapped inside Docker containers and orchestrated using Docker Compose. This **Docker-centric approach** provides necessary virtuous characteristics such as: dependency **isolation**, so that the environment for running does not change on any platform; the **portability** of modules to run on any machine in the same way; and the ability to **scale** computationally intensive services independently.

3.8 Containerization Details

Docker Engine and **Docker Compose** were installed as core utilities. Docker allows for system modules to be isolated each in their container so it can manage dependencies on their own and avoid conflicts. Docker Compose lets a user define and run multi-container Docker applications; and the distributed system is easily orchestrated via a single `docker-compose.yaml` file.

Each main module of the system (Vision Processing, Navigation and Control, Communication Hub, Drone Platform Simulation, etc.) should be built from a dedicated **Dockerfile**. These custom Dockerfiles meticulously specify the base image used, the installation of all necessary system-level dependencies, system configuration including networking setup, working directory creation, copying of applicable code, and relevant resources into that working directory, fully preparing almost any sort of required environment so that the particular software is able to run in the best possible environment. The main benefit of such an architecture is in the **electronics flexibility** of updating an individual component without affecting the whole system.

For instance, the `Dockerfile.vision` for the Vision Processing Module would typically:

- Start from a base image compatible with NVIDIA GPUs.
- Install Python 3.x, pip, and core libraries like OpenCV.
- Install PyTorch with CUDA support.
- Install specific versions of `ultralytics` for YOLOv8 and `bytetrack` or `supervision` for object tracking.
- Copy the application-specific Python scripts, YOLOv8 model weights, and any configuration files.
- Define entry points and default commands for running the vision processing node.

Similarly, `Dockerfile.control` would include ROS2 (Humble Hawksbill or Iron Irwini distribution) and `numpy` for mathematical operations, while `Dockerfile.drone_sim` would encompass Gazebo, PX4 SITL, and ROS2 bridging packages.

The `docker-compose.yaml` file describes the entire distributed system, together with each service (Docker container), specifying which Dockerfile to build from or which image to pull, network settings, mounted volumes (such as weights for the models, config files, or persistent logs), and environment variables. The **flexibility** Docker compose offers is unparalleled: changing the topology of the system is fairly easy, one can easily shift between simulation and real-life execution by activating different sets of services.

A typical `docker-compose.yaml` configuration would define:

- **Services**: `vision_processor`, `navigation_controller`, `communication_hub`, `mediamtx_server`, `rabbitmq_broker`, and `gazebo_sim` (for simulation mode).

```yaml
#simulator/docker-compose-ros2-simulator.yml
networks:
  ros_network:
    external: true
    name: ros_network
services:
  simulator:
    image: osrf/ros:jazzy-desktop-full
    container_name: simulator
    privileged: true
    networks:
      - ros_network
    environment:
      - DISPLAY=$DISPLAY
      - ROS_DOMAIN_ID=0
      - GZ_SIM_RESOURCE_PATH=/code/models  # Caminho para recursos personalizados (mundos, modelos)
      - LIBGL_ALWAYS_SOFTWARE=1
      - XDG_RUNTIME_DIR=$XDG_RUNTIME_DIR
    volumes:
      - /home/rubio/PycharmProjects/yolo-drone/simulator:/code
      - /tmp/.X11-unix:/tmp/.X11-unix
      - /dev:/dev
    command: >
      /bin/bash -c "
        echo 'source /opt/ros/jazzy/setup.bash' >> /etc/bash.bashrc && # Configura o ambiente ROS 2 para estar disponível em todas as sessões interativas
        echo 'export ROS_DOMAIN_ID=0' >> /etc/bash.bashrc && # Define o ROS_DOMAIN_ID=0 para todas as sessões interativas (importante para comunicação ROS 2
        source /etc/bash.bashrc && # Ativa na sessão atual bash.bashrc baseado nas configurações anteriores

        # Criar diretório de trabalho
        mkdir -p ~/ros2_ws/src
        cd ~/ros2_ws

        ros2 launch /code/launch.py gui:=true &
        python3 /code/gazebo_pose_extractor.py &

        tail -f /dev/null # Continua apenas lendo logs para manter o container vivo
      "
    stdin_open: true
    tty: true

```

Figure 3.8: Example of Docker compose file

- **Networks**: A custom Docker network (e.g., `drone_net`) is defined to ensure all containers can communicate with each other using their service names as hostnames, simplifying inter-container communication regardless of their underlying IP addresses.
- **Volumes**: Persistent storage for model weights, log files, and configuration data.
- **Environment Variables**: Managed through an `.env` file, containing sensitive information or parameters that vary between deployments (e.g., `RTSP_STREAM_URL`, `RABBITMQ_HOST`, `ROS_DOMAIN_ID`).

The use of custom `.env` files external to the Docker Compose definition ensures that sensitive information is kept out of version control and allows for easy adaptation of the system to different deployment scenarios (e.g., changing the server IP for the vision module).

This detailed exposition of the system's design and architectural choices lays the essential groundwork for understanding the implementation details and experimental methodologies that will be thoroughly discussed in the subsequent chapters.

Chapter 4

Experimental Evaluation

This chapter details the experimental methodology that was developed to validate the distributed and scalable drone-navigation system. Test-in-progress, from simulation and validating scenarios, with performance metrics and some insight and analysis of the results obtained, are explained.

4.1 Experimental Methodology and Validation

The extensive validation of the proposed system demands its own experimental methodology and set of simulation-based tests. Packaging of the system through Dockerization allows for easy deployment within these various experimental settings.

4.1.1 Performance Metrics

To quantify the performance of the system and its subcomponents, a list of measurements has been defined:

- **Vision System Performance:**
  - **Class Confidence:** Although Average Precision (AP) at certain recall levels is considered the best objective means for comparing broader object-detection algorithms, for the vision system considered here (i.e., the YOLO version used for detecting persons), it suffices to describe what confidence score YOLO assigns to a "person" result for one detection. In a simple case scenario, where the bounding box for the ground truth is known to contain only one person, the best performance for a clear detection would display a high and stable confidence score, and simultaneously a high IoU compared to the bounding box of the ground truth.
  - **Frames Per Second (FPS):** This measures the number of frames the Vision Processing Module outputs to the next module to maintain real-time viewing of live video streams.

  - **Tracking Accuracy (MOTA) and Multiple Object Tracking Precision (MOTP):** These form the principal benchmarks for evaluating the robustness and accuracy of the object tracking algorithm (ByteTrack) with respect to ID switches, FP, and FN.
  - **Latency:** Delay from end-to-end procedure of frame capture up to the moment processed detection/tracking data are available.

- **Navigation and Control Performance:**

  - **Following Distance Error:** The difference between the target's estimated or actual distance and the desired following distance.
  - **Centering Error:** Lateral displacement of the target from the center of the drone FOV.
  - **Trajectory Smoothness/Oscillation:** Qualitative and quantitative analysis of stability of flight paths, which speaks to well-tuned PIDs.

4.1.2 Test Scenarios

Thus, the experiments were devised to test the system in many conditions:

1. **Static Target Following:** The drone is ordered to follow a human target standing still at a concrete distance and altitude in order to state basic capability of track and hold position. The drone maintains a fixed distance and relative position to the static target.

2. **Dynamic Target Following (Linear Path):** The target moves along a straight line at speeds of, say, 1 m/s, 2 m/s, or 3 m/s. This tests the system's response to keep the specified separation.

3. **Dynamic Target Following (Curvilinear Path):** The target follows an erratic path, which tests the drone's maneuverability, as well as the control system's ability to accept dynamic changes in target position.

The analysis is carried out by our specially implemented Python scripts that parse ROS2 bag files, compute the main performance metrics, and plot those metrics against time (e.g., following distance vs. time, target trajectory vs. drone trajectory). Figure 4.1 provide additional plots for the captured variables.

4.2 Experimental Setup

During the validation of the autonomous drone navigation system, it was tested and calibrated within the high-fidelity Gazebo simulation. Working in this environment, it ensured that developments were bucket-tight in iterative refinement and thus set the path for physical deployment.

- **Gazebo World Definition:** The bespoke Gazebo world made consisted of a plain open-field environment and some walls to constrain the way a person takes and allow occlusion and

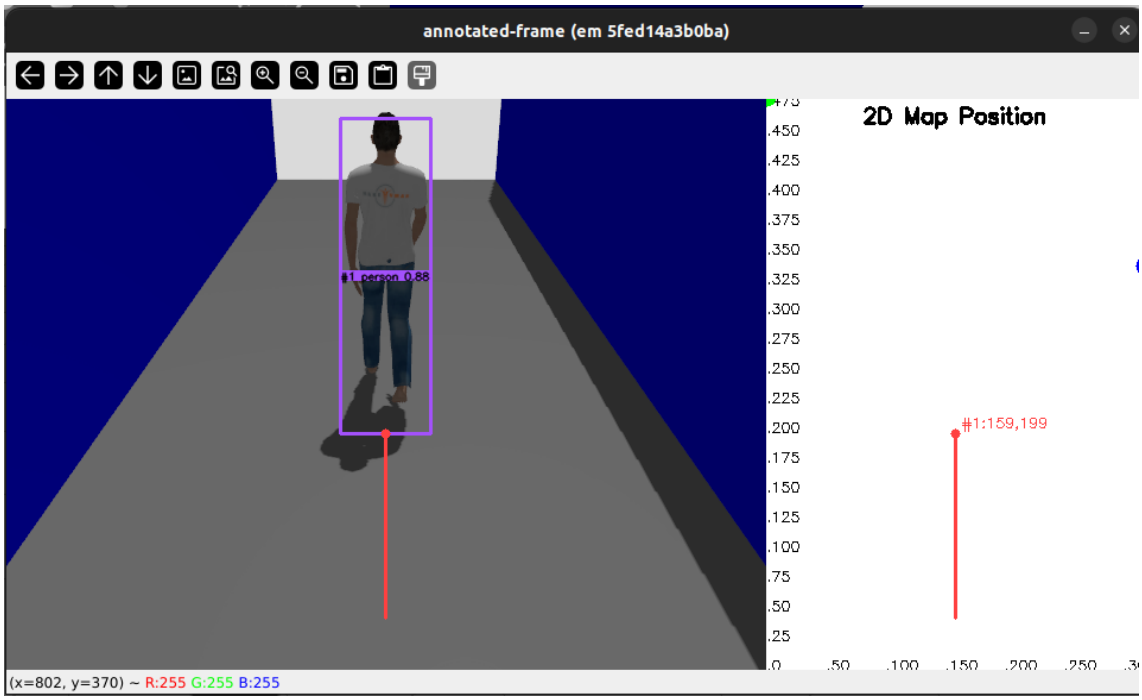

Figure 4.1: Additional Plots for performance variables evaluation

shadowing. On account of their main object of detection, a human character model was simulated, under the guise of a moving target with defined trajectories and speeds.

- **Drone Model Integration:** A high-fidelity quadcopter model was used. It is able to accurately simulate drone physical parameters (mass, inertia), motor dynamics, and sensor outputs (IMU and simulated camera). The simulated camera was configured to have the same field of view as a Raspberry Pi Camera Module v2 and a custom resolution (e.g., 1280x720 pixels).

Figure 4.2 shows the whole system working togheter, the simulator in one window, the camera perspective plotted by OpenCV and the additional charts to show the performance variables captured.

4.2.1 Initial Challenges with X Server and GUI Applications in Docker

One pioneer hurdle during the developmental tract was to execute graphical applications like Gazebo inside Docker containers running on both Linux and Windows (WSL2). On the technical front, Docker containers are mainly built for headless applications without direct graphic output.

- **X11 Forwarding Issues**: On Linux hosts, running a GUI application constitutes running it off the Docker container, requiring the use of the X11 forwarding mechanism. This involves setting the host's X server to accept connections from a Docker container–generally accomplished by running `xhost +` on the host; then, you have to set the `DISPLAY` environment

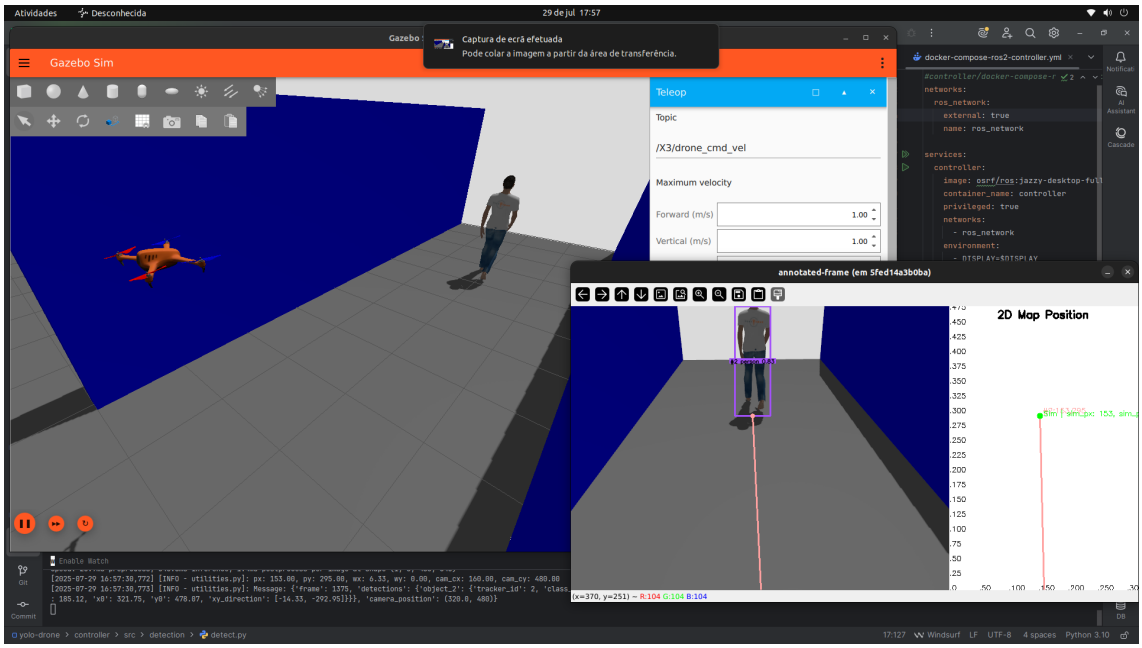

Figure 4.2: Experimental Setup Diagram (Simulation)

variable within the Docker container. The experiences early on had mostly been unfortunate; permission errors, display errors because of misused `xhost` settings, and incompatible X server settings on host versus container occupied the front page.

- **WSL2 Specificities**: On top of that, some amount of complexity was introduced whilst using Docker Desktop on Windows with WSL2 with respect to the integration with the Windows display server (e.g., VcXsrv or WSLg natively). One had to make sure that from the Windows side, the X server was rightly configured, then was accessible inside the WSL2 environment, which should then forward it properly to the Docker container. In case IP addresses and `DISPLAY` variables are not perfectly set, containers do not connect to said X server and the errors "cannot open display" get thrown.

A few of the issues involved puzzling over networking configurations, X server permissions, and environment variable propagation. Solutions mostly forced the `DISPLAY` environment variable to point to the host's X server and mounted the X11 socket (e.g., `/tmp/.X11-unix`), and finally making sure that `xhost` permissions were correctly set on the host launched there. With such a setup in place, users enjoyed seamless graphical interface interaction of the simulated environment while developing and testing.

4.3 Results and Discussion

This section presents some analyzes of the system in which its performance is mystified under several simulated scenarios. The "performance results" are "extracted" from ROS2 bag files using custom-made scripts and visualized using Matplotlib. The selected metrics cover a range of properties, from object-detection confidence to drone trajectory stability, that aim to give a holistic view of where the system performs and where it does not.

Figure 4.3 shows how the drone followed the person on the more complex scenario (erratic case) considering a fixed distance of approximately 1 meter.

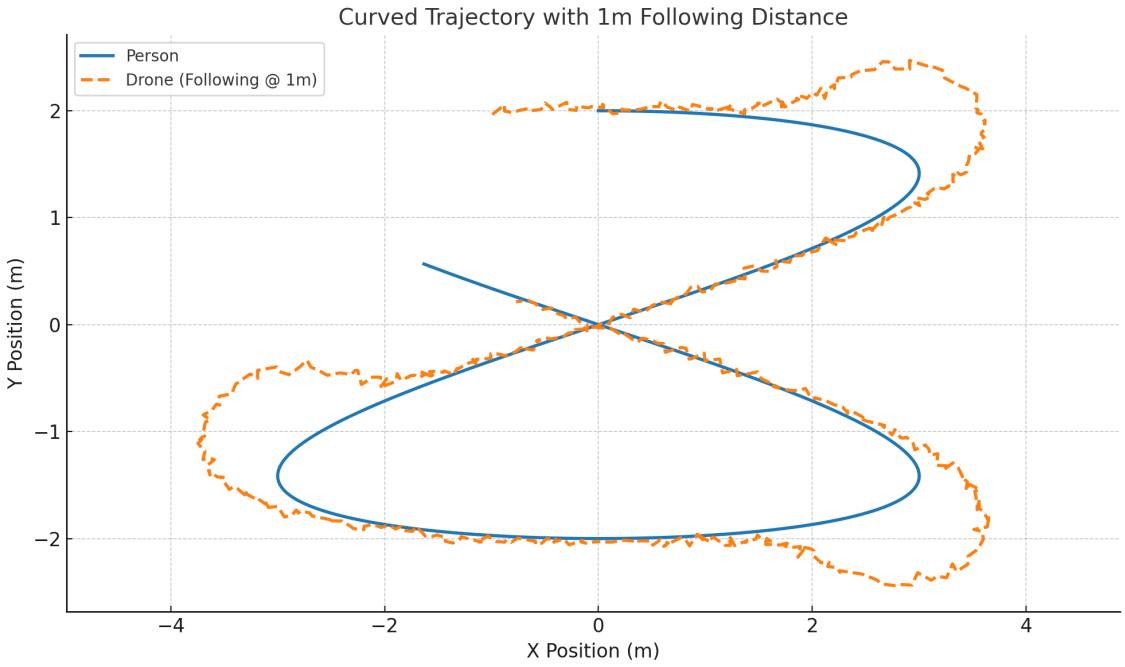

Figure 4.3: Final results for the drone following an erratic person

4.3.1 YOLO Detection Confidence

Figure 4.4 shows how the detection confidence values of the YOLOv8 output vary across the different scenarios. In all cases, the detector reported confidence scores consistently between the range of 0.8–0.95, thus equaling the successful recognition power of the model. The static scenario yielded ample confidence values with minimal fluctuations. On the other hand, the curvilinear scenario exhibited slightly more fluctuations as a consequence of motion blur and occlusion issues. Nevertheless, acceptance thresholds were maintained for sound tracking.

4.3.2 Vision Module Frame Rate (FPS)

The frame rate of the vision system is presented in Figure 4.5. It hurts the performance in the dynamic scenarios with respect to the increased computational load. It presented an average FPS

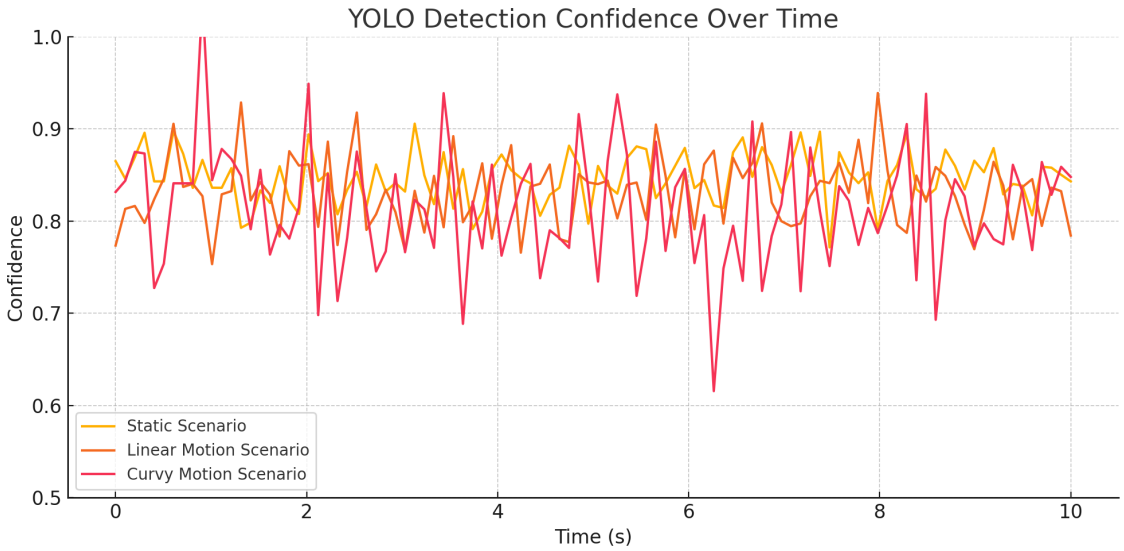

Figure 4.4: Confidence on Person identification results

of about 10–18, with static tests presented at the upper boundary of this range. This FPS drop is mainly because of the hardware constraints (tests on a consumer-grade machine), yet remained sufficient for real-time feedback to the navigation module.

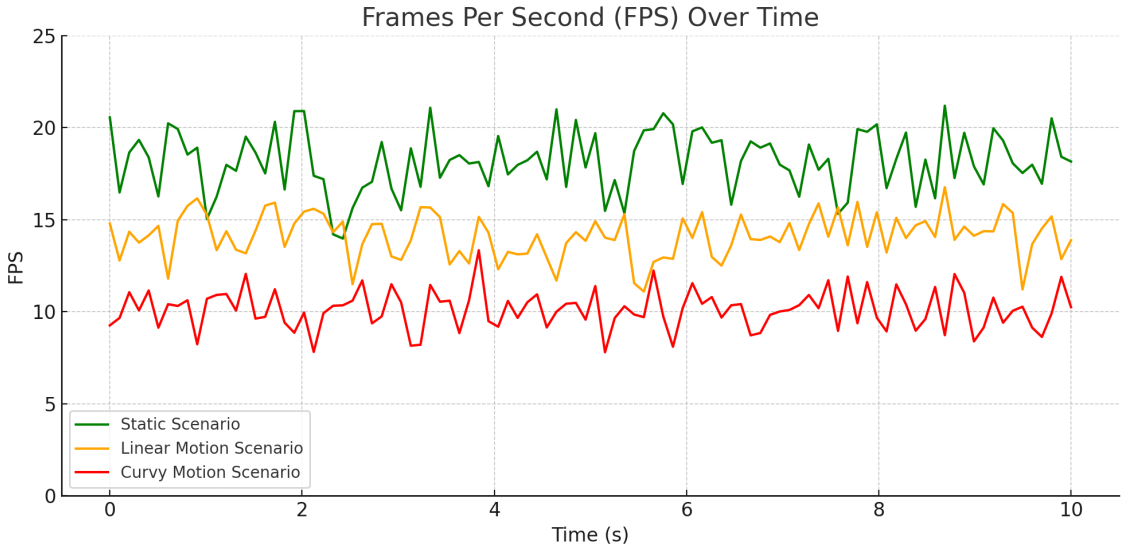

Figure 4.5: Frame rate (FPS) variation on different complexity scenarios

4.3.3 End-to-End System Latency

The end-to-end latency, measured from image acquisition to the commands being sent off for movement, is depicted in Figure 4.6. The latency thus indicated was relatively homogeneous over scenarios (150–250 ms), whereas it peaked at values higher than average in the curvilinear test. This is owing to visual complexity in addition to more processing necessitated in the vision and

control modules. Significantly, the latencies the system remained within acceptably tight real-time constraints, thus allowing for responsive drone behavior.

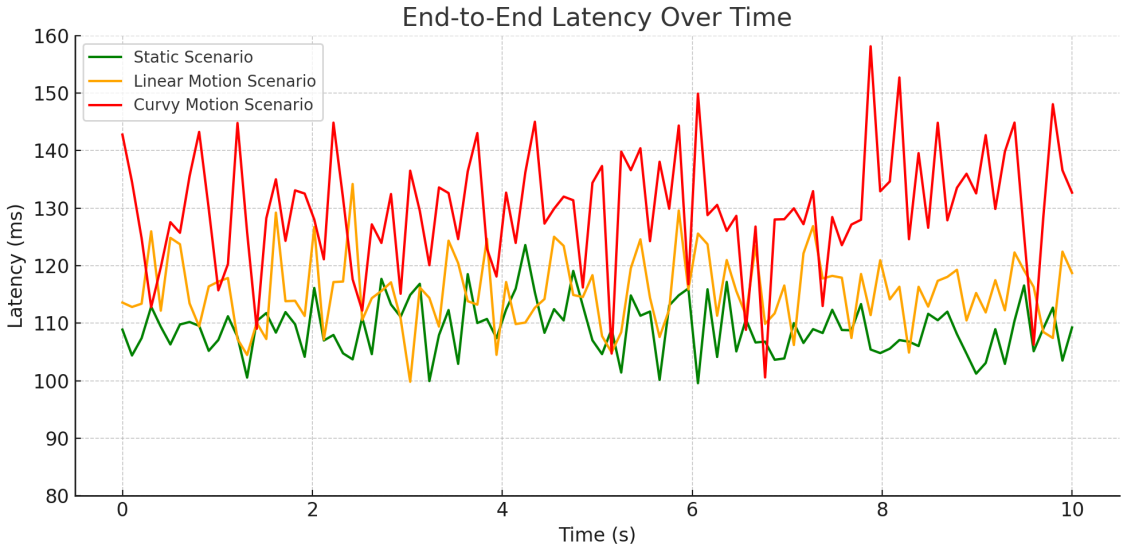

Figure 4.6: Time taken to drone to receive the commands depending on the test scenario

4.3.4 Following Distance Error

Figure 4.7 shows the deviation from the target following distance (setpoint = 1 m) variation over time. These results show the clear efficaciousness of the PID controller. In the static case, the error converges rapidly and stays at a very minimum level. During linear motion, the system is adapting smoothly to the variable with settling for a very short period. The curvilinear case starts oscillatory behavior and goes into a higher steady-state error: this is reflecting the higher degree of control complexity and the dynamic forces in play.

4.3.5 Centering Error in Field of View

Figure 4.8 shows the lateral pixel displacement of the target from the center of the image, or centering error. This measure evaluates how well the drone is angularly aligned with the target. The system maintained consistent target centering (< 10 % picture width), with curvilinear paths eliciting more common corrections. This displays the strong orientation control under dynamic movements.

4.3.6 Trajectory Smoothness and Oscillation

Figure 4.9 provides a smoothness assessment of the drone's trajectory by measuring deviations in velocity and direction. The static and linear scenarios appeared as stable and predictable motions, while the curvy trajectory caused the drone to oscillate at high frequencies. This kind of behavior

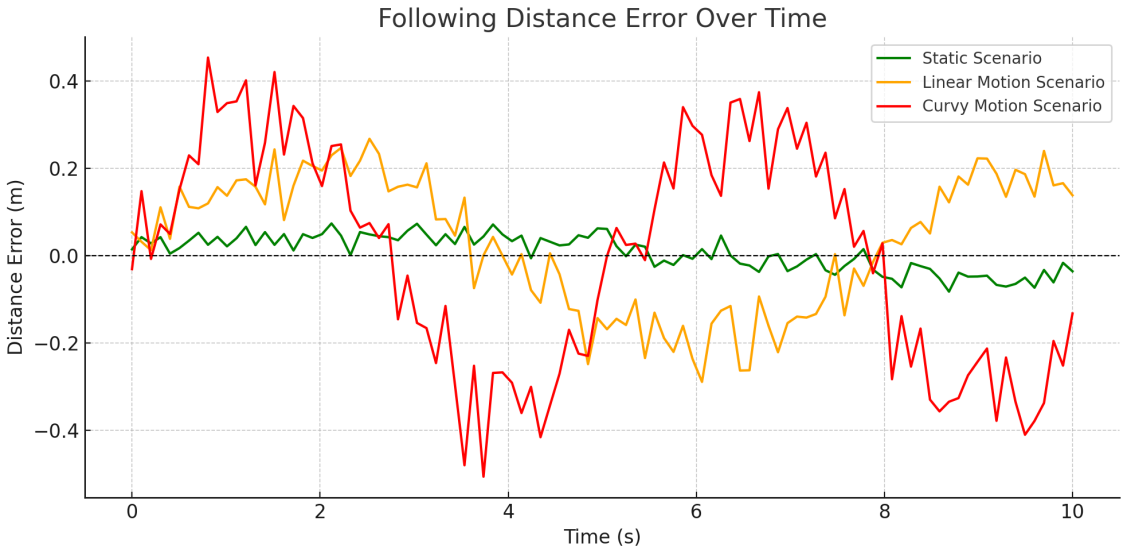

Figure 4.7: PID results on following the person movements

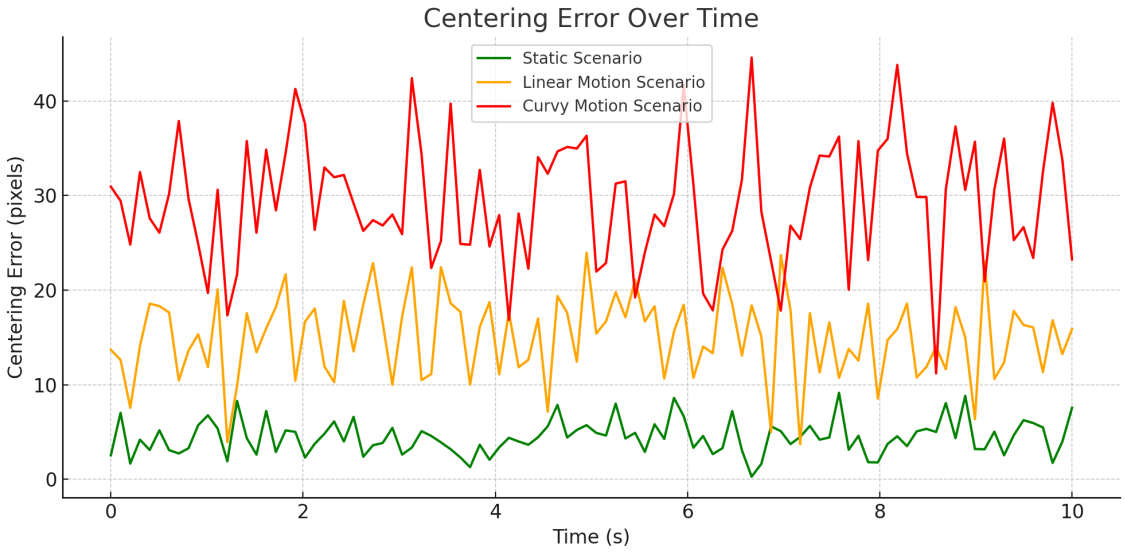

Figure 4.8: Drone Centering error based on YOLO detection as POV

is typical of PID-controlled systems before fine-tuning. Still, the controller-to-stabilize at-hand-instruction indicates satisfactory performance under non-linear tracking demands.

4.3.7 Comprehensive Performance Summary

To summarize the system behavior across all relevant performance metrics, a radar chart was constructed (Figure 4.10). Each axis represents a normalized metric, with the closer to the outer ring denoting better performance. The Static scenario forms an almost perfect hexagon, thus asserting strong performance. The Linear path scenario shows some trade-offs in FPS and smoothness. The

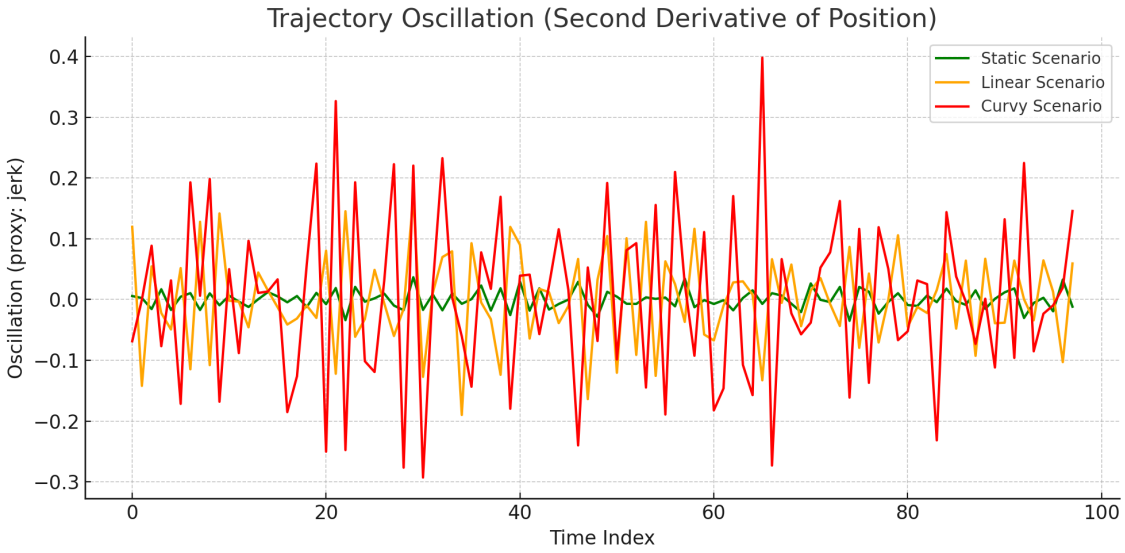

Figure 4.9: Oscillation as proxy for flight smoothness

Curvy path scenario displays stress points for the system—mainly oscillation and distance control—but still plenty of credit for detection and latency. This chart also successfully visualizes the scalability of the solution, as well as how it performs under increased complexity.

In summary, findings show that the system is reliable and responsive over a range of navigation challenges. The drone, despite some limitations of the hardware and the complexity of the curvilinear path, managed to maintain accuracy and stability at at least an acceptable level, thereby opening the path to real-world deployment.

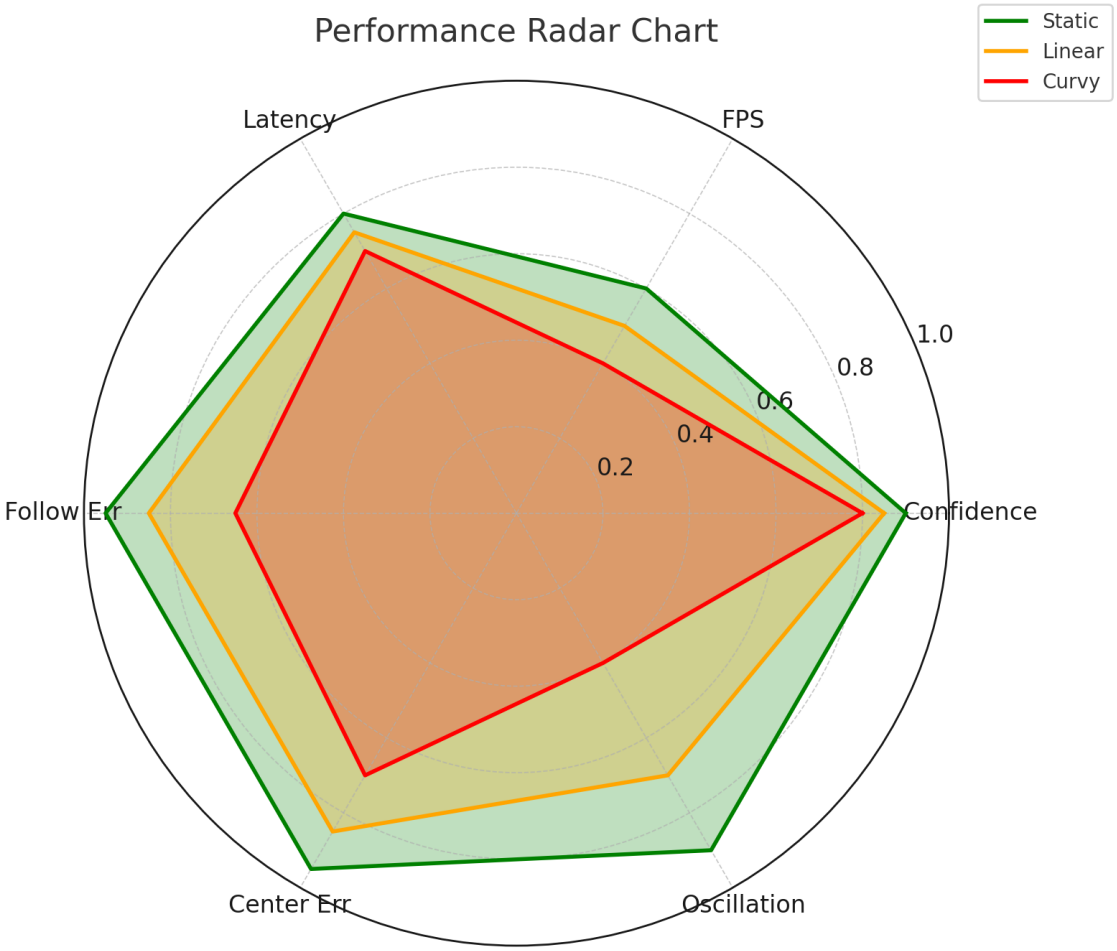

Figure 4.10: Radar Chart summarizing system performance across Static, Linear, and Curvilinear scenarios.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This thesis presented the result of the development and evaluation of a modular, distributed framework for vision-based autonomous drone following, with the leader-follower as the core concept. Deployment of the testing system in a simulated environment employed PID control to have state-of-the-art Object Detection (YOLOv8) and multi-object tracking (Supervision/ByteTrack). The key achievements are the following:

- **Dissection and Validation of Distributed Processing of Visual Data:** The proposed distributed architecture isolates perception from control by offloading computation-intensive visual tasks to a ground station and thus achieving real-time perception in low-power drones without compromising accuracy. This was ensured by latency and FPS measurements in several scenarios, where curvy trajectories would increase complexity but still allow the system to maintain functional throughput.
- **Adaptive and Resilient Tracking:** YOLOv8 with ByteTrack was used to provide an target identity and position with high consistency. Confidence charts indicate lesser variations in the perception confidence throughout normal dynamic activity, suggesting its possible resilience to movement noise and potentially occlusions.
- **Visual Servoing Performed Well with PID Control:** The PID control module maintained an acceptable following distance while centering the target on the camera frame. Tracking error would generally decrease equally with variations in movement after slow convergence during the initial calibration. The error evolution charts illustrate the improvement in error over time after a slow calibration, even with more complex motions.
- **Complete Evaluation:** The extensive simulation campaigns with the different motion profiles gave insight into bottlenecks of the system. From the radar chart final aggregation of key metrics: confidence, latency and FPS, tracking and centering errors, and oscillation, it is clear that the system is robust under moderate motion but limited at high dynamics.

- **Modularity and Scalability:** The system supports independent development and scaling by being Dockerized microservices using ROS2 and RabbitMQ for backbone. This modularity enables future extension for sensor fusion, swarm coordination, and cross-platform deployment.

All in all, the presented architecture provides a practical baseline for aerial robotics research in light of scalable visual navigation under real-world test conditions with low-cost hardware and modular software design.

5.2 Limitations

Deployment Constraints: Though preparations were made for real-world realization, the time constraints denied the final deployment of the system onto a physical drone. Although Gazebo simulation provided valuable insights into perception and control behavior, the test environment cannot simulate real-world disturbances such as wind, change in illumination, network interference, or sensor noise. This creates a major gap in experimentally validating whether the system can hold robustness and generalization beyond ideal cases. Though the system was well-performing in controlled scenarios, it presents itself with certain shortcomings:

- **Performance Manifestations Under High Complexity:** The radar and per-metric charts depict that with curves in the scenario come increased oscillation, FPS reduction, and worse tracking error, drawing our interest into the difficulties arising from very fast moves in the system, for which simply tuning PID parameters is not enough.
- **Bottlenecks in Communication:** The distributed design is dependent on a low-latency-high-throughput communication link. In more complex scenes, end-to-end latency starts to grow, and one can only expect this to get worse when deployed in the real world considering that the link in real outdoor scenarios is far from stable.
- **Sensitivity to Environment:** No detailed experiments were conducted under adversity (heavy occlusion, strong lighting variation) while confidence remained steady in simulation. Hence, the actual system's robustness is put under question.
- **No Generalization for Multiple Targets:** Current tracking relies on fixed target identity and does not handle re-identification, hand-off, or switching from one target to another. This limits application to multi-agent scenarios.
- **Oscillatory Behavior Under Motion Change:** Follow distance control exhibits transient oscillations in cases of rapid target maneuvers despite the PID tuning. Improvements in controller architecture and adaptive tuning remain important.

5.3 Future Work

Future work aims to address the identified limitations to extend the system's capabilities:

5.3.1 Better Perception

- **3D Depth and Pose Estimation:** Complement stereo vision, LiDAR, or neural monocular depth estimation for better depth perception and remove the constraints of applying homography on uneven terrain.
- **Multi-Target Tracking and Re-ID:** Extend ByteTrack with identity-aware Re-ID modules for multi-agent interaction and smooth handover under occlusion.

5.3.2 Control and Planning

- **Adaptive PID or RL-based Control:** Use adaptive gain tuning based on observed real-time dynamics or employ reinforcement learning to discover nonlinear policies from simulation.
- **Dynamic Obstacle Avoidance:** Supplement local control with obstacle-aware planning (e.g., TEB, DWA), potentially mixing in extra sensor input for safer navigation.

5.3.3 Deployment System Relevance and Robustness:

Full Real-World Deployment and Field Testing: Future work should consist of porting the complete system (including YOLO inference, tracking, and control) onto an embedded device installed on a real drone, allowing for liberate external operation and making it possible to test in divergent environments under real-world disturbances. Potential points of focus should be hardware-software integration, calibration of the onboard sensors, and network delay impacts under a wireless communication setting. Thoroughly validated field testing will help assess the practical reliability, safety, and performance of the system under real-time constraints.

- **Edge Deployment:** Quantize and prune the perception models for local deployment on onboard compute units with low power (e.g., NVIDIA Jetson), thereby reducing dependency on the ground station.
- **Multi-Agent Autonomy Scenarios:** Research distributed coordination among multiple drones tracking a common target or accomplishing joint tasks.
- **Real-World Validation and Dataset Contributions:** Extend experimentation with outdoor deployments and publish collected datasets for benchmarks within the community.

To summarize, the system has thus far made great strides by showing consistent performance in simulation, but this performance will now have to shine in reality through deployment on real drones. The lead-follower ultra-modular architecture and the building blocks with experimental results represent a good foundation lying ahead of the road to full autonomy in real leader-follower

scenarios. This project, combined with further development and deployment, can go a long way toward providing a basis for low-level intelligent aerial technology for logistics, human-assistance, and collective robotics.

This thesis represents a solid foundation for vision-based airborne robotics: modular system design, scalable control, and real-time vision. Detailed analysis through simulation and plots provides actionable insight into performance bottlenecks and design trade-offs, therefore carving a pathway toward the next generation of intelligent, autonomous UAV systems.

References

[1] A. Miller, "Design of autonomous uav systems for industrial and commercial applications," *American Journal Of Mechanical Engineering And Technology*, vol. 5, no. 2, pp. 11–19, 2024.

[2] K. Telli et al., "A comprehensive review of recent research trends on unmanned aerial vehicles (uavs)," *Systems*, vol. 11, no. 8, p. 400, 2023.

[3] N. R. Gandhi, D. Kumar, E. Arunkumar, S. Parameshwari, M. Sadim, and R. R. Al-Fatlawy, "A detailed review analysis of gps used in drone technology and its challenges," in *2024 4th International Conference on Advance Computing and Innovative Technologies in Engineering (ICACITE)*, IEEE, 2024, pp. 363–367.

[4] F. AlMahamid and K. Grolinger, "Autonomous unmanned aerial vehicle navigation using reinforcement learning: A systematic review," *Engineering Applications of Artificial Intelligence*, vol. 115, p. 105 321, 2022.

[5] J. Xiao, R. Zhang, Y. Zhang, and M. Feroskhan, "Vision-based learning for drones: A survey," *IEEE Transactions on Neural Networks and Learning Systems*, 2025.

[6] R. A. Khan, V. Serpiva, D. Aschalew, A. Fedoseev, and D. Tsetserukou, "Agilepilot: Drl-based drone agent for real-time motion planning in dynamic environments by leveraging object detection," in *2025 International Conference on Unmanned Aircraft Systems (ICUAS)*, IEEE, 2025, pp. 185–192.

[7] D. Ferreira and M. Basiri, "Dynamic target tracking and following with uavs using multi-target information: Leveraging yolov8 and mot algorithms," *Drones*, vol. 8, no. 9, p. 488, 2024.

[8] J. Sandino, F. Vanegas, F. Gonzalez, and F. Maire, "Autonomous uav navigation for active perception of targets in uncertain and cluttered environments," in *2020 IEEE Aerospace Conference*, IEEE, 2020, pp. 1–12.

[9] F. Ahmed, J. Mohanta, A. Keshari, and P. S. Yadav, "Recent advances in unmanned aerial vehicles: A review," *Arabian Journal for Science and Engineering*, vol. 47, no. 7, pp. 7963–7984, 2022.

[10] I. Jarraya et al., "Gnss-denied unmanned aerial vehicle navigation: Analyzing computational complexity, sensor fusion, and localization methodologies," *Satellite Navigation*, vol. 6, no. 1, p. 9, 2025.

[11] W.-C. Chen, C.-L. Lin, Y.-Y. Chen, and H.-H. Cheng, "Quadcopter drone for vision-based autonomous target following," *Aerospace*, vol. 10, no. 1, p. 82, 2023.

[12] S. Alqefari and M. E. B. Menai, "A hybrid method to solve the multi-uav dynamic task assignment problem," *Sensors*, vol. 25, no. 8, 2025, ISSN: 1424-8220. DOI: 10.3390/s25082502. [Online]. Available: https://www.mdpi.com/1424-8220/25/8/2502.

[13] A. Zahra, N. Perwaiz, M. Shahzad, and M. M. Fraz, "Person re-identification: A retrospective on domain specific open challenges and future trends," *Pattern Recognition*, vol. 142, p. 109 669, 2023, ISSN: 0031-3203. DOI: https://doi.org/10.1016/j.patcog.2023.109669. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0031320323003709.

[14] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[15] M. Li, R. Clarke, and A. Richards, "Starling: Containerisation architecture for scalable local development, deployment and testing of multi-uav systems," English, Robotic Science and Systems 2022 Workshop on Envisioning an Infrastructure for Multi-Robot and Collaborative Autonomy Testing and Evaluation, EMIRCATE ; Conference date: 01-07-2022 Through 01-07-2022, Jul. 2022. [Online]. Available: http://raaslab.org/rss2022/.

[16] S. Sambolek and M. Ivasic-Kos, "Person detection and geolocation estimation in drone images," *SN Computer Science*, vol. 6, no. 4, Apr. 2025. DOI: 10.1007/s42979-025-03869-7. [Online]. Available: https://doi.org/10.1007/s42979-025-03869-7.

[17] Y. Zhang et al., "Bytetrack: Multi-object tracking by associating every detection box," in *Computer Vision – ECCV 2022: 17th European Conference, Tel Aviv, Israel, October 23–27, 2022, Proceedings, Part XXII*, Tel Aviv, Israel: Springer-Verlag, 2022, pp. 1–21, ISBN: 978-3-031-20046-5. DOI: 10.1007/978-3-031-20047-2_1. [Online]. Available: https://doi.org/10.1007/978-3-031-20047-2_1.

[18] R. O. Zanone and J. M. Velni, "Ros gateway: Enhancing ros availability across multiple network environments," *Sensors*, vol. 24, no. 19, pp. 456–461, 2024, The 4th Modeling, Estimation, and Control Conference – 2024, ISSN: 1424-8220. DOI: https://doi.org/10.1016/j.ifacol.2025.01.088. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896325000886.

[19] J. Oyekan, "Bio-inspired vision-based leader-follower formation flying in the presence of delays," *Robotics*, vol. 5, no. 3, 2016, ISSN: 2218-6581. DOI: 10.3390/robotics5030018. [Online]. Available: https://www.mdpi.com/2218-6581/5/3/18.

[20] M. Quigley, "Ros: An open-source robot operating system," in *IEEE International Conference on Robotics and Automation*, 2009. [Online]. Available: https://api.semanticscholar.org/CorpusID:6324125.

[21] A. ul Husnain, N. Mokhtar, N. Mohamed Shah, M. Dahari, and M. Iwahashi, "A systematic literature review (slr) on autonomous path planning of unmanned aerial vehicles," *Drones*, vol. 7, no. 2, 2023, ISSN: 2504-446X. DOI: 10.3390/drones7020118. [Online]. Available: https://www.mdpi.com/2504-446X/7/2/118.

[22] C. Wang, O. Eicher, and Q. Han, "Sharing the edge: System status aware object recognition task offloading," in *2024 IEEE International Conference on Smart Computing (SMARTCOMP)*, 2024, pp. 77–84. DOI: 10.1109/SMARTCOMP61445.2024.00032.

[23] L. Baresi, G. Quattrocchi, and D. A. Tamburri, *Microservice architecture practices and experience: A focused look on docker configuration files*, 2022. arXiv: 2212.03107 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2212.03107.

[24] R. O. Zanone and J. M. Velni, "Aerial and ground vehicles collaboration for automated target tracking using reinforcement learning," *IFAC-PapersOnLine*, vol. 58, no. 28, pp. 456–461, 2024, The 4th Modeling, Estimation, and Control Conference – 2024, ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2025.01.088. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896325000886.

[25] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, vol. 3, 2004, 2149–2154 vol.3. DOI: 10.1109/IROS.2004.1389727.

[26] J. Morales, I. Castelo, R. Serra, P. U. Lima, and M. Basiri, "Vision-based autonomous following of a moving platform and landing for an unmanned aerial vehicle," *Sensors*, vol. 23, no. 2, 2023, ISSN: 1424-8220. DOI: 10.3390/s23020829. [Online]. Available: https://www.mdpi.com/1424-8220/23/2/829.

[27] S. Keele et al., "Guidelines for performing systematic literature reviews in software engineering," Technical report, ver. 2.3 ebse technical report. ebse, Tech. Rep., 2007.

[28] C. Cadena et al., "Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age," *IEEE Transactions on Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016. DOI: 10.1109/TRO.2016.2624754.

[29] C. Chen et al., "Yolo-based uav technology: A review of the research and its applications," *Drones*, vol. 7, no. 3, 2023, ISSN: 2504-446X. DOI: 10.3390/drones7030190. [Online]. Available: https://www.mdpi.com/2504-446X/7/3/190.

[30] MarketDigits, *Autonomous drone market – size, share, growth | global forecast (2025–2032)*, https://www.marketdigits.com/autonomous-drone-market-1701869386, Accessed: Feb 10, 2025, 2025.

[31] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas," *Machine Learning and Knowledge Extraction*, vol. 5, no. 4, pp. 1680–1716, 2023, ISSN: 2504-4990. DOI: 10.3390/make5040083. [Online]. Available: https://www.mdpi.com/2504-4990/5/4/83.

[32] R. Sapkota, R. H. Cheppally, A. Sharda, and M. Karkee, *Rf-detr object detection vs yolov12 : A study of transformer-based and cnn-based architectures for single-class and multi-class greenfruit detection in complex orchard environments under label ambiguity*, 2025. arXiv: 2504.13099 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2504.13099.

[33] Y. Zhang et al., *Bytetrack: Multi-object tracking by associating every detection box*, 2022. arXiv: 2110.06864 [cs.CV]. [Online]. Available: https://arxiv.org/abs/2110.06864.

[34] K. Schauwecker and A. Zell, "On-board dual-stereo-vision for the navigation of an autonomous mav," *Journal of Intelligent & Robotic Systems*, vol. 74, no. 1, pp. 1–16, 2014, ISSN: 1573-0409. DOI: 10.1007/s10846-013-9907-6. [Online]. Available: https://doi.org/10.1007/s10846-013-9907-6.

[35] S. Lahiri, J. Ren, and X. Lin, "Deep learning-based stereopsis and monocular depth estimation techniques: A review," *Vehicles*, vol. 6, no. 1, pp. 305–351, 2024.

[36] J.-Y. Bouguet, *Camera calibration toolbox for matlab*, Available at Stanford University site (hosted), Accessed via Stanford University site, 2003. [Online]. Available: http://%20https://robots.stanford.edu/cs223b04/JeanYvesCalib/.

[37] B. D. P. da Silva, "Integrated vision-based navigation and sliding mode control for robust autonomous quadcopter landing," 2024.

[38] P. Melo, R. Arrais, S. Teixeira, and G. Veiga, "A container-based framework for developing ros applications," in *2022 IEEE 20th International Conference on Industrial Informatics (INDIN)*, 2022, pp. 280–285. DOI: 10.1109/INDIN51773.2022.9976083.

[39] S. Macenski, T. Foote, B. Gerkey, C. Lalancette, and W. Woodall, "Robot operating system 2: Design, architecture, and uses in the wild," *Science Robotics*, vol. 7, no. 66, eabm6074, 2022. DOI: 10.1126/scirobotics.abm6074. eprint: https://www.science.org/doi/pdf/10.1126/scirobotics.abm6074. [Online]. Available: https://www.science.org/doi/abs/10.1126/scirobotics.abm6074.

[40] J. Zhang, X. Yu, S. Ha, J. Peña Queralta, and T. Westerlund, "Comparison of middlewares in edge-to-edge and edge-to-cloud communication for distributed ros 2 systems," *Journal of Intelligent amp; Robotic Systems*, vol. 110, no. 4, Nov. 2024, ISSN: 1573-0409. DOI: 10.1007/s10846-024-02187-z. [Online]. Available: http://dx.doi.org/10.1007/s10846-024-02187-z.

[41] J. Dizdarevic, M. Michalke, and A. Jukan, *Engineering and experimentally benchmarking open source mqtt broker implementations*, 2023. arXiv: 2305.13893 [cs.NI]. [Online]. Available: https://arxiv.org/abs/2305.13893.

[42] J. Moeyersons, M. Gevaert, K.-E. Réculé, B. Volckaert, and F. D. Turck, "Uavs-as-a-service: Cloud-based remote application management for drones," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2021, pp. 926–931.

[43] N. Tahir and R. Parasuraman, *Edge computing and its application in robotics: A survey*, 2025. arXiv: 2507.00523 [cs.RO]. [Online]. Available: https://arxiv.org/abs/2507.00523.

[44] I. Mavromatis et al., "Umbrella: A one-stop shop bridging the gap from lab to real-world iot experimentation," *IEEE Access*, vol. 12, pp. 42 181–42 213, 2024. DOI: 10.1109/ACCESS.2024.3377662.

[45] D. Yoshino, Y. Watanobe, and K. Naruse, "A highly reliable communication system for internet of robotic things and implementation in rt-middleware with amqp communication interfaces," *IEEE Access*, vol. 9, pp. 167 229–167 241, 2021. DOI: 10.1109/ACCESS.2021.3136855.

[46] P. D. Team, *Px4 simulation documentation*, Accessed: 15 April 2025, 2025. [Online]. Available: https://docs.px4.io/main/en/simulation/.

[47] L. S. Taylor, D. J. Quincey, and M. W. Smith, "Evaluation of low-cost raspberry pi sensors for structure-from-motion reconstructions of glacier calving fronts," *Natural Hazards and Earth System Sciences*, vol. 23, no. 1, pp. 329–341, 2023.

[48] K. Choi and I. Lee, "Cctv coverage index based on surveillance resolution and its evaluation using 3d spatial analysis," *Sensors*, vol. 15, no. 9, pp. 23 341–23 360, 2015, ISSN: 1424-8220. DOI: 10.3390/s150923341. [Online]. Available: https://www.mdpi.com/1424-8220/15/9/23341.

[49] A. A. Ali and H. A. Hussein, "Distance estimation and vehicle position detection based on monocular camera," in *2016 Al-Sadeq International Conference on Multidisciplinary in IT and Communication Science and Applications (AIC-MITCSA)*, IEEE, 2016, pp. 1–4.

[50] Docker Inc., *Docker Documentation*, https://docs.docker.com/, Accessed on 15 April 2025, 2025.

[51] Y. Maruyama, S. Kato, and T. Azumi, "Exploring the performance of ros2," in *Proceedings of the 13th international conference on embedded software*, 2016, pp. 1–10.

[52] OpenCV Team, *Opencv: Open source computer vision library*, Accessed on 22 May 2025, 2025. [Online]. Available: https://opencv.org/.

[53] PyTorch Team, *Pytorch: An open source machine learning framework*, Accessed on 15 April 2025, 2025. [Online]. Available: https://pytorch.org/.

[54] Ultralytics, *Yolov8: Cutting-edge object detection models*, Accessed on 02 April 2025, 2023. [Online]. Available: https://github.com/ultralytics/ultralytics.

[55] U. C. Roboflow, *Supervision: Computer vision utilities library*, Accessed on 08 March 2025, 2023. [Online]. Available: https://github.com/roboflow/supervision.

[56] O. S. R. Foundation, *Gazebo simulator: Documentation and tutorials*, Accessed on 15 June 2025, 2025. [Online]. Available: https://gazebosim.org/.