UNIVERSIDADE DE SÃO PAULO
Instituto de Ciências Matemáticas e de Computação

Harpia: A Hybrid System for UAV Missions

Verônica Vannini

Tese de Doutorado do Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional (PPG-CCMC)

ICMC USP SÃO CARLOS

Verônica Vannini

Harpia: A Hybrid System for UAV Missions

Thesis submitted to the Instituto de Ciências Matemáticas e de Computação – ICMC-USP – in accordance with the requirements of the Computer and Mathematical Sciences Graduate Program, for the degree of Doctor in Science. *FINAL VERSION*

Concentration Area: Computer Science and Computational Mathematics

Advisor: Prof. Dr. Cláudio Fabiano de Motta Toledo

USP – São Carlos
December 2023

Ficha catalográfica elaborada pela Biblioteca Prof. Achille Bassi
e Seção Técnica de Informática, ICMC/USP,
com os dados inseridos pelo(a) autor(a)

Vannini, Veronica
V269h Harpia: A Hybrid System for UAV Missions / Veronica Vannini; orientador Cláudio Fabiano de Motta Toledo. -- São Carlos, 2023.
120 p.

Tese (Doutorado - Programa de Pós-Graduação em Ciências de Computação e Matemática Computacional) -- Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, 2023.

1. Robotics. 2. Artificial Intelligence. 3. UAV. 4. Autonomous Systems. I. Fabiano de Motta Toledo, Cláudio, orient. II. Título.

Bibliotecários responsáveis pela estrutura de catalogação da publicação de acordo com a AACR2:
Gláucia Maria Saia Cristianini - CRB - 8/4938
Juliana de Souza Moraes - CRB - 8/6176

Verônica Vannini

Harpia: Um sistema híbrido para missões com VANTs

Tese apresentada ao Instituto de Ciências Matemáticas e de Computação – ICMC-USP, como parte dos requisitos para obtenção do título de Doutora em Ciências – Ciências de Computação e Matemática Computacional. *VERSÃO REVISADA*

Área de Concentração: Ciências de Computação e Matemática Computacional

Orientadora: Prof. Dr. Cláudio Fabiano de Motta Toledo

USP – São Carlos
Dezembro de 2023

Dedico esse trabalho aos meu pais,
que sempre me incentivaram a olhar o mundo a minha volta.

ACKNOWLEDGEMENTS

Agradeço a Deus, pois tudo esta sobre suas mãos. - *Salmos 95:3*

Ao meu pai Guto, que sempre foi um modelo para mim de sabedoria e conhecimento, a minha mãe Vanice, que com muito amor me ensinou a olhar a beleza de cada dia, a minha irmã Isadora, que me ensinou resiliência e companheirismo e ao Karl que sempre esta disposto a ajudar, e ao Thomas, meu sobrinho que trouxe um pouco mais de felicidade pra família.

Aos meus avós, Álvaro, Nancy, Walter e Vany, por sempre perguntarem como anda meus estudos, e mesmo sem entender o que eu faço, mantém o interesse em minhas pesquisas.

Ao meu professor e orientador Dr. Cláudio Fabiano Motta Toledo, que com extrema paciência me orientou, ensinou e me motivou - da sua mais carioca forma possível - a continuar. Obrigada por acreditar em mim e no meu projeto.

Ao professor Dr. Onofre Trindade Jr., que mesmo aposentado continua me apoiando e me ensinando.

Aos meus alunos de IC, Gustavo, Matheus, Pedro e Rafael, que me mostraram como muitas vezes é mais difícil estar do lado do orientador, mas também por todo apoio e trabalho desenvolvido neste tempo.

A todas do Basquete CAASO, vocês me ajudaram a me tornar a melhor versão de mim, e sem vocês eu não teria aguentado esses dois anos.

Aos amigos, Antônio, Camila, Paula, Rafael, Elisa, Ryan e Beatriz, que mesmo quando longe, estão presentes.

Ao Instituto SENAI de inovação pela bolsa de doutorado concedida e a oportunidade de levar meu projeto para um ambiente de aplicação real.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001

“Nothing happened”
(Roronoa Zoro)

RESUMO

VANNINI, V. **Harpia: Um sistema híbrido para missões com VANTs**. 2023. 120 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2023.

Este projeto de doutorado apresenta Harpia, um sistema híbrido de planejamento de inteligência artificial para VANTs (Veículos Aéreos Não Tripulados) como foco em autonomia. Harpia tem como objetivo executar tarefas para aplicações de propósito geral com mínima intervenção humana. Para facilitar o entendimento, o problema abordado é ambientado em uma fazenda onde o sistema autônomo deve ser capaz de realizar missões com segurança. A arquitetura do sistema é implementada usando o Sistema Operacional Robótico e inclui funcionalidades como o replanejamento de tarefas e o planejamento de trajetória com desvio de obstáculos. O replanejamento pode ocorrer após mudanças na missão em tempo real ou devido a um comportamento imprevisível do VANT. Harpia combina a Linguagem de Definição de Domínio de Planejamento para planejamento de tarefas, uma Rede Bayesiana para avaliar a execução da missão, um algoritmo de K-Vizinhos Mais Próximos para selecionar um planejador de trajetória, Análise de Componentes Principais e um modelo de Árvore de Decisão para avaliar a saúde da aeronave. Portanto, a novidade do Harpia concentra-se na robustez para o planejamento autônomo e o replanejamento da sequência de tarefas e trajetórias para regiões de interesse. As principais contribuições incluem uma arquitetura de sistema autônomo para planejar missões com intervenção humana mínima, sem limitações por tarefas específicas e computacionalmente simples para operar em diversos cenários. Os testes computacionais relatam resultados para 220 cenários simulados, nos quais o Harpia lidou adequadamente com todas as situações, por exemplo, tomando decisões sobre o replanejamento de tarefas com 97,57% de precisão com base na saúde da bateria e escolhendo o melhor planejamento de trajetória para cada caso com pelo menos 95% de precisão.

Palavras-chave: Robótica, Inteligência Artificial, VANT, Sistemas Autônomos.

ABSTRACT

VANNINI, V. **Harpia: A Hybrid System for UAV Missions**. 2023. 120 p. Tese (Doutorado em Ciências – Ciências de Computação e Matemática Computacional) – Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos – SP, 2023.

This doctoral project presents Harpia, a hybrid artificial intelligence planning system for UAVs (Unmanned Aerial Vehicles) with a focus on autonomy. Harpia aims to perform tasks for general-purpose applications with minimal human intervention. To facilitate understanding, the problem addressed is set on a farm where the autonomous system must be capable of carrying out missions safely. The system architecture is implemented using the Robotic Operating System (ROS) and includes functionalities such as task re-planning and trajectory planning with obstacle avoidance. Re-planning can occur after real-time mission changes or due to unpredictable UAV behavior. Harpia combines the Planning Domain Definition Language (PDDL) for task planning, a Bayesian Network (BN) for evaluating mission execution, a K-Nearest Neighbors (KNN) algorithm for selecting a trajectory planner, Principal Component Analysis (PCA), and a Decision Tree (DT) to assess the health of the aircraft. Therefore, the novelty of Harpia focuses on robustness for autonomous planning and re-planning of the sequence of tasks and trajectories for regions of interest. The main contributions include an autonomous system architecture to plan missions with minimal human intervention, unconstrained by specific tasks, and computationally simple to operate in diverse scenarios. Computational tests report results for 220 simulated scenarios, in which Harpia adequately handled all situations, for example, making decisions about task re-planning with 97.57% accuracy based on battery health and choosing the best planning trajectory for each case with at least 95% accuracy.

Keywords: Robotics, Artificial Intelligence, UAV, Autonomous Systems.

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF SOURCE CODES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
BN	Bayesian Network
CEP	Complex Event Processing
CL	Classification Learner
CNN	Convolutional Neural Networks
DFD	Diagnostic Feature Designer
EKF	Extended Kalman Filter
FD	Fault Detection
FDD	Fault Detection and Diagnosis
FDI	Fault Detection and Isolation
FDMAE	Fault Detection Model Acceleration Engine
FPGA	Field-programmable Gate Array
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HGA	Hybrid Genetic Algorithm
ICAPS	International Conference on Planning and Programming
IFA	In-flight Awareness System
IFA^{2}S	In-Flight Awareness Augmentation System
IMU	Inertial measurement unit
INS	Inertial Navigation System
IPC	International Planning Competitions
K-NN	K-nearest neighbors
LIDAR	Light Detection and Ranging
LSTM	Long Short-Term Memory Neural Networks
MA-DPCA	Moving Average Dynamic Principal Component Analysis
MDP	Markov Decision Process
MFCCs	Mel-Frequency Cepstral Coefficients
MILP	Mixed-Integer Linear Programming
MKAD	Multiple Kernel based Anomaly Detection
MOSA	Mission Oriented Sensors Array
MTOW	Maximum take-off weight

NFZ	No-Fly Zone
PCA	Pearson Correlation Coefficient
PDDL	Planning Domain Definition Language
PFP	Potential Field Planning
POMDPs	Partially Observable MDPs
R-CNN	Region-based Convolutional Neural Networks
RC	Ray Casting
ROI	Regions of Interest
ROS	Robotic Operating System
RTT	Rapidly-exploring Random Trees
SLAM	Simultaneous Localization and Mapping
SOA	Service-Oriented Architecture
SVM	Support Vector Machines
UAV	Unmanned Aerial Vehicle

CONTENTS

CHAPTER

1

INTRODUCTION

1.1 Motivation

In recent years, Unmanned Aerial Vehicles (UAVs) have permeated various real-world applications, including agriculture, transport, logistics, and surveillance (ARFAOUI, 2017; Shakhatreh *et al.*, 2019; DENG *et al.*, 2018). Their increasing prominence is attributed to the declining cost of UAV construction and the embedded sensors tailored for specific tasks. The appeal of UAVs lies in their flexibility for data acquisition and cost-effectiveness (ZHANG; KOVACS, 2012) and, compared against terrestrial sensors or satellites, UAVs offer undeniable benefits. The recent advancements strengthened their processing capabilities, onboard sensors, obstacle avoidance autonomy, autopilot systems, and overall flight duration (BAVLE *et al.*, 2018; TABOR; GUILLIARD; KOLOBOV, 2018; RASTGOO *et al.*, 2018). The work in (Shakhatreh *et al.*, 2019) reviews several types of research about public and civil UAV applications, reporting the business chance that arises for manufacturers, investors, and business service providers. It is estimated a market value of $127 billion for UAV in civil scenarios, reaching more than 100,000 jobs related to unmanned aircraft operations by 2025.

While many studies propose computational architectures for UAVs, a gap exists for a modular, general-purpose system tailored for UAV mission execution and safety. The existing solutions are often specialized, lacking modular design and varied autonomy levels in decision-making. Integrating artificial intelligence resources for UAV planning and scheduling is still nascent and hinges on the specific application, besides being mandatory for an effective autonomous decision system. Complex issues must be addressed within UAV operations, such as sensor malfunctions, unexpected route obstructions, battery failures, and weather unpredictability. An autonomous system for a UAV must deal with those issues to guarantee flight safety, which can range from less drastic measures, e.g., a path re-planning, to hard decisions like emergency landing. The same re-planning measure can be applied for mission and path re-planning during the current mission execution once goals and tasks change. Thus, a robust autonomous sys-

tem architecture must fulfill real-time adjustments and intelligent task management as primary requirements.

Driven by these challenges, this doctoral project aims to craft an intelligent and autonomous general-purpose system for UAVs. The focus is on safety and mission execution, emphasizing system modularity and robustness. The motivation seeks to design a system that accomplishes mission without compromising aircraft autonomy and safety. A modular design ensures such flexibility, providing the necessary robustness to allow a smart adaptation when external measures change the mission or internal system errors must be mitigated.

1.2 Contextualization

The facilitated access to small electronic components has spurred the growth of the UAV industry, reaching a broader spectrum of professionals (VEMULAPALLI, 2019). From military exclusivity, UAVs now cater to small to medium enterprises, be it a local farmer monitoring crop growth or a construction company inspecting structural integrity. Current UAV mission planning tools are semi-automated once the users typically determine a sequence of waypoints and associated actions. Although these tools account for UAV-specific internal metrics, they often neglect external variables such as environmental changes or have problems dealing with specific aspects of some missions. For example, the *Mission Planner*[1] system used in (POBKRUT; EAMSA-ARD; KERDCHAROEN, 2016), like some analog systems like *QGround Control*[2] are widely used for agricultural applications. These systems allow the operator to access and analyze mission log files, configure autopilot settings, track the aircraft and its information in real-time, and create a click-through waypoint path using Google Maps/Bing images. Despite being robust systems, they do not have route calculation (they only work with a straight-line flight from point to point) and obstacle avoidance. Their security is based on *threshold*, which only triggers a safety measure if any variable exceeds its limit and mission actions are not re-configurable.

The mentioned systems, while robust, lack features like route optimization and obstacle avoidance. Their safety mechanisms are rudimentary, mission adaptability is limited, and onboarding novel mission protocols is complex. The current project will be contextualized in developing an autonomous system for planning and scheduling tasks or missions for UAVs. The tasks will cover various actions for different applications, and the system shall demand minimal human intervention. The project is also contextualized in the planning and scheduling tasks or missions within a structured and non-convex environment, where non-convexity stands for obstacles that must be avoided or regions prohibited for flight. Self-awareness is an inherent functionality to mitigate failures coming from internal systems. In Artificial Intelligence (AI), the project is contextualized as a study that demands AI techniques in sub-areas such as reasoning, planning, and scheduling aimed at autonomous decision-making in real-time.

[1] https://ardupilot.org/planner/
[2] http://qgroundcontrol.com/

1.3 Challenges and Goals

AI techniques have been employed to develop efficient UAV systems in (GONZALEZ *et al.*, 2016; RAMIREZ-ATENCIA *et al.*, 2017; STRUPKA; LEVCHENKOV; GOROBETZ, 2017) with the level of UAV autonomy ranging from a ground control system, with a responsible human pilot, to a flight fully autonomous. Autonomy for UAVs is challenging since it is expected or desired that the chances of failure are lower than those accepted for general aviation (WRIGHT, 2014). The development of systems that enhance the level of autonomy for UAVs, by embedding intelligent in-flight decision-making capacities, remains as a challenge. These autonomous systems must also be easy to use by non-expert users who can benefit from the autonomous aspects of their operation in day-to-day tasks. UAV missions are often constrained by time, geography, and communication (BODIN *et al.*, 2018). Integrating task planning into UAVs demands (CASHMORE *et al.*, 2015):

- Definition of a domain model aligned with the UAV's capability and the target environment.
- Ensuring real-time action execution by controllers.
- Adaptive response to environmental changes.

Thus, the research question stated by this thesis is: "Is it possible to develop a system that integrates AI capabilities for autonomous decision-making within different scenarios where real-time changes must be addressed with minimum human intervention ?"

Based on such question, the main goal of this thesis is:

- Designing autonomous and risk-aware UAV systems, leveraging reliable decision-making processes, which will be evaluated in different scenarios.

From this primary goal, this project has three technical aims:

1) **Planning:** The UAV must plan and re-plan mission and trajectories autonomously without violating the mission goals and the no-fly zones.
2) **Risk mitigation:** The UAV must be self-aware of its surroundings to avoid actions that put it at risk.
3) **Self-diagnose:** The UAV must be self-aware of its operation. In the event of a failure, the aircraft must be able to diagnose the faulty component and take safety actions to prevent damage.

1.4 Text Organization

This document is organized into chapters in such a way that:

- Chapter 2: reviews of works related to the approaches proposed in this project.
- Chapter 3: describes the problem addressed.
- Chapter 4: presents the foundations and basic concepts used in this project.
- Chapter 5: presents the methods and methodologies used to address the proposed problems and architecture.
- Chapter 6: reports the results obtained.
- Chapter 7: contain the final remarks of this project.
- Appendix A: presents the generated tree for this project.
- Appendix B: presents complete PDDL code.

CHAPTER

2

LITERATURE REVIEW

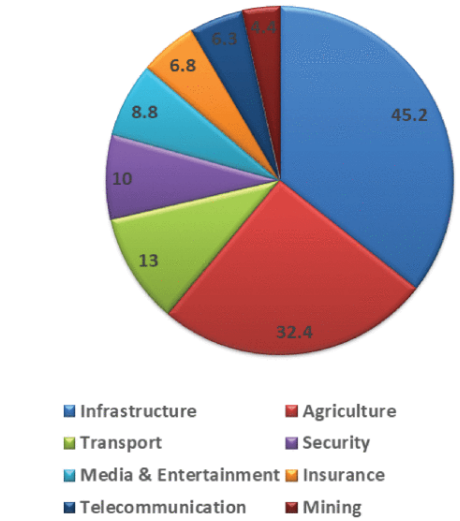
2.1 UAV Applications

The use of UAVs in precision agriculture has been advancing during the last decade with the increased use of AI techniques. For example, the recent work in (MA *et al.*, 2022) reports a wheat ear counting method employing UAV, where transfer learning from the ground-based counting model improves the overall wheat ears counting. The works in (KIM *et al.*, 2019) and (RADOGLOU-GRAMMATIKIS *et al.*, 2020) review research about using UAV in precision agriculture. The authors in (KIM *et al.*, 2019) point out the limitations of UAV systems for agriculture applications, where the mission and path planning must optimize battery and flight time to accomplish more tasks in crop fields. The review in (RADOGLOU-GRAMMATIKIS *et al.*, 2020) reports 20 UAV applications for aerial crop monitoring or spraying tasks, analyzing the UAV systems where the majority presents an architecture design focused on specific purpose tasks for monitoring and spraying. The authors identify the need for decision support systems. Both (KIM *et al.*, 2019) and (RADOGLOU-GRAMMATIKIS *et al.*, 2020) mention the lack of systems handling autonomous operation, decision-making, and risk mitigation for UAVs in precision agriculture.

In the context of civil applications, the work in (Shakhatreh *et al.*, 2019) reviews UAV applications in remote sensing, construction inspection, precision agriculture, delivery of goods, monitoring of road traffic, wireless coverage, and surveillance. It is estimated as a market for UAV figures around $127 billion, where $45 and $32.4 billion are in civil infrastructure and agriculture, respectively, with relevant values also estimated for transport, media, and telecommunication in Figure 1. This leads to a growth in the global UAV payload market (radars, sensors, cameras, LIDARs, among others) that can achieve $3 billion by 2027, as shown in Figure 2. The authors highlight the need to have UAVs with embedded software and hardware that can perform remote sensing in parallel with monitoring possible threats during flight. Mission execution requires methods for mission planning, where the UAV becomes capable of dealing with potential risks

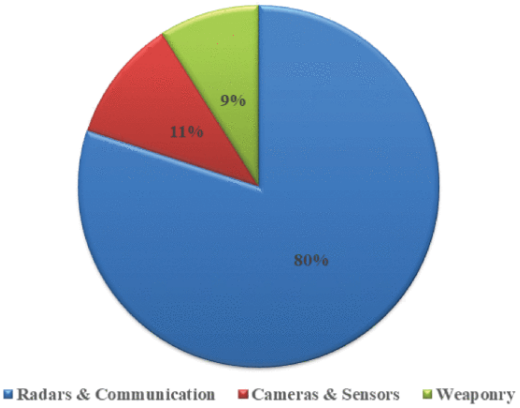
without human intervention.

Figure 1 – Estimated values in billion for UAV application in industries.



Source: Shakhatreh *et al.* (2019).

Figure 2 – Estimated values by payloads.



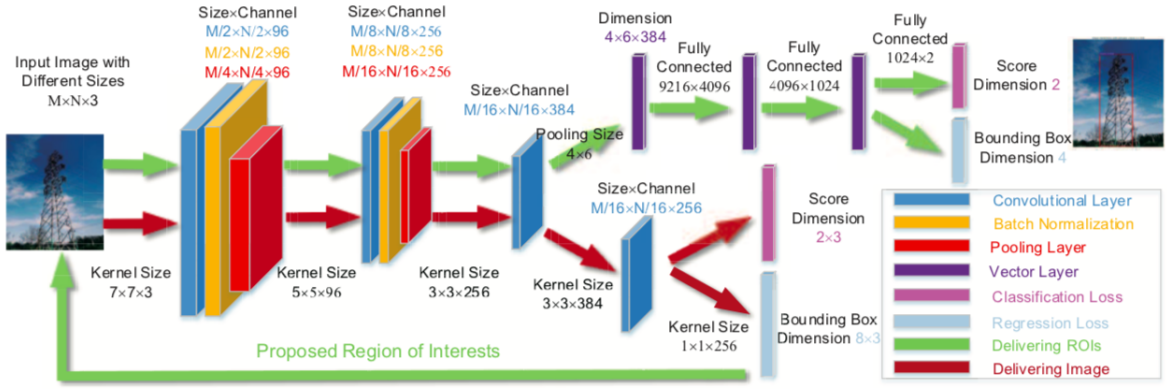
Source: Shakhatreh *et al.* (2019).

An example of a civil application is a building inspection. In the article, (JUNG *et al.*, 2018), a route planning algorithm for inspecting tall structures is developed. The developed algorithm separates the structure into layers according to their height; for each layer, the method calculates a specific route according to the samples needed to reconstruct the structure. Another example in (DENG *et al.*, 2018) is a route planner for monitoring defects in water channels, where the planner considers the collaboration of a terrestrial robot with the aircraft to cover the extension of the channels to be monitored. Another possible application for UAVs is their use in hazardous environments, such as industrial leaks. The authors in (MENG *et al.*, 2018) report the use of UAVs in a hazardous environment with the aircraft automatically executing the action of press a safety button in a difficult-to-access location. Another scenario using UAVs is firefighting, as described in (HAKSAR; SCHWAGER, 2018), where the authors present a distributed system that uses Monte Carlo to simulate multiple aircraft within the fire environment.

Deep reinforcement learning generates a decentralized solution that improves the solutions regardless of the number of aircraft and the size of the forest.

To deal with the inspection of transmission networks, the authors in(BIAN *et al.*, 2018) propose Region-based Convolutional Neural Networks (R-CNN), shown in Figure3, for the identification of towers through images. The UAV distance is calculated by identifying the towers, and a flight path is obtained based on a fixed distance. A Convolutional Neural Networks (CNN) is also presented in (KOURIS; BOUGANIS, 2018) for autonomous flight, where the C-NN learns to move around without colliding with obstacles indoors and without needing a map.

Figure 3 – R-CNN schematic



Source: Bian *et al.* (2018).

Some research focuses on more specific tasks related to the UAV itself. The work described in (HAMAZA; GEORGILAS; RICHARDSON, 2018) focuses on improving the aerodynamics of flight according to the study of UAV physics and its sensors. The authors at (MOHAIMENIANPOUR; VAUGHAN, 2018) create a neural network for image and motion detection, where the focus is controlling the aircraft through commands identified by the captured images. However, no real-world application is reported using UAVs. The authors at (SARTORI *et al.*, 2018) exclusively study estimating aircraft location in real flights on external environments, improving its dynamics through mathematical modeling. The states are estimated with a filter based on Extended Kalman Filter (EKF), fed with data from low-cost UAV sensors, such as compass and barometer. In the work of (SVACHA *et al.*, 2018), a filter is developed to define the position of the UAV in the three-dimensional axes, using only information from the Inertial measurement unit (IMU) and data yaw. The authors propose a Riemannian unscented Kalman filter for estimating aircraft altitude and speed. The system is tested in a closed environment with 20 Vicon cameras [1] to confirm the measures proposed by the filter.

[1] http://www.vicon.com/

The developed algorithm in (ROELOFSEN; GILLET; MARTINOLI, 2015) predicts the movement of the aircraft being viewed to calculate a diversion route; the authors use computer vision to detect and avoid other aircraft. The disadvantage is that the system only identifies other UAVs through an attached marker. For the system to find and divert an aircraft, it needs to have a marker, which makes it impossible to avoid unknown aircraft. In a similar application, the work in (SAPKOTA *et al.*, 2016) uses only a monocular camera to identify other aircraft by estimating the position of the object found.

Another application is the control of multiple UAVs. The authors in (CHUNG *et al.*, 2016) simultaneously create a take-off, flight, and landing control system for 50 aircraft. In flight, one of the UAVs is chosen as the leader and given a route, while the other aircraft are given the command to follow it. The work in (KENMOGNE; DREVELLE; MARCHAND, 2018) describes an approach based on position estimation with multi UVAs cooperation from acquired images. The method presents better results when compared to the most commonly used method: EKF. Another work dealing with the trajectory for multiple UAVs is (FALOMIR; CHAUMETTE; GUERRINI, 2018), which applies a potential fields algorithm for path planning that avoids obstacles previously known.

The primary orientation sensor of UAVs is the GPS; however, in some applications, it is desired to fly indoors where the GPS does not work. Thus, studies have focused on developing systems that do not depend on this sensor. In (BAVLE *et al.*, 2018), a particle filter is proposed to estimate location, using Simultaneous Localization and Mapping (SLAM) and *deep-learning*. The authors in (GHASEMLOU; OKANE; SHELL, 2018) study a sensor's feasibility, usability, and importance in case of damage in flight. The authors used label maps to model each modification in the set of sensors and actuators. As this map grows exponentially, it was necessary to sample all the maps and use a decision tree classifier. The classifier determines if a branch is not critical and can be deleted, returning a tree with the information about observations/actions that are most important. The work in (CARRIO *et al.*, 2018) reports an embedded system for obstacle avoidance that collects the depth maps and, with the help of *deep-learning*, learns to dodge obstacles for different types of aircraft. One of the advantages of the developed algorithm is to handle trajectories within 3D maps. The authors in (ARANTES *et al.*, 2016) apply a hybrid method combining genetic algorithms with Voronoi diagrams to solve the non-convex path planning with risk allocation. The technique finds satisfactory solutions within a few seconds, generating trajectories to avoid many obstacles within different scenarios.

The authors in (ARANTES *et al.*, 2019) introduce a Mixed-Integer Linear Programming (MILP) formulation that finds optimal solutions for complex maps within a reduced computational time. Harpia will apply metaheuristics approaches once the systems must find solutions within a short time. Model control predictive is applied by (LUIS; VUKOSAVLJEV; SCHOELLIG, 2020) to develop a parallel method for path planning of multiple robots handling collision avoidance and re-planning in real-time. The authors evaluated the system through

Matlab simulations with satisfactory results when mitigating collisions within short periods. A hybrid evolutionary algorithm is described in (SOUZA; TOLEDO, 2020) without handling the chance constraint with risk allocation. The method combines the evolutionary approach and the ray casting technique for obstacle avoidance, reporting a satisfactory performance for path planning within a short time.

The authors in (CAMPO; LEDEZMA; CORRALES, 2020) propose a low-cost UAV system for crop data acquisition, handling issues such as wind and battery consumption. A path planning system optimizes the coverage paths, comparing wavefront, Dijkstra, and spiral algorithms. A task planning system for UAV in (SUN; WANG; ZHANG, 2020) executes plant protection tasks, e.g., sowing or pesticide spraying. Dragonfly algorithm is applied to achieve near-optimal schedules for a set of 20 instances generated from real-world data. In (LINDQVIST *et al.*, 2022), it is presented a hardware and software architecture for a legged-aerial autonomous system applied to missions in underground areas. In the autonomous aerial system, an artificial potential field formulation and the so-called deepest-point heading regulation technique, employing a 3D Light Detection and Ranging (LIDAR) for clustering points, allows an efficient reactive behavior for obstacle avoidance. UAV is transported by the legged ground robot, which uses a SLAM system to be aware of its localization. It is a complex system framework whose reported results show its capability to execute missions autonomously within underground areas.

The work in (SONG; PARK; PARK, 2022) approaches the task assignment for UAVs by introducing a multi-objective mathematical model, where system design aspects such as the location of base stations, number of UAVs in each station, and UAV schedules for mission execution are also defined. The proposed solution must work under a disaster management situation. Therefore, a two-phase method is applied to deal with the computational complexity and improve accuracy when solving the proposed multi-objective model.

2.2 Architectures

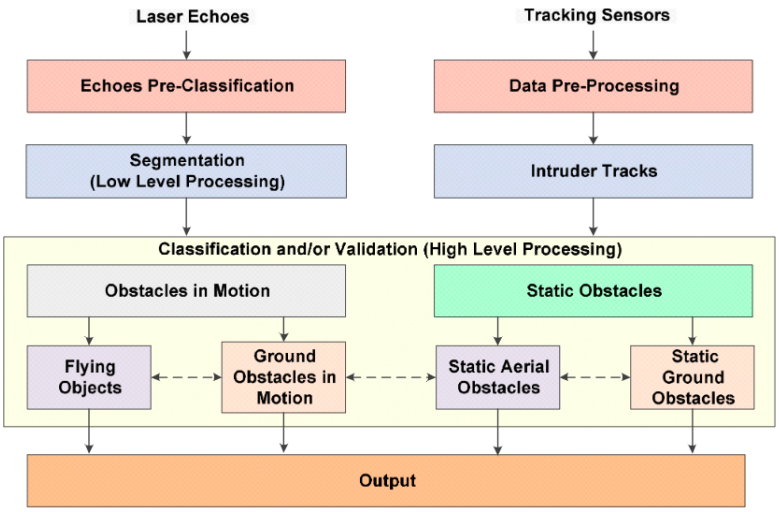
Some works in the literature present hardware/software architectures for UAVs. An architecture is proposed in (GUNETTI; DODD; THOMPSON, 2010) for UAVs using parallel and associative memory, preference-based deliberation, belief maintenance, goal decomposition, and adaptation through a generalization from experience. These aspects provide a robust architecture for autonomous flight. Three agents are defined and called planner, executor, and mission manager, allowing them to perform actions such as target analysis, target attack, orbit in a position, search in an area, and transit between points. These functionalities are tested in simulation software.

The authors in (BADRA; AïELLO; CHAUDEMAR, 2023) explore the modeling of Unmanned Aerial Vehicle (UAV) missions using a Model-Based Systems Engineering (MBSE) approach. The complexity of UAV missions requires new techniques for effective design and

understanding. The paper employs a bottom-up approach, starting with a specific surveillance mission modeled in Simulink Stateflow. Through semantic clarification and multiple simulations, a generalized model is developed that ensures consistency and reusability for any UAV mission. The ARCADIA/Capella methodology is applied for operational, functional, logical, and physical analyses, providing a comprehensive multi-viewpoint representation of the UAV mission.

A predictive control system is described in (PRODAN *et al.*, 2013) for UAV trajectories, considering external disturbances to the flight. The control system has two main modules: the faster one runs incorporated in the UAV, while the slower one runs from the ground station. The onboard system controls the UAV's dynamics, and the ground station system plans the entire trajectory. Another architecture is presented in Figure 4 for UAV trajectory control as described in (RAMASAMY *et al.*, 2016), based on a laser obstacle warning and avoidance system. The developed architecture allows the perception and avoidance of obstacles, having three main algorithms: one focused on trajectory prediction, another dedicated to estimating the collision of detected obstacles, and the third focused on collision prevention planning.

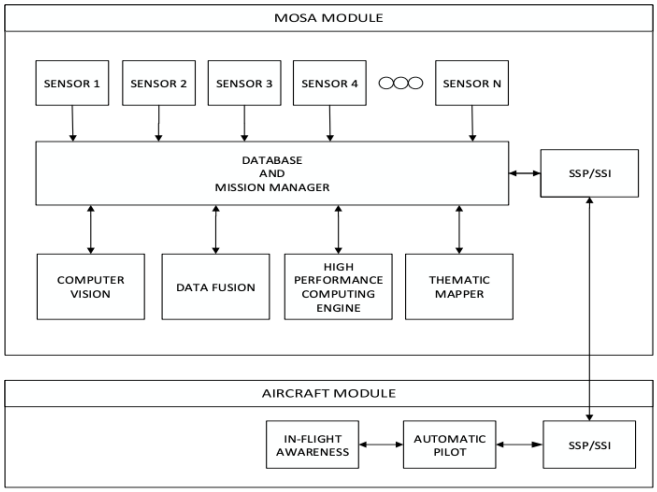
Figure 4 – Trajectory control architecture



Source: Ramasamy *et al.* (2016).

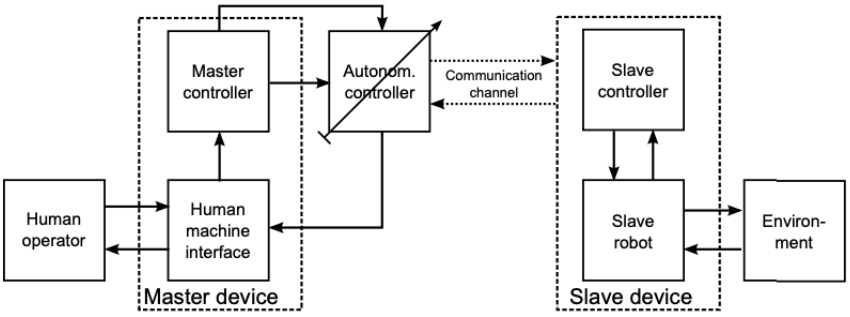
The Mission Oriented Sensors Array (MOSA) system is introduced as being responsible for the trajectory execution in (FIGUEIRA *et al.*, 2013) as illustrated in Figure 5. MOSA can dynamically adjust itself to mission characteristics, choosing sensors that are more adapted to each situation. It is a system focused on fulfilling the mission. The authors in (FIGUEIRA *et al.*, 2015) apply MOSA as a reference to design a system that automatically produces thematic maps. A semi-autonomous architecture is proposed in (WOPEREIS *et al.*, 2015) as shown by Figure 6. The main objective is to have variable safety distances with the user's permission. Thus, allowing the aircraft to reach the destination becomes possible, avoiding obstacles where this would not be possible without increasing the risk of collision. The authors in (XUE *et al.*, 2016) present a hardware/software architecture for automatic navigation and spraying in crop fields. In this

Figure 5 – Trajectory control architecture



Source: Figueira *et al.* (2015).

Figure 6 – Bi-lateral architecture overview

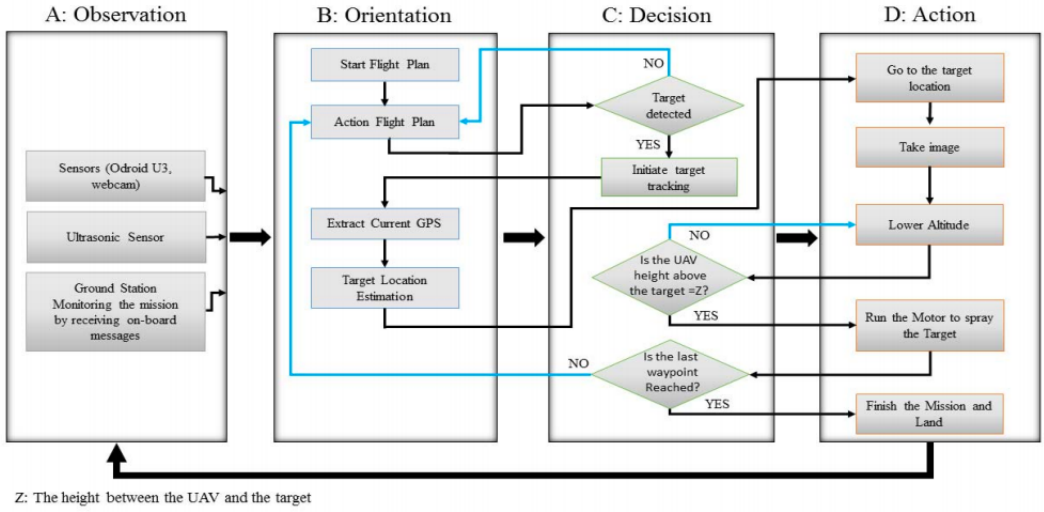


Source: Wopereis *et al.* (2015).

case, the flight and spray control systems are built into an uncrewed helicopter. However, path planning is done from the ground station that reports mission data to flight control, which will follow the waypoints and send a signal to start the spray control system. The onboard computer reports the real-time status of the spray system to the ground station.

A generic system, based on computer vision, is described in (ALSALAM *et al.*, 2017) to control the UAV in applications such as spraying in agricultural fields and pest detection, among others. The current waypoints may change in these scenarios, and path corrections are performed autonomously during flight. The proposed level of autonomy for the onboard system makes it possible to detect the location of weeds or pests and spray locally instead of passing through the entire crop field. The decision-making system is shown in Figure 7. The system encodes an Observation, Orientation, Decision, and Action loop. The hardware is designed to control a quadcopter using a companion computer Odroid U3+ integrated autopilot (Pixhawk) and an Arduino micro. Robotic Operating System (ROS) packages are also used to develop navigation, camera, and sensing nodes.

Figure 7 – Proposed decision flow



Source: Alsalam *et al.* (2017).

The UAV architecture proposed by (BOUBETA-PUIG *et al.*, 2018) supports real-time decision-making, which integrates Complex Event Processing (CEP) with a Service-Oriented Architecture (SOA). The system handles critical situations in which the CEP processes and evaluates many events in real time. SOA allows the system to handle data from different sources and domains. The hardware architecture uses a DJI F-450 UAV, ArduPilot APM 2.6 autopilot, a Raspberry Pi 2 companion computer, and message queue telemetry transport. The authors (YAN; LI; CHEN, 2018) introduce a hardware/software architecture for the navigation system focused on redundancy. The hardware integrates two high-performance DSP-type FPGAs and various subsystems, such as the Inertial Navigation System (INS) and the Global Navigation Satellite System (GNSS). A dual navigation computer and a fault-tolerant filtering algorithm are applied. The system uses a federated filter, an ideal filter for decentralized merging based on information sharing. This filter is robust and used in the described two-level system for the navigation system, with two subfilters for location sensors, using EKF to merge the outputs of the subfilters to obtain the final result of the filter. Thus, one or more filters are dedicated to sensors operating in parallel and a primary fusion filter.

The work in (USACH *et al.*, 2018) focuses on safety and presents an architecture for remotely piloted aircraft. The architecture differentiates event monitoring and contingency decision-making by evaluating the best contingency management policy through formal methods. Three classifications are presented. The first is Fault Tolerance, where there is a tolerance for the aircraft to continue flying in non-ideal conditions. The second is Risk Mitigation, where safeguards are added to lessen the severity or probability of the consequence of the failure. The third is Flight Termination, which means immediately aborting the flight in the most severe failure cases. The work proposes an architecture according to DO-178[2] with four main modules:

[2] Compliance Testing | Expert Certification Support

- *Safety Monitor* that monitors the occurrence of 5 failures: failure in communication with the radio, loss of GPS, loss of aircraft control, traffic alert, and alert of violation of mission regions (NFZ).

- *Contingency Manager* who chooses a security measure to lessen the risks.

- *Mission Manager* which is the executor of actions.

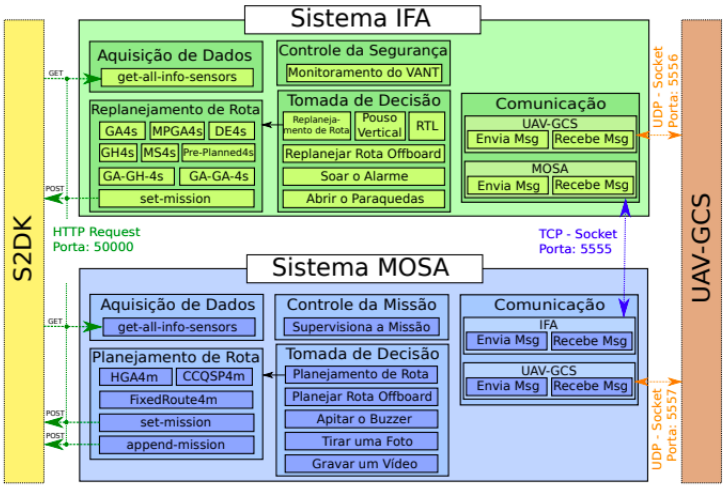
- *Flight Termination Systems* which is the system capable of triggering safety measures in any situation on the aircraft.

The authors of (POIESI; CAVALLARO, 2018) present a communication architecture between multiple UAVs, aiming to follow a single target by sharing the computational vision. The work also presents the dynamics of multiple UAVs when flying together without GPS. For the aircraft to fly without colliding, each uses its view centered on a target and maintains the relative distance of its two nearest neighbors to deduce its steering commands. The authors of (SAMPEDRO *et al.*, 2018) developed an architecture for indoor environments that combines laser sensor readings with *deep reinforcement learning*. The neural network receives the distance data filtered from the laser and in its output layer and returns linear speeds to the aircraft. The objective is to create a reactive flight for rotary wing UAVs with two main focuses: reaching the objective waypoint and ensuring real-time obstacle avoidance.

A Service-Oriented Architecture is implemented in (ARANTES, 2019) with different mission and security modules as illustrated in Figure 8. These modules are based on the definitions presented in (FIGUEIRA, 2016; MATTEI, 2015). Mission-Oriented Sensor Array (MOSA), as already mentioned, is a system focused on mission with the function of controlling four main aspects in SOA: (i) the behavior of the mission according to its objective; (ii) the input of data from the sensors; (iii) the planning of routes with obstacle avoidance; (iv) the communication with other subsystems. In-flight Awareness System (IFA) is a multiple failure diagnosis system described in (MATTEI, 2015), where simulation results with SimuLink are reported. In (ARANTES, 2019), MOSA and IFA systems were implemented in Java, with IFA having priority over the MOSA system, which means that IFA takes control of the mission when a failure is identified. The failure cases analyzed were battery problems, weather conditions, GPS failure, MOSA system corruption, and autopilot non-functioning. These failures were analyzed by violating previously defined thresholds for specific security parameters.

The work in (MATTEI *et al.*, 2021) extends the IFA, proposing a novel onboard system called In-Flight Awareness Augmentation System (IFA^{2}S) to improve flight safety. The system introduces a top-down control structure and a state machine, as shown by Figure 9 to allow IFA^{2}S solutions assessment. IFA^{2}S stays in an Idle state, waiting for an event that leads to a state change based on the likelihood of hazards from a range of parameter values. The authors

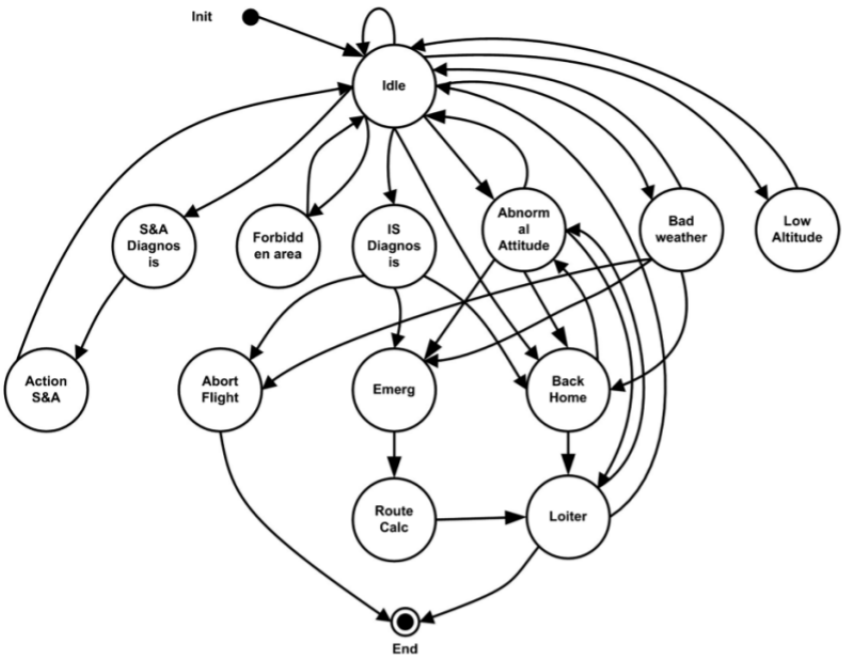
Figure 8 – Architecture presented



Source: Arantes (2019).

simulated the system considering threat situations like a failure in aircraft systems, air collision, bad weather, low or abnormal altitude, flight over non-fly zones, and emergency landing.

Figure 9 – State Machine in IFA^2S



Source: Mattei *et al.* (2021).

2.3 Safety and Security

The work in (HIRECHE *et al.*, 2018) explores the application of Bayesian Network (BN) in fault diagnosis and decision-making for autonomous systems. The article introduces various

case studies, including monitoring GPS systems for accuracy and reliability, tracking applications with error mitigation, and resource-aware monitoring for optimizing performance. It delves into learning BN parameters from incomplete data. The article also investigates decision-making mechanisms' implementation based on BN models, presenting speed-up results and resource utilization for FPGA platforms. Ultimately, it suggests a co-design approach for integrating hardware-accelerated diagnosis with software-based decision-making for mission managers in autonomous systems, emphasizing adaptability and real-time responsiveness within swarming UAV networks.

The real-time detection and identification of the root causes of operational anomalies in UAVs have become essential to prevent mishaps and ensure post-event forensic analysis. In (SADHU; ZONOUZ; POMPILI, 2020), the authors developed novel deep learning architectures, harnessing the power of CNNs and Long Short-Term Memory Neural Networks (LSTM) to detect and classify drone miss-operations based on raw sensor data. Unlike traditional model-based methods, these architectures focus on a data-driven approach, utilizing UAV IMU sensor data. Such methods are advantageous as they can decipher complex, nonlinear patterns within the sensor data without manual feature design, providing insights into UAV behavior under potential crash events.

The research in (WANG *et al.*, 2020) addresses the safety concerns of unmanned aerial vehicles, particularly their higher accident rate due to the absence of real-time pilot control, introducing the Fault Detection Model Acceleration Engine (FDMAE). This solution combines the deep learning capabilities of Long Short-Term Memory (LSTM) networks with a model pruning technique based on Principal Component Analysis (PCA) to optimize computational efficiency. The pruned model is integrated into an airborne Field-Programmable Gate Array (FPGA) platform, offering a promising avenue for real-time UAV fault detection while considering the size, weight, and power consumption constraints inherent in UAV applications.

In (ALOS; DAHROUJ; DAKKAK, 2020), the authors introduce an innovative unsupervised algorithm designed to evaluate UAV behavior and thereby enhance its safety. The extraction of values that characterize the relationships between pairs of UAV variables is central to their approach, explicitly focusing on the Pearson Correlation Coefficient (PCA), the Y-intercept, and the linear regression slope. The algorithm employs a PCA-based anomaly detection system to pinpoint abnormal flights and the variables contributing to potential faults. The method was experimentally tested on the synthetic dataset and demonstrated comparable efficacy against the state-of-the-art Multiple Kernel based Anomaly Detection (MKAD) method. The proposed method did not need a sizable training dataset, which underscores the potential advantages of unsupervised algorithms in UAV anomaly detection.

In their research on mission-critical autonomous flights, the authors in (AHMAD *et al.*, 2022) propose an intelligent framework for automated failure prediction, detection, and classification. They emphasize the importance of ensuring the safety and reliability of aerial

vehicles, typically addressed through redundant hardware, and highlight the value of analytical redundancy. The study introduces a multi-layered machine learning pipeline that processes the ALFA dataset containing flight sensor data. This pipeline consists of data transformation, preprocessing, feature selection, and LSTM-based failure prediction, followed by classification using sliding window-based data aggregation and decision trees. The research emphasizes the importance of selecting relevant features for accurate failure prediction, with LSTM networks chosen due to their efficacy in handling time series data. The framework successfully detected failures with 100% accuracy in tests and demonstrated the capability to predict failures on average within one-tenth of a second of their occurrence.

The research in (BASKAYA; BRONZ; DELAHAYE, 2017) delves into the challenge of Fault Detection and Diagnosis (FDD) for small UAVs in an environment increasingly populated with these devices, necessitating enhanced safety measures. Noting the hardware limitations of UAVs, the authors recommend analytical redundancy over traditional hardware redundancy. Their study thoroughly examines both model-based and data-driven FDD methods. While model-based methods rely on accurate aircraft models and might be unsuitable due to uncertainties in small UAVs, data-driven methods, particularly machine learning, emerge as attractive alternatives. They employ Support Vector Machines (SVM) complemented by Principle Component Analysis (PCA) to distill the data by reducing feature space dimensions and optimizing the detection process. Using simulated gyro and accelerometer data, they demonstrate SVM's proficiency in distinguishing between standard and faulty flight conditions. With PCA aiding in data simplification and feature prioritization, their findings validate SVM's precision in fault detection. The study concludes with an emphasis on further exploration into controller diagnosis interaction, multi-fault classification, and SVM's real-time training possibilities.

The work in (LI; LI; ZHONG, 2016) emphasizes the significance of Fault Detection and Isolation (FDI) for ensuring UAV flight safety. Recognizing the challenges in constructing precise physical models for UAV flight control due to variable parameters and wind disturbances, the paper highlights the limitations of model-based FDI methods, which often require accurate models for effective implementation. The authors propose a data-driven FDI scheme using Moving Average Dynamic Principal Component Analysis (MA-DPCA) to address these limitations. This scheme captures dynamic relations between system variables for enhanced fault detection from the historical UAV data. For fault isolation, they introduce the concept of contribution analysis. Simulations using a fixed-wing UAV in a steady flight state validate the efficacy of the MA-DPCA approach, particularly highlighting its prowess in fault detection. However, the method's current limitation to steady flight states is acknowledged, with ongoing research aimed at extending its applicability to maneuver flights.

The authors in (KHAN *et al.*, 2019) propose real-time anomaly detection for (UAVs), which challenge traditional methods reliant on well-defined features by embracing data-driven techniques. They highlight the utility of the isolation forest algorithm, evidenced by its exemplary

performance with the Aero-Propulsion System Simulation dataset and real-time UAV experiments. A pivotal Principal Component Analysis (PCA) analysis revealed specific anomalies during UAV take-off and hovering. Assisted by the visualization power of the violin plot, they offered a detailed statistical examination of these anomalies. This research underscores the imperative for adaptive, real-time anomaly detection strategies in the UAV sphere, combined with the analytical prowess of PCA.

In (BALDINI *et al.*, 2023), the authors introduced UAV-FD, an innovative dataset designed to foster the advancement and verification of Fault Detection (FD) algorithms tailored for multi-rotor drones. Utilizing MATLAB for data processing, they harnessed the Diagnostic Feature Designer (DFD) for feature extraction and subsequently trained an array of classifiers using the Classification Learner (CL). A significant aspect of their approach was the application of Principal Component Analysis (PCA) to condense the feature space. Despite this reduction, non-linear classifiers, particularly the Support Vector Machines (SVMs), demonstrated notable accuracy. The research spotlighted a quadratic SVM classifier, fine-tuned via ANOVA on a chosen set of 51 features, which achieved an impressive 98.5% accuracy. The study also delved into the utility of a binary classification tree, revealing key influential variables. Recognizing the imperative to mitigate false positives in aeronautics, the authors discussed the computational intricacies faced in real-world UAV applications.

(KOŁODZIEJCZAK *et al.*, 2023) incorporated Principal Component Analysis (PCA) and tree algorithms to streamline and enhance the fault detection process, comparing model-based approaches with data-driven ones. The tree algorithms provided structured decision-making based on feature importance derived from the acoustic data. The study highlighted potential pitfalls in real-world applications, such as unrealistic assumptions and extended data frames, and advanced the acoustic FDI system for drones, particularly emphasizing the significance of Mel-Frequency Cepstral Coefficients (MFCCs) in fault detection.

2.4 Conclusion

Harpia approaches risk mitigation as (HIRECHE *et al.*, 2018) by applying Bayesian Network to decide autonomously about re-planning based on battery health. The path planning system in Harpia will also deal with obstacle avoidance, executing online path re-planning. However, we are not assuming a dynamic scenario like the one described in (LUIS; VUKOSAVLJEV; SCHOELLIG, 2020). Harpia deals with non-fly zones similar to (MATTEI *et al.*, 2021), but without defining all those risk requirements or using a state machine for autonomous behavior. Our autonomous system avoids obstacles or non-fly areas by solving a non-convex path planning problem similar to (ARANTES *et al.*, 2016), (ARANTES *et al.*, 2019) and (SOUZA; TOLEDO, 2020).

The system proposed in this thesis will generate mission planning aiming at autonomous

execution such as (CAMPO; LEDEZMA; CORRALES, 2020; SUN; WANG; ZHANG, 2020). However, we advance from these previous works by including Bayesian Networks (BN) and K-nearest neighbors (K-NN) classifiers to improve mission and path planning decisions. Different from (SONG; PARK; PARK, 2022), we are not using mathematical formulation aiming to reduce the computational complexity of the overall embedded system.

This work proposes a real-time fault identification based on data analysis as proposed in (SADHU; ZONOUZ; POMPILI, 2020; WANG *et al.*, 2020; KHAN *et al.*, 2019). Previous research efforts, such as (SONG; PARK; PARK, 2022), have a PCA algorithm primarily focused on leveraging mathematical formulations to reduce computational complexity. In contrast, our work resembles (KHAN *et al.*, 2019), using PCA to increase the accuracy of fault detection and identification. One of the main differences between this work and (KHAN *et al.*, 2019) is that it uses data from a specific sensor, and our project uses native information from the auto-pilot, returning a more reusable result.

The systems in (PRODAN *et al.*, 2013; RAMASAMY *et al.*, 2016; XUE *et al.*, 2016; ALSALAM *et al.*, 2017; MA *et al.*, 2022; ZHANG; LI; DONG, 2022; LINDQVIST *et al.*, 2022) accomplish a specific type of task, where considerations about planning several tasks are not addressed. Harpia proposes a robust planning system where different tasks can be autonomously executed.

Table 1 summarizes the main features covered by Harpia against those reviewed in this chapter. **Mission** means those papers whose UAV systems deal with Mission Planning autonomously, and the same idea applies to **Path** for works reporting systems with Path Planning. It is also listed papers in **Risk Mitigation** for those impending mission failure or path hazards, **Non-Convex** for result in environments with obstacles or non-fly zones, and **Fault Detection** means those papers whose UAV can identify a hardware failure from internal sensors.

Table 1 – Contributions with related work based on some specific features.

Reference	Mission	Risk Mitigation	Path	Non-Convex	Fault Detection
(MA *et al.*, 2022)	No	No	Yes	No	No
(SONG; PARK; PARK, 2022)	Yes	No	No	No	No
(LINDQVIST *et al.*, 2022)	Yes	No	Yes	Yes	No
(ZHANG; LI; DONG, 2022)	No	Yes	Yes	Yes	No
(MATTEI *et al.*, 2021)	No	Yes	Yes	Yes	Yes
(SOUZA; TOLEDO, 2020)	No	Yes	Yes	Yes	No
(CAMPO; LEDEZMA; CORRALES, 2020)	No	No	Yes	No	No
(SUN; WANG; ZHANG, 2020)	Yes	No	Yes	No	No
(ARANTES *et al.*, 2019)	No	Yes	Yes	Yes	No
(HIRECHE *et al.*, 2018)	Yes	Yes	No	No	Yes
(ALSALAM *et al.*, 2017)	Yes	No	Yes	No	No
(ARANTES *et al.*, 2016)	No	Yes	Yes	Yes	No
(RAMASAMY *et al.*, 2016)	No	Yes	Yes	Yes	No
(XUE *et al.*, 2016)	Yes	No	No	No	No
(PRODAN *et al.*, 2013)	Yes	No	Yes	No	No
(WANG; WILLIAMS, 2015)	No	No	No	No	Yes
(ALOS; DAHROUJ; DAKKAK, 2020)	No	No	No	No	Yes
(SADHU; ZONOUZ; POMPILI, 2020)	No	No	No	No	Yes
(AHMAD *et al.*, 2022)	No	No	No	No	Yes
(BASKAYA; BRONZ; DELAHAYE, 2017)	No	No	No	No	Yes
(LI; LI; ZHONG, 2016)	No	No	No	No	Yes
(KHAN *et al.*, 2019)	No	No	No	No	Yes
(BALDINI *et al.*, 2023)	No	No	No	No	Yes

CHAPTER

3

PROBLEM APPROACHED

This chapter describes the problem addressed by considering two aspects. First, to facilitate understanding, the main aspects of the problem are described in the context of an application in agriculture. Next, the problem is described by characterizing the sub-problems that must be addressed to develop the proposed autonomous system.

3.1 General Description

Suppose a drone's actions on a farm include plantation imaging and spraying biological agents, as shown in Figure 10a. The regions R[1...5] represent spraying areas, and the colors indicate the type of biological agent to be applied. The monitoring and spraying actions may require multiple flights, which also demand some stop-by support base areas B[1...3] to recharge the battery, fill the tank with biological agents, or process images.

The re-planning of actions can happen online for an unforeseen situation, such as adding or excluding some actions by the user or an action for risk mitigation. The path planning occurs within a static and non-convex scenario. It means we know the map, its areas of interest, and no-fly zones in advance. This is a pretty realistic scenario when handling drones on farms. Thus, besides the mission planning problem, we are also solving a non-convex path planning problem with stay-in and stay-out areas similar to (ARANTES *et al.*, 2019).

In the presence of a reliable communication link between the ground station and the embedded system of the UAV, it is possible to update the information about the map and mission in real-time. Our system covers these aspects when planning tasks and paths, reaching a reasonable level of self-awareness for the UAV regarding its surroundings and internal functioning. The present research focuses on a system designed for autonomy in dealing with external and internal events. We illustrate that aspect in our current architecture's implementation for a scenario where the following constraints must be handled: (i) battery needs a recharge (internal event);

(a) Map 1

(b) Map 2

(c) Map 3

Figure 10 – Farm scenarios for planning tasks: regions of interest, no-fly zones and support bases.

(ii) obstacle avoidance must be handled (external event), (iii) trajectories must be re-planned when new goals are added or removed from the current plan (external event); (iv) measures must be taken (emergency landing or return to base) when the system identifies an anomalous flight pattern that could be dangerous to the aircraft or the environment (internal event). On the other hand, the details about how a specific sensor should be implemented or integrated into our architecture are not the main points of this work.

In this scenario, an autonomous UAV system must satisfy some basic requirements:

1) The UAV must autonomously plan and re-plan the execution of such tasks.

2) The path planning has to be done without violating non-fly zones.

3) The UAV has to be aware of its surroundings and avoid actions that put itself at risk.

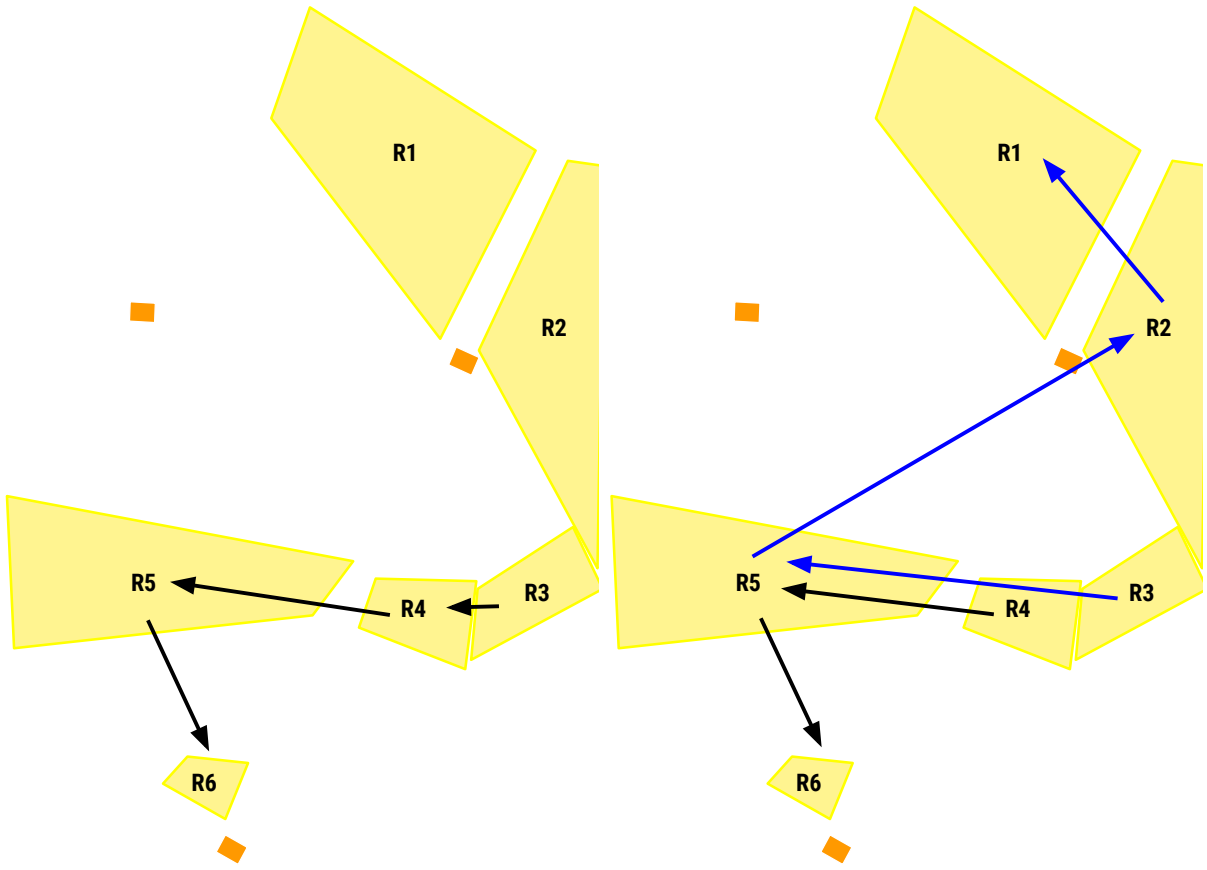
4) The UAV must execute safety actions to avoid damages.

Two types of planning are considered here: mission planning and path planning. The mission planning will define the sequence of execution for each task within the areas of interest. In this work, the following requirements must be fulfilled for mission planning:

1) The non-fly zones, bases, and areas of interest are previously known.

2) Non-fly zones, bases, and areas of interest can be added or removed during the mission execution, demanding a mission re-planning as illustrated by Figure 11.

3) The mission planned can be aborted when the autonomous system takes the safety measures.



(a) Current Mission Planning (b) Mission Re-Planning

Figure 11 – Re-planning sequence of actions. The black arrows show the previous sequence of actions, and blue arrows indicate the re-planning one with the removed and added regions.

Figure 11a shows the current mission plan with a mission re-planning demanded when the UAV was in region R3. At this point, regions R4 and R6 were removed from the current plan, with R1 and R2 being added, leading to the re-planning shown in Figure 11b.

The path planning defines the trajectory with two types of routes, named specific routes and routes between regions. The specific route defines a pattern customized to be executed when a specific task is executed. For example, a patter of trajectory that allows one to take pictures or spray insecticides in a given region of interest, as illustrated in Figure 12.

Figure 12 – Specific routes example

(a) Imaging route

(b) Pulverizing route

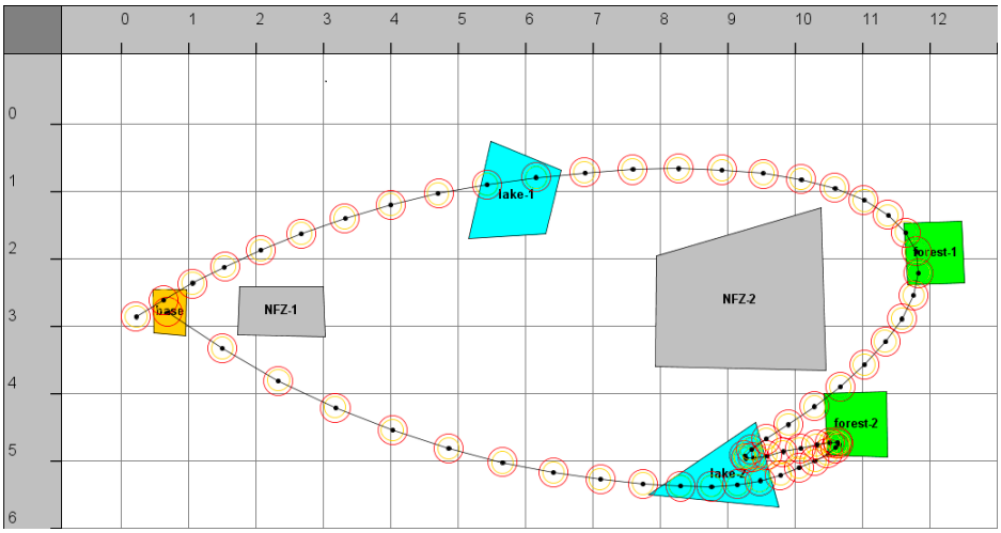
Source: Elaborated by the author.

These routes can be defined by specific planners that depend on variables such as the type of plantation, type of biological input or insecticide to be sprayed, image accuracy, and camera information, among others. Therefore, these routes can be previously defined since they will be inputs to our autonomous system. Thus, we do not employ a path-planning method for such routes, which will be input provided to our system by the users. For this reason, we create specific routes to simulate an action in a region of interest.

On the other hand, the routes between the regions of interest are defined online, where the mission planner must trigger a route planning and re-planning algorithm. These routes are planned in a structured and non-convex environment. This means that the route planner between regions of interest must deal with the no-fly zones *(No Flight Zones - NFZ)* in Figure 10. The system proposed here can incorporate one of the three available algorithms: Hybrid Genetic Algorithm (HGA) - (SOUZA; TOLEDO, 2020), Rapidly-exploring Random Trees (RTT), and Potential Field Planning (PFP) - (SAKAI *et al.*, 2018). Figure 13 illustrates paths between regions of interest.

The UAV is a critical system since it can cause damage to people or property, therefore,

Figure 13 – Possible scenario when scheduling tasks for a farm



Source: Arantes (2017).

we need to provide contingency plans. These plans are measures to take in case any situation deviates from the expected one. There are some basic actions that every aircraft should perform in the event of a failure:

- **Emergency landing:** In some situations, it is impossible to return to the take-off point or flight to a support base. In this case, the system must trigger a route planner that lands the aircraft in a safe region.
- **Vertical Land:** In the case of rotary-wing UAVs, in some extreme cases, it is possible to stop the aircraft and land it exactly where it is, avoiding further damage.

Considering that more than one failure can occur, it is necessary to have a decision system to choose which safety action is most appropriate for the current scenario. According to (RUSSELL; NORVIG, 2016), decision problems can be solved with Decision Networks based on Bayesian networks, Markov Decision Process (MDP) (MDP) and with Partially Observable MDPs (POMDPs). Thus, related approaches were proposed for a decision-making method that is capable of providing the autonomous system the necessary self-awareness.

3.2 Problem Definition

The present section mathematically states the mission and path-planning problems our autonomous system addresses. First, we define next the mission planning problem.

Definition 1. The mission planning problem, illustrated in a farm scenario as previously described, aims to find a solution $\mathscr{S}^* = \langle \mathbf{x}^*, \mathbf{u}^* \rangle$ from $\langle \mathbf{x}, \mathbf{u}, \mathscr{I}, \mathscr{M}, \mathscr{P}, \mathscr{J} \rangle$ such as:

- $\mathbf{x} = [x_0, x_1, ..., x_{T-1}]$: variables of the UAV states through the time steps $t = 0, ..., T-1$.
- $\mathbf{u} = [u_0, u_1, ..., u_{T-1}]$: variables of controls $(\mathbf{u}_t)$ applied in UAV through the time steps.
- $\mathscr{I} = \langle \bar{\mathbf{x}}_0, \Sigma_{x_0} \rangle$ initial conditions with $\mathbf{x}_0 \sim \mathscr{N}(\bar{\mathbf{x}}_0, \Sigma_{x_0})$.
- $\mathscr{M} = \langle A, B, \Sigma_{w_t} \rangle$ stochastic plant model: $\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t + \omega_t$, where ω_t is an additive noise with $\omega_t \sim \mathscr{N}(0, \Sigma_{w_t})$.
- $\mathscr{P}$: *State Plan* (see *Def.* 2).
- $\mathscr{J}$: objective function to be optimized.

Definition 2. The state plan is given by the tuple $\mathscr{P} = \langle \mathscr{E}, \mathscr{C}, \mathscr{A} \rangle$, where:

- $\mathscr{E} = \{e_0, e_1, ...\}$: set of discrete events $e_i \in \mathscr{E}$.
- $\mathscr{C} = \{r_1, r_2, ...\}$: constraints related to the plant state through the time steps. $[\mathscr{C}_r^{lb}, \mathscr{C}_r^{ub}]$, for events.
- $\mathscr{A} = \{a_1, a_2, ...\}$: set of actions with each action $a \in \mathscr{A}$ happening between two events.

Figure 14 illustrates events, constraints and episodes for our planning problem, where we have a directed acyclic graph with events drawn as vertices and episodes as rectangles. The episode is a constraint with $a = \langle e_a^S, e_a^E, \Pi_a, R_a \rangle$, where e_a^S and e_a^E are the initial and final event for the episode a. The set Π_a has the time steps when episode a is activated, and R_a summarizes the set of constraints to be satisfied. The authors in (ONO; WILLIAMS; BLACKMORE, 2013) report three episodes of interest:

1. *Start-in* $(a \in \mathscr{A}^S)$: $x_t \in R_a$ holds in the episode beginning.
2. *End-in* $(a \in \mathscr{A}^E)$: $x_t \in R_a$ holds at the episode ending
3. *Remain-in* $(a \in \mathscr{A}^R)$: $x_t \in R_a$ while the episode is activated.

For example, in figure 14, the event e_2 triggers the action "clean-camera" with $\langle e_2^S, e_2^E, \Pi_2, R_2 \rangle$, where the plan will define e_2^S and e_2^E as the initial and final event for the episode trigged by action "clean-camera" and the set Π_2 of related time steps when "clean-camera" is activated. Also, the mission plan must not violate any constraints R_2 for such an event.

The path-planning problem will handle the obstacle avoidance through the previously defined non-fly zones. The trajectory will take into account the risk of violate a non-fly zone, which will be described using chance-constraints as follows:

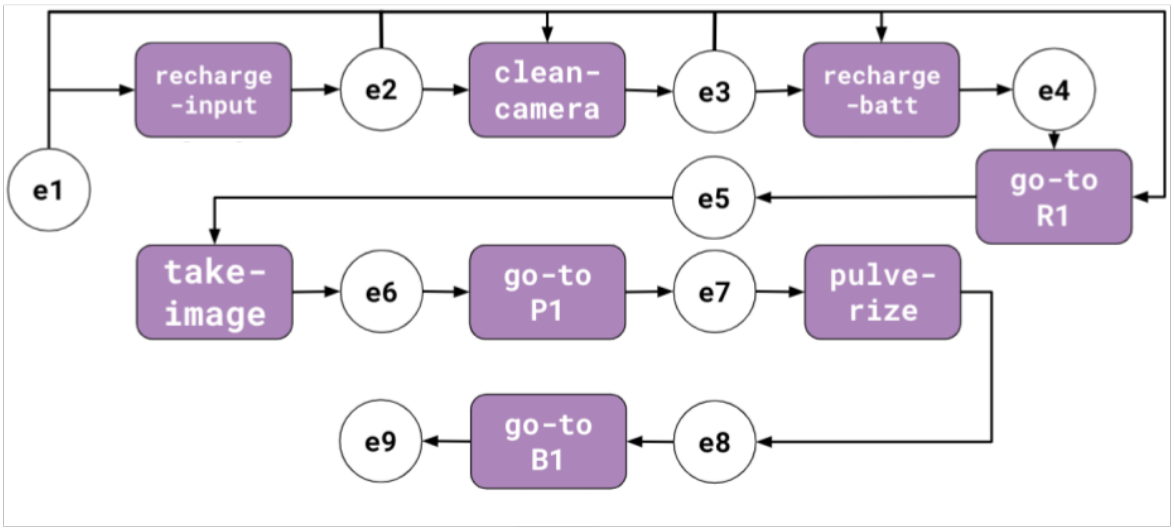


Figure 14 – Example of state plan for the planning problem in a farm scenario.

Definition 3. The non-convex path planning problem with chance-constraints is stated as:

$$Minimize\ \Theta() = \sum_{t} \|u_t\| \tag{3.1}$$

where:

$$x_T = x_{goal} \tag{3.2}$$

$$\mathbf{x}_{t+1} = A\mathbf{x}_t + B\mathbf{u}_t + \omega_t \qquad \forall(t) \tag{3.3}$$

$$\omega_t \sim \mathcal{N}(0, \Sigma_{w_t}) \qquad \forall(t) \tag{3.4}$$

$$x_t \in \mathbb{I}_j \Leftrightarrow \bigwedge_{i \in H_j^{\mathbb{I}}} h_i^T x_t \leq g_i \tag{3.5}$$

$$x_t \in \mathbb{O}_j \Leftrightarrow \bigvee_{i \in H_j^{\mathbb{O}}} h_i^T x_t \geq g_i \tag{3.6}$$

The objective function (3.1) can minimize some metrics related to the control, while constraints (3.2) define the goal as the last state of the UAV when planning a trajectory. Constraints (3.5) and (3.6) describe hyperplane defining convex regions for stay-out (obstacles or non-fly zones) and stay-in (landing spots) regions. Stay-out is a disjunction of linear constraints, while stay-in is established as the conjunction of linear constraints. The plan obtained from the State Plan in *Def.* 2 will demand the resolution of several path planning problems, as stated in *Def.* 3, during its execution.

3.3 Final Remarks

This chapter discussed and stated the problem of autonomous UAV systems through an example based on a farm scenario. We introduced a general description of the problem, where the need for obstacle avoidance, mission, and path re-plans are stated. The relevance of defining regions of interest and no-fly zones for path planning was emphasized. Furthermore, we underscored the significance of path planning in static and non-convex scenarios. We elaborated on the essence of UAV's self-awareness, where the autonomy must handle external and internal events. Since it is a complex problem, we state the requirements assumed in this project to build what is named here as an autonomous system, focusing on the UAV's capabilities in planning, executing, and responding to failures.

Lastly, we delved into the formal problem definition by presenting equations that encapsulate the mission and path planning problems in a farm scenario. The chapter offered a mathematical understanding of the path planning problem, highlighting the significance of constraints, episodes, and regions. As drones become increasingly prevalent in agriculture and other sectors, the challenges associated with autonomous systems become more complex. This chapter aimed to shed light on the challenges addressed by this study, offering insights into robust and efficient autonomous UAV systems proposed by our study.

CHAPTER

4

FOUNDATIONS

This chapter presents the foundations proposed to deal with the problem described in chapter 3. The foundations focus on the following main aspects to develop an autonomous system:

- Autonomy
- Code reusability.
- Programming language independence.
- Modularity with ease of communication between modules.
- Robust task planning.
- Aircraft Safety.

4.1 Autonomy of robotic systems

The definition of autonomy in the Michaelis dictionary is given by:

- Ability to self-govern, to direct itself by its laws or its own will; sovereignty
- Own faculty of some institutions regarding the decision on organization and norms of behavior, without bending or being influenced by external impositions.
- Moral or intellectual freedom of the individual; personal independence; right to make decisions freely.

The authors in (RUSSELL; NORVIG, 2016) state the concept of autonomy for an agent in Artificial Intelligence (AI) as the ability to deal with partial or incorrect prior knowledge

through learning. Thus, when we refer to an autonomous robotic system, it can plan, control, and act with minimal human interaction. Autonomy is often confused with automatic when we restrict autonomy to a single aspect of the system, such as locomotion autonomy. Several definitions of levels of autonomy have been created, and the authors at (BLASCH, 2018) present some types of levels of autonomy:

A) **National Institute of Standards and Technology (NIST)**: separated the levels into two categories. First, the cognitive defines how much the system can understand its environment from its sensors. According to the perceptive –the level grows when the robot has a greater capacity to transform data into information.

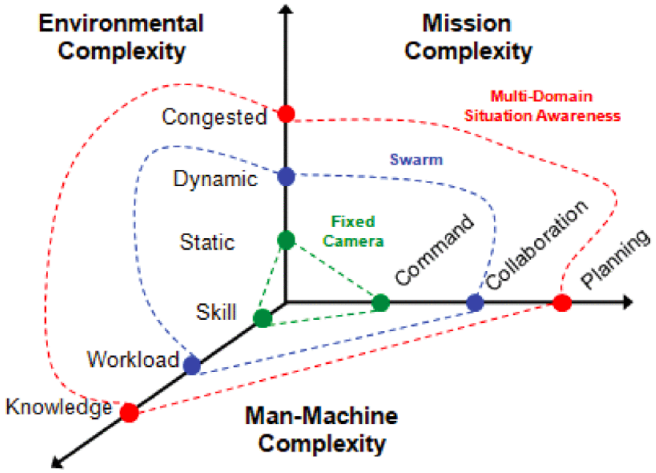
B) **The NATO standards**: performance metrics for contextual autonomy (mission and operational environment) and non-contextual autonomy (external events). System autonomy should address control, communication, mission, and interface.

C) **The Society of Automotive Engineers (SAM)**: evaluates autonomy levels incrementally, considering the lowest autonomy level 0, where full human interaction is required, up to level 5, where no interaction is required.

D) **ALFUS Contextual Autonomy Capability (CAC)**: classifies the autonomy complexity factors in three axes, namely environment factors, mission requirements, and the complexity in human-machine interaction as represented in Figure 15. In this case, four automation characteristics are considered: perception, modeling, planning, execution, and mission.

In this work, we chose to work with the three axes of autonomy represented in Figure 15. Therefore, the methodology adopted here consists of moving forward within a dynamic environment with mission planning and embedded knowledge.

Figure 15 – Autonomy complexity



Source: Blasch (2018).

4.2 Robot Operating System - ROS

We employed the Robot Operating System (ROS) since it meets the aspects of code reusability, programming language independence, and modularity through an easy way to keep robust communication between modules. Given the wide variety of robots, hardware, and sensor functionality types, creating robotics software becomes daunting. ROS provides a modular layer of structured communication, connecting the robot to its peripherals and facilitating the programming of robotic systems (QUIGLEY *et al.*, 2009). ROS is not an operating system per se, but it is structured to provide resources for the deployment of high-level robotic systems, meeting four fundamental implementation concepts:

1) ***Nodes:*** These are executable processes, making ROS modular, where each node represents a software module. The nodes communicate through the exchange of messages.

2) ***Messages:*** Messages are data structures that can be primitive types (integer, boolean, string, etc.), arrays of types, and also composed of other types of message. To convey these messages, *nodes* publish messages in threads.

3) ***Topics:*** These are logical structures, so *nodes* can publish data and/or subscribe to topics for reading data. There are no limits on how many nodes can publish and subscribe to topics, and there are no limits on nodes per topic.

4) ***Services:*** Services are particular types of nodes analogous to web services. Such services receive a request when called and return a response. This type of node was created to work with synchronous actions.

ROS was created in 2007 by a group of researchers at Stanford with its first version was released in 2009[1], and since then, it has been widely used in robotics. Its structure facilitates communication between components of a robot and between different robots, making possible the interaction of different ecosystems for a single purpose.

Many robotics projects use ROS (DENG *et al.*, 2018; SARTORI *et al.*, 2018; SAMPEDRO *et al.*, 2018; MOHAIMENIANPOUR; VAUGHAN, 2018; VANEGAS *et al.*, 2016; WOPEREIS *et al.*, 2015; JIANG; ELBAUM; DETWEILER, 2013; SAPKOTA *et al.*, 2016; CHUNG *et al.*, 2016) as previously mentioned.

[1] information taken from: https://www.ros.org/ accessed on August 15, 2023

4.3 Problem Definition Domain Language - PDDL

Problem Definition Domain Language (PDDL) is a descriptive language created and updated in International Planning Competitions (IPC) since 1998 (FOX; LONG, 2003). PDDL aims to allow robust planning of tasks by stating a syntax that facilitates this. Its syntax was inspired by the Lisp language and had two main execution codes: domain and problem. The domain code describes the environment and the rules to be followed, and the problem code presents the initial state, the desired final state, and the search metric.

PDDL is processed through solvers, and no exclusive compiler exists to find an action plan. IPC was organized in Delft, Netherlands, in 2018 within the International Conference on Planning and Programming (ICAPS)[2]. The objective of the competition was to promote and highlight planning challenges, introducing new problems as a reference for future research. Researchers created different solvers for a set of problems made available more efficiently, according to the parameters defined by the competition.

4.3.1 Domain code

The code 1 is an example of the domain structures: **define**, **requirements**, **types**, **predicates**, and **functions**. The structure **define** determines the document type (**domain** or **problem**) and its name. The **requirements** are the characteristics used in the domain that are checked by the solver at the beginning to confirm if it is possible to solve the problem. The **types** section allows object typing, and the language allows object hierarchy.

In **predicates**, all types of Boolean predicates are declared, defined by a name, being a tuple of variables with their respective types. Functions are declared in **functions** similarly to predicates, but while predicates represent a boolean value, functions have numerical values. Actions are the transition between states and within the actions, **parameters** represent the values to be manipulated. The **precondition** specifies all necessary conditions for an action being performed, which means the state the system must be in to allow an action. The state after the action is described in **effect**.

Source code 1 – Domain example - incomplete code *turtlebot* in PDDL

```
1: (define (domain turtlebot_example)
2: (:requirements :strips :typing :disjunctive-preconditions)
3: (:types
4:    robot
5:    waypoint
6:    floor
7: )
8: (:predicates
```

[2] https://ipc2018.bitbucket.io/ accessed January 28, 2020

```
 9:
10:   ;;robot predicates
11:
12:   (person_greeted ?v - robot)
13:   (person_guided ?v - robot)
14:   (robot_at ?v - robot ?wp - waypoint)
15:   (undocked ?v - robot)
16:   (docked ?v - robot)
17:   ;; [...]
18:
19:   ;;waypoint & floor predicates
20:
21:   (dock_at ?wp - waypoint)
22:   (waypoint_at_floor ?wp - waypoint ?fl - floor )
23:   (elevator_access_waypoint ?wp - waypoint)
24:   (visited ?wp - waypoint)
25:   ;; [...]
26: )
27:
28: ;; Move to any waypoint, avoiding terrain
29: (:action goto_waypoint
30:   :parameters (?v - robot ?from ?to - waypoint ?fl - floor)
31:   :precondition (and
32:     (undocked ?v)
33:     (outside_elevator ?v)
34:     (robot_at ?v ?from)
35:     (waypoint_at_floor ?from ?fl)
36:     (waypoint_at_floor ?to ?fl)
37:     (hallway_waypoint ?from)
38:     (hallway_waypoint ?to)
39:     (allowed_goto_waypoint ?v)
40:     )
41:   :effect (and
42:     (robot_at ?v ?to)
43:     (not (robot_at ?v ?from))
44:     (not (allowed_goto_waypoint ?v))
45:     )
46: )
47: (:action guide_person
48:   :parameters (?v - robot ?wp - waypoint)
49:   :precondition
50:     (and
```

```
51:     (undocked ?v)
52:     (outside_elevator ?v)
53:     (robot_at ?v ?wp)
54:     (person_greeted ?v)
55:     (destination_waypoint ?wp)
56:     )
57:   :effect (and
58:     (person_guided ?v)
59:     (allowed_goto_waypoint ?v)
60:     )
61: ))
```

In code 1, the *:requirements* define:

1) **:strips -** Allows the use of basic add and delete effects.

2) **:typing -** Allows the use of types for objects. Typing is similar to classes and sub-classes in Object Oriented Programming.

4) **:disjunctive-preconditions-** Allows use of **or** in **goals** and **preconditions**.

4.3.2 Problem Code

The code 1 shows an example of a problem in PDDL, available in the ROSPlan examples repository[3].

In the problem, after defining it as a problem and its name in **define**, it must be related to its **domain**. In **objects**, we instantiate objects within their respective **types**. The system's initial state is declared in **init**, using predicates and functions specified in the domain. In the same way, **goal** declares the state you want to reach.

Source code 2 – Problem example *turtlebot* in PDDL

```
1: (define (problem turtlebot_example_task)
2: (:domain turtlebot_example)
3: (:objects
4:   kinba - robot
5:   wp0-0 wp0-1 wp0-2 wp0-3 wp0-4 wp0-5 wp0-6 wp1-6 wp1-7 wp1-8 - waypoint
6:     fl0 fl1 - floor
7: )
8: (:init
9:   (robot_at kinba wp0-0)
10:  (docked kinba)
```

[3] https://github.com/KCL-Planning/rosplan_demos accessed on August 15, 2023

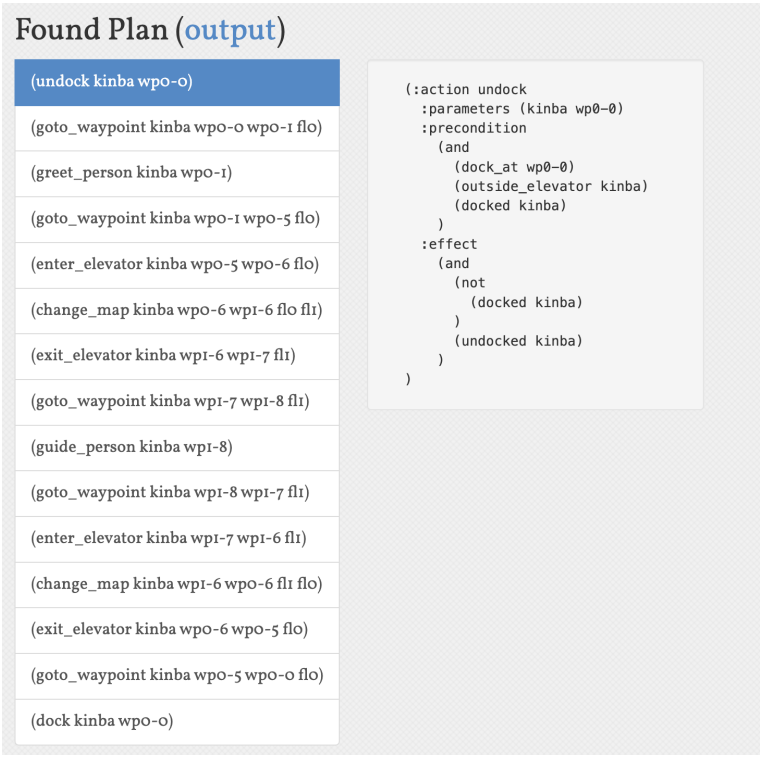
```
11:    (outside_elevator kinba)
12:    (allowed_goto_waypoint kinba)
13:
14:    (dock_at wp0-0)
15:
16:    (elevator_waypoint wp0-6)
17:    (elevator_waypoint wp1-6)
18:    (current_elevator_waypoint wp0-6)
19:
20:    (elevator_access_waypoint wp0-5)
21:    (elevator_access_waypoint wp1-7)
22:
23:
24:    (greeting_waypoint wp0-1)
25:    (destination_waypoint wp1-8)
26:
27:    (hallway_waypoint wp0-0)
28:    (hallway_waypoint wp0-1)
29:    (hallway_waypoint wp0-2)
30:    (hallway_waypoint wp0-3)
31:    (hallway_waypoint wp0-4)
32:    (hallway_waypoint wp0-5)
33:    (hallway_waypoint wp1-7)
34:    (hallway_waypoint wp1-8)
35:
36:    (waypoint_at_floor wp0-0 fl0)
37:    (waypoint_at_floor wp0-1 fl0)
38:    (waypoint_at_floor wp0-2 fl0)
39:    (waypoint_at_floor wp0-3 fl0)
40:    (waypoint_at_floor wp0-4 fl0)
41:    (waypoint_at_floor wp0-5 fl0)
42:    (waypoint_at_floor wp0-6 fl0)
43:    (waypoint_at_floor wp1-6 fl1)
44:    (waypoint_at_floor wp1-7 fl1)
45:    (waypoint_at_floor wp1-8 fl1)
46:
47: )
48: (:goal (and
49:    (person_guided kinba)
50:    (person_greeted kinba)
51:    (docked kinba)
52:
```

```
53: ) ) )
```

The problem starts with the robot (defined as *kinba*) at waypoint 0 (*(robot_at)*), parked (*docked*) at waypoint 0 (*dock_at*). As the predicates are unrelated, it is necessary to define that the robot is at a point and is simultaneously parked at a point. This is due to the language structure that allows to relax rules to facilitate finding a plan. If the action *undock* did not have the effect *(not (docked ?v))* the robot would be both parked and moving.

The problem seeks as a final state having a person guided, person greeted, and parked (*docked*), where all objectives must be achieved. In Figure 16, the planner's response is presented through text and visual form.

Figure 16 – Problem solution for 2 in 1



Source: Elaborated by the author.

The PDDL language is used in (KUHNER *et al.*, 2018) with the authors describing an adaptive interface with goal setting, where the user establishes the general objective with high-level commands. In (STEENSTRA, 2019), a PDDL-based system is developed for task planning of an autonomous underwater vehicle, considering positioning uncertainty and limited communication with the agent.

4.4 ROSPlan

ROSPlan is an architecture that incorporates job planning into the ROS system (CASH-MORE *et al.*, 2015). This tool has two main components shown in Figure 17, **Knowledge Base**

and **Planning System**.

- **Knowledge Base:** Collection of interfaces intended to store the current environment variables as a PDDL model. It automatically generates problem files in PDDL and translates actions from PDDL to actions in ROS system, notifying the planner if the environment changes.
- **Planning System:** It is responsible for planning management. This is done by generating the problem according to the available information from the Knowledge Base, transferring the problem instance to the planner, performing the *parsing* of the generated plan, and sending it to the platform as actions.

The main processes of the Planning System are shown in Figure 17 and described below:

- **Problem Generator:** Responsible for automatically generating a problem in PDDL, publishing it in a ROS topic, or saving it in a file.
- **Planner:** Communication interface with the solver that sends the specified problem and publishes the result of the planner (action plan) in a topic or file.
- **Plan Parser:** Module that converts the PDDL file plan into ROS messages for execution.
- **Plan Dispatch:** It encapsulates the execution of the plan in process, executes the service related to the plan, and returns an error message if it occurs.

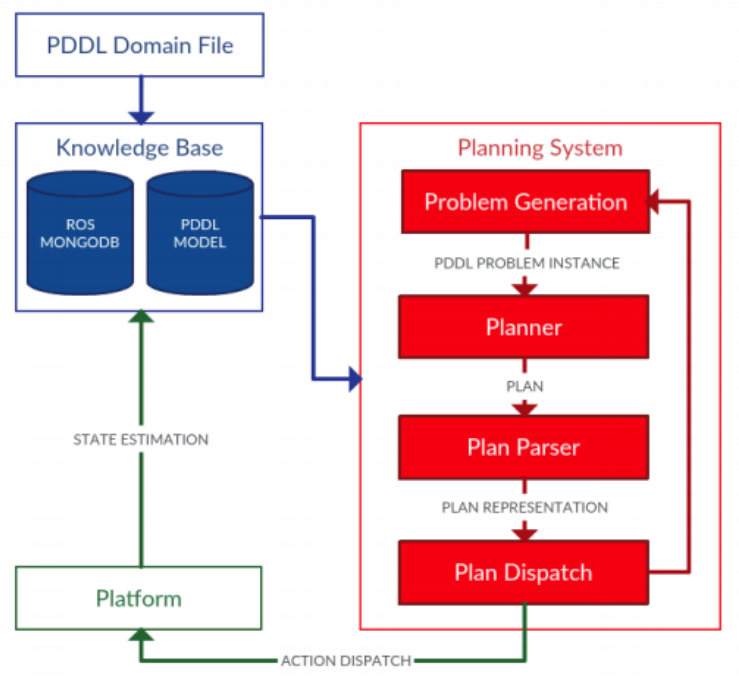
ROSPlan does not have a specific *solver* for PDDL; the system is modular and allows the user to choose the solver that best suits the problem.

4.5 Final Remarks

In this chapter, we proposed the necessary basic concepts to address the challenges outlined in Chapter 3. Key concepts for developing an autonomous system include autonomy, code reusability, programming language independence, modularity with seamless communication between modules, robust task planning, and aircraft safety.

The concept of autonomy, as discussed in the Michaelis dictionary and literature (RUSSELL; NORVIG, 2016; BLASCH, 2018), was explored by highlighting its various dimensions and levels. We adopted a three-axis approach to autonomy, considering environmental factors, mission requirements, and complexity in human-machine interaction. This approach forms the foundation for our methodology's dynamic environment, mission planning, and embedded knowledge.

Figure 17 – ROSPlan Architecture



Source: Sanelli *et al.* (2017).

To implement our methodology, we used the Robot Operating System (ROS), a powerful framework that aligns with our objectives of code reusability, programming language independence, and modularity. ROS's architecture, including nodes, messages, topics, and services, facilitates seamless communication between different components of a robotic system and across diverse robots.

Furthermore, we introduced the Problem Definition Domain Language (PDDL) as a descriptive language to model planning problems. PDDL provides a structured syntax for defining the environment, rules, initial and final states, and search metrics. It has been widely used in the planning community and complements our methodology by enabling robust task planning.

The ROSPlan architecture was presented as a pivotal component of our methodology, integrating job planning into the ROS ecosystem. The Knowledge Base and Planning System components work synergistically to manage knowledge representation, problem generation, planning, and plan execution. ROSPlan's flexibility allows users to choose the most suitable PDDL solver for their problem.

In summary, our methodology combines the richness of autonomy, the flexibility of ROS, and the expressiveness of PDDL to create a holistic approach to developing autonomous systems capable of robust task planning and execution. The following chapters will delve into the practical implementation and evaluation of this methodology in the context of specific robotic applications.

CHAPTER

5

METHODOLOGY

This chapter presents the methodology proposed to deal with the problem described in chapter 3 using the materials presented in chapter 4.

In this chapter, we introduce Harpia, an autonomous system designed for UAV operations. Harpia operates seamlessly from a ground station or as an embedded system within the UAV platform, excelling in decision-making and independent execution of missions. Its capabilities encompass advanced features such as meticulous mission planning, adaptive diagnostics, and real-time risk mitigation. Developed on the Robot Operating System (ROS), Harpia integrates Planning Domain Definition Language (PDDL) for task planning, K-Nearest Neighbors (KNN) for dynamic path planning, Bayesian Networks for risk assessment, and fault detection using Principal Component Analysis (PCA) and Decision Tree classification. This chapter comprehensively overviews Harpia's architecture, functionalities, and role in advancing UAV operations' autonomy.

5.1 Harpia

This section describes the proposed autonomous system, Harpia, which can run from a ground station or be embedded in the UAV platform. The UAV with Harpia embedded can operate without a communication link with the ground station. In this case, our system can replace the pilot for designed decision-making and assume some behaviors for critical situations, e.g., performing an emergency landing or returning to base.

The current decision-making features of Harpia improve flight safety, facilitate handling UAV operations, and increase the efficiency of carrying out missions. Mission safety improves since Harpia defines the mission plan and the related trajectories, taking into account obstacle avoidance as an external factor and battery health as an internal one. Besides the risk mitigation component, the system continuously evaluates flight health to detect internal problems.

The operation of the UAV under Harpia becomes easy once the pilot only needs to set the necessary inputs, launch Harpia system with some commands, and start the UAV. Next, Harpia autonomously carries out the mission plan and its execution. Finally, the efficiency of carrying out missions arises from the system's ability to generate optimized plans using PDDL to describe the scenario of a current mission.

5.1.1 Harpia overview

In the current commercial solutions, the pilot is usually in charge of the UAV flight. For example, the scenarios described in Figure 10 would generate one mission for each region, with the user defining the missions' execution sequence and the pilot controlling each flight. There is a need for commercial systems that execute repetitive missions without the pilot guidance, as reported in (COHN *et al.*, 2017).

There are tailor-made commercial solutions for in-flight diagnostics of UAV systems. Still, the authors in (NOUACER *et al.*, 2020) report the demand for autonomous and adaptive diagnostics systems. Such systems can improve robustness, reduce costs, and increase autonomy. Our system advances by adding more autonomy to plan and re-plan missions and trajectories and monitoring UAV safety.

Harpia system is developed on ROS following figure 18 architecture. The requirement to embed Harpia onboard is a companion computer, such as a Raspberry Pi model B or a similar one, with at least 4GB SDRAM. In the presence of a reliable link between the ground station and the UAV, we do not need to embed the proposed system. In this case, Harpia can run using a regular laptop at the ground station. On the other hand, if the communication link is unreliable, Harpia can be embedded in single-board computers, as mentioned. However, we must highlight that the system is currently running from a ground station, and its embedded version will be validate as a future work

The systems in ROS nodes are:

- **ROSPlan:** We add here the Kings College's system ROSPlan , describe in (SANELLI *et al.*, 2017), to use the available PDDL structure and solvers.

- **Decision Support & Making:** Manage the mission plan and execution.

  - **Mission Planning:** Calls ROSPlan features for mission planning.
  - **Risk Mitigation:** A Bayesian Network (BN) was developed to decide about keeping the plan execution, re-planning actions, or abort the current plan.
  - **Mission Goal Manager:** Updates goals since users may add or remove them.

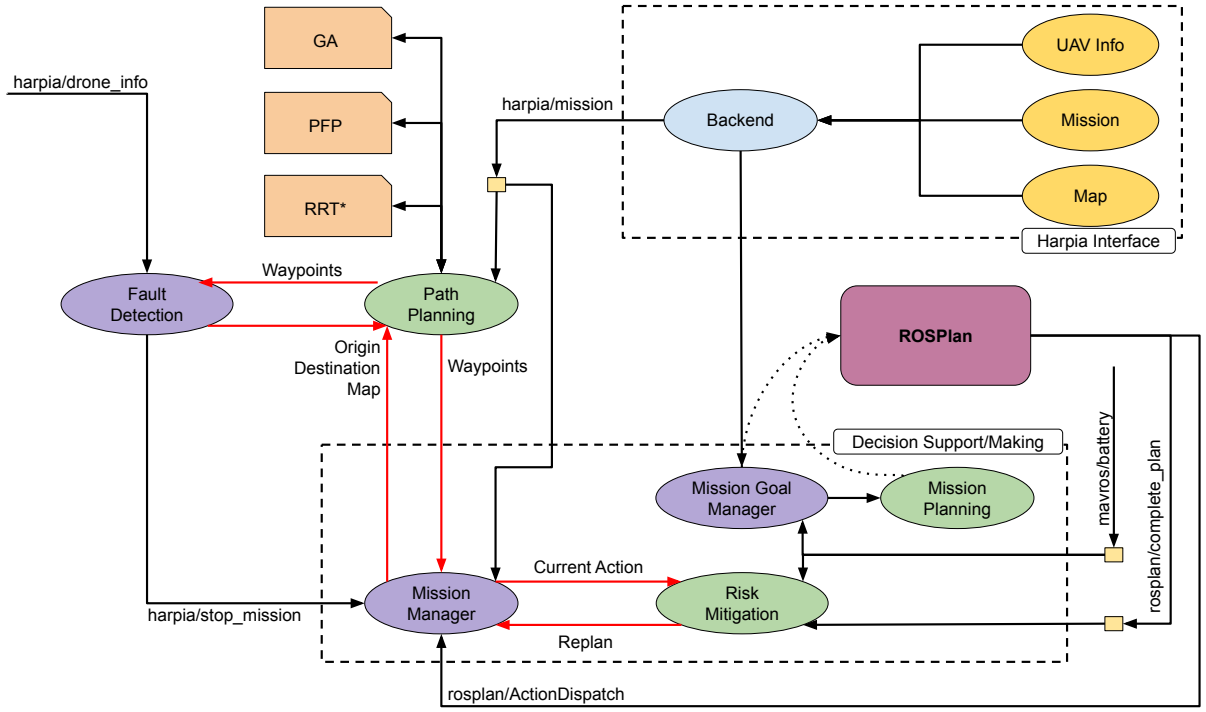


Figure 18 – Description of the ROS nodes and communication

- **Path Planners:** This module interfaces with different path planners. The K-nearest neighbor (KNN) algorithm was employed as a machine learning approach to select a planner for each scenario.

- **Fault Detection**: Supervises flight safety using combined PAC-Tree algorithms to identify anomalous flight patterns.

Algorithms 3-5 summarize the iterations among ROS nodes in Harpia responsible for the Mission. The **Mission Goal Manager** (Algorithm 3) feeds the knowledge base to create the PDDL domain by sending the input data: UAV Info, Map, and Goals. UAV Info has the hardware attributes of the aircraft,e.g., avionics system and sensor features, which depend on the UAV platform employed and the mission executed. Map describes the current environment (map area, no-fly zones, support bases), and Goals have the regions of interest to be reached and the actions to be executed. We integrate all these Harpia Interface inputs using ROS. **Mission Goal Manager** will update inputs and the ROSPlan knowledge base, while **Mission Planning** returns true. In Algorithm 4, the **Mission Planning** node calls **ROSPlan** to generate the problem and a plan to solve it.

If the problem or plan generations do not succeed, we have a problem, and the **Mission Goal Manager** will receive False reporting to the system. For instance, the users can receive a message telling them that it is not possible to execute the mission with the current data. If everything works, **Mission Planning** will execute **ROSPlan** to dispatch an action to **Mission Manager**. In Algorithm 5, the **Risk Mitigation** system is called to verify if the action can be

executed.

At this point, BN will evaluate the feasibility of the current sequence of actions. If there is a problem, the system will try to re-plan the mission by calling **ROSPlan** again with the current UAV state. If it is not possible to generate a new plan, **Mission Planning** will receive false and report that plan generation does not succeed to **Mission Goal Manager**. Otherwise, we have re-planned the mission, and **Mission Manager** will execute the plan actions. Next, we describe how PDDL, BN, path planning, and fault detecion algorithms work in Harpia.

Source code 3 – Mission Goal Manager

```
1: ROSPLAN.knowledge base(UAV, Map, Goals)
2: while(Mission_Planning()):
3:     UAV,MAP,Goals = updateInput()
4:     ROSPLAN.knowledge_base(UAV, Map, Goals)
```

Source code 4 – Mission Planning

```
 1: stop = False
 2: if(problem == ROSPLAN.generateProblem()):
 3:     if(plan == ROSPLAN.generatePlan(problem):
 4:         while(not stop):
 5:             action = ROSPLAN.actionDispatch(plan)
 6:             if(action is Null):
 7:                 stop = True
 8:             else
 9:                 stop = Mission_Manager(action.plan)
10: return stop
```

Source code 5 – Mission Manager

```
1: fault = Mission_Fault_Detection(action.plan)
2: if (fault):
3:     replan = ROSPLAN.generatePlan(problem, action, plan)
4:     if(not replan):
5:         return False
6: Do_Action(action.plan)
7: return False
```

5.1.2 PDDL

The task planning problem is solved using the PDDL2.1 language, which is expressive enough for domain and problem definition (FOX; LONG, 2003). Figure 19a gives an overview of the domain, where we assume high-level planning to schedule actions. For instance, the UAV must load the right input for spraying. The battery must be enough to execute the trajectory, and the on-board camera must usually be clean after a spraying action.

PDDL2.1 allows us to properly define the domain and actions for a constrained plan, where the UAV can verify preconditions before performing actions. This will prevent the aircraft from flying without enough battery, leading it to recharge it in a support base (`go_to_base`). UAV autonomously goes to the base and loads the battery as often as necessary to carry out a mission.

Figure 19b illustrates the model for a problem. Our system establishes the communication between the user and ROSPlan. As mentioned, the user only needs to provide information about the map, mission goals, and UAV hardware. The system updates the knowledge base for each mission request, calls for a plan, and the planned dispatch. In this example, the mission objectives are to capture the images of regions three to six. We only add the distances between the goals' regions and the bases to avoid unnecessary complexity while searching for a planned schedule.

5.1.3 PATH PLANNING

The Path Planning system will receive origin, destination, and obstacle positions to calculate a feasible route. It is called by $Do\_Action()$ in Algorithm 5 with trajectories being requested through *go_to* actions. The trajectory can be generated by one of the three available algorithms: Hybrid Genetic Algorithm (HGA) - (SOUZA; TOLEDO, 2020), Rapidly-exploring Random Trees (RRT) and Potential Field Planning (PFP) - (SAKAI *et al.*, 2018).

We made changes in RRT and PFP algorithms to address obstacle avoidance. RRT applies Ray Casting (RC) algorithm to check collisions following the same approach introduced in (SOUZA; TOLEDO, 2020) for HGA. The Ray Casting (RC) algorithm is used in computer graphic applications since it traces rays from a source and finds the nearest point blocking the beam. Polygons represent non-fly zones or obstacles in our scenarios, and RRT will avoid such polygons by using RC to validate the waypoints sampled when building branches. In this context, RC traces horizontal rays from each waypoint and calculates the number of intersections with the polygon.

In RRT, we first define the Ray Cast Point (RC_P) to verify a waypoint inside the obstacle. If the ray intercepted the polygon an odd number of times, the waypoint is inside the area; if it blocked an even number, the waypoint is outside the area, as summarized by expression (5.1). Next, Ray Cast Segment (RC_S) checks if there are intersections between the segment of two consecutive waypoints and the polygons, as stated by expression (5.2):

```
(define (domain harpia)

    (:requirements  :typing  :strips  :disjunctive-preconditions  :equality )
    (:types
        region - object
        base - region)
    (:functions

        (battery-amount)
        (input-amount)
        (recharge-rate-battery)
        (discharge-rate-battery)
        (battery-capacity)
        (input-capacity)
        (recharge-rate-input)
        (distance ?from - region ?to - region)
        (velocity)
        (picture-path-len ?region - region)
        (pulverize-path-len ?region - region)
        (total-goals)
        (goals-achived)
        (mission-length)
    )
     (:predicates

        (been-at ?region - region)
        (carry)
        (at ?region - region)
        (can-spray)
        (can-recharge)
        (taken-image ?region - region)
        (pulverized ?region - region)
        (can-take-pic)
        (its-not-base ?region - region)
        (pulverize-goal ?region - region)
        (picture-goal ?region - region)
        (hw-ready ?from - region ?to - region)
    )


    (:action go_to
        :parameters (
            ?from-region - region
            ?to-region - region)
        :precondition (and
            (at ?from-region)
            (> (battery-amount) (+ (* (/ (distance ?from-region ?to-region) (velocity))
                                      (discharge-rate-battery)) 15))
        )
        :effect (and
                (not (at ?from-region))
                (been-at ?to-region)
                (at ?to-region)
                (decrease (battery-amount )
                    (*
                        (/
                            (distance ?from-region ?to-region)
                            (velocity)
                        )
                        (discharge-rate-battery)
                    )

                )
                (increase (mission-length) (distance ?from-region ?to-region))
                )
    )
    (...)
```

(a) PDDL Domain overview

```
(define (problem task)
(:domain harpia)
(:objects
    region_1 region_2 region_3 region_4 region_5 - region
    region_6 region_7 region_8 region_9 region_10 region_11 region_12 - region
    base_1 base_2 base_3 base_4 - base
)
(:init

    (at base_1)

    (picture-goal region_1)
    (picture-goal region_2)
    (picture-goal region_3)
    (picture-goal region_4)
    (picture-goal region_5)
    (picture-goal region_6)


    (= (battery-amount) 100)
    (= (input-amount) 0)
    (= (discharge-rate-battery) 0.042)
    (= (battery-capacity) 100)
    (= (input-capacity) 3)


    (= (distance region_1 region_2) 2659.31)
    (= (distance region_1 region_3) 2474.48)
    (= (distance region_1 region_4) 2981.27)
    (= (distance region_1 region_5) 3855.53)
    (= (distance region_1 region_6) 3065.19)
    (= (distance region_1 base_1) 2137.48)
    (= (distance region_1 base_2) 1421.39)
    (= (distance region_1 base_3) 4930.21)
    (= (distance region_1 base_4) 3092.56)
    (= (distance region_2 region_1) 2659.31)
    (= (distance region_2 region_3) 1352.25)
    (= (distance region_2 region_4) 2627.34)
    (= (distance region_2 region_5) 1231.98)
    (= (distance region_2 region_6) 498.419)
    (= (distance region_2 base_1) 1835.59)
    (= (distance region_2 base_2) 1870.26)
    (= (distance region_2 base_3) 2292.66)
    (= (distance region_2 base_4) 1043.42)
    (= (distance region_3 region_1) 2474.48)
    (= (distance region_3 region_2) 1352.25)
    (= (distance region_3 region_4) 1275.89)
    (= (distance region_3 region_5) 1922.1)
    (= (distance region_3 region_6) 1307.94)
    (...)

    (= (velocity) 3.5)



    (= (mission-length) 0)

)
(:goal (and
    (taken-image region_1)
    (taken-image region_2)
    (taken-image region_3)
    (taken-image region_4)
    (taken-image region_5)
    (taken-image region_6)
    (at base_3)
))
(:metric minimize (mission-length))
)
```

(b) PDDL Problem overview

Figure 19 – Incomplete PDDL domain and problem for Harpia

$$RC_P(x_s, \mathbb{O}_j) = \begin{cases} 1 & \text{, if } x_s \in \mathbb{O}_j \\ 0 & \text{, otherwise} \end{cases} \tag{5.1}$$

$$RC_S(x_s, x_{s+1}, \mathbb{O}) = \sum_{j=0}^{|\mathbb{O}|} |\overline{x_s, x_{s+1}} \bigcap \mathbb{O}_j| \tag{5.2}$$

where x_s and x_{s+1} are two consecutive waypoints defining the segment $\overline{x_s, x_{s+1}}$, and $\mathbb{O}$ is the set of obstacle with $\mathbb{O}_j \in \mathbb{O}$.

Equations 5.4 and 5.6 calculate the repulsive field in PFP algorithm, whose novelty is the inclusion of chance-constrains.

$$PF(x_t) = RP(x_t) + AP(x_t) \tag{5.3}$$

$$AP(x_t) = \frac{d(x_t, D)}{d(O, D)} \tag{5.4}$$

$$RP(x_t) = \sum_{j=1}^{|\mathbb{O}|} (Pr(x_t \in Z_{\mathbb{O}_j})) \tag{5.5}$$

$$Pr(x_t \in Z_{\mathbb{O}_j}) = 1 - F(x_t) \tag{5.6}$$

Equation 5.3 adds up attractive and repulsive potentials in Equations 5.4 and 5.6, respectively. Euclidean distance $d(A,B)$ measures the UAV's position (x_t) from the destination D, where O is the origin. The repulsive field applies chance constraints as reported in (ARANTES *et al.*, 2019); thus, the probability of collision with an obstacle $Z^i_{\Phi^n}$ is given by Pr, and Φ is the set of obstacles. Equation 5.3 indicates how to find Pr from the cumulative distribution function $F(x_t)$ as illustrated by Figure 20 for a normal distribution.

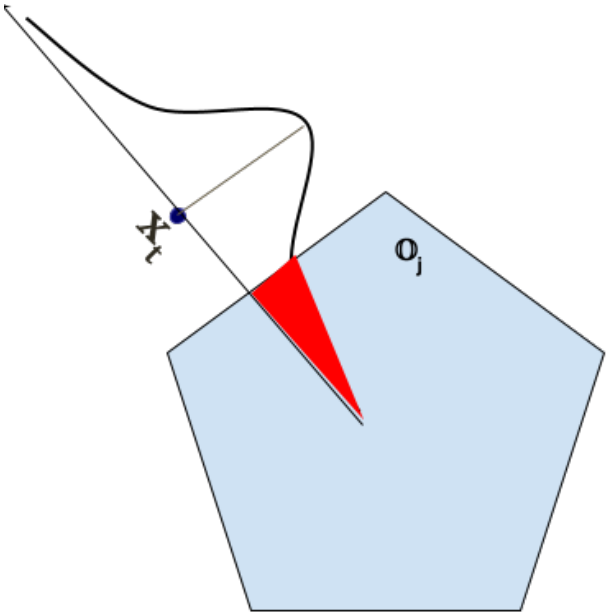


Figure 20 – Risk incurred by the uncertainty related to the state x_t.

Each algorithm produces feasible routes at different run times and generates a path with distinct advantages. Moreover, the path planning system chooses the path planner based on each specific situation, where the variables analyzed to make a choice are battery health, obstacle amount between the regions and the distance (in a straight line) between the origin and destination points.

K-Nearest Neighbors (KNN) technique selects the path planners based on the data about the path to be planned. KNN is a supervised machine learning algorithm for classification

problems, which classifies a data point based on its proximity of neighbors belonging to some classes. The algorithm calculates the distance between data points and selects the 'k' closest points, where 'k' is a pre-defined number, to determine new data classification. The classification can be done, e.g., by voting or weights related to the distance from the K's nearest neighbors.

Our path-planning system employs the KNN classifier to decide which path-planning algorithm to use based on specific variables. The classifier considers the variables battery health (b), time to run the algorithm (t), ratio (r, see Equation 5.9), route length (l), the quantity of waypoints (q). We used the elbow method to determine the optimal value of 'k', which involves plotting the variation in the percentage of variance explained concerning the number of clusters. As 'k' increases, the variance explained increases until adding more clusters doesn't fit the data better. This inflection point, resembling an "elbow" in the plot, providing an optimal value for 'k', which returned $K = 5$ in our study as shown by Figure 21. In this way, by training the KNN classifier on trajectories generated by each path planner for various origins and destinations, we can discern patterns and tendencies in the choice of algorithms for different scenarios.

$$fitness = a + t + r + l + q \tag{5.7}$$

$$a = b * r \tag{5.8}$$

$$r = \frac{\sum_{i=1}^{n} d(x_i, x_{i+1})}{d(O,D)} \tag{5.9}$$

$$\hat{\varsigma} = \frac{\sigma - min(\sigma)}{max(\sigma) - min(\sigma)}, \forall \sigma \in \{a,t,r,l,q\} \tag{5.10}$$

Figure 22 shows the tendencies in the choice of each algorithm. For example, HGA is usually chosen when there are fewer obstacles, PFP is chosen for medium distances, and RRT is for longer distances.

5.1.4 Risk Mitigation

In this section, we describe using the Bayesian Network to mitigate the risk of the mission plan failing due to a lack of battery. The battery consumption becomes high for some reasons, such as an increase in the distance to flight when new areas of interest or obstacles are added, greater resistance in the air caused by wind, and rate consumption higher than estimated due to battery malfunction.

The Bayesian Network (BN) is a semantic representation of probabilistic models, structured as a directed graph (RUSSELL; NORVIG, 2016). The nodes and edges represent variables and conditional dependency between nodes, respectively. Bayesian networks can handle anomaly detection, classification, and clustering, among other tasks in machine learning.

Harpia calls a BN within the Risk Mitigation system every time ROSPlan activates an action. The idea is to verify the plan's feasibility based on the battery level. Figure 23 shows the

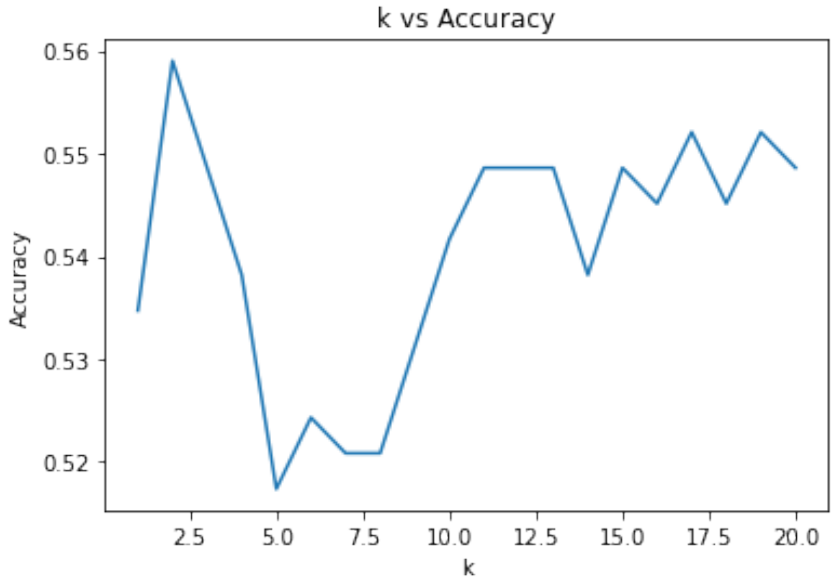


Figure 21 – Elbow method to evaluate K

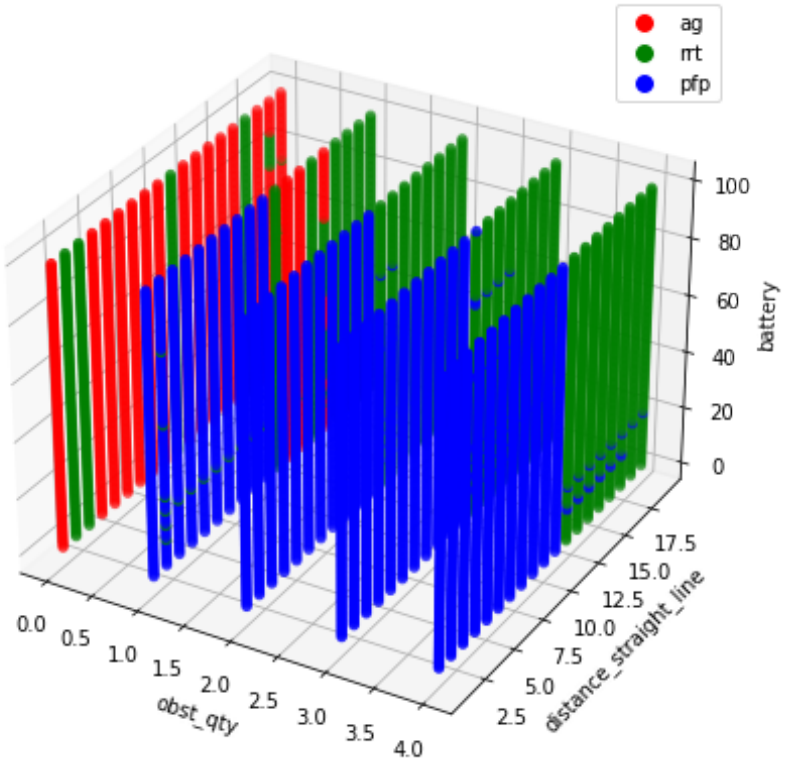


Figure 22 – KNN and path planner selection based on some features.

incremental build process of the BN, which is related to the next action to be performed, e.g., BN is initiated before the drone takes a picture in region 2.

Table 2 has the probability distribution employed, based on the Li-ion discharge voltage curve, where Y is the percentage of available battery to execute the plan's selected action, and P(Y) is the chance of executing such action without a re-planning. For example, we can have

Figure 23 – BN result example where replan is needed. The images show the addition of new nodes on the BN, setting the actions that happened and calculating the probability of re-plan for added nodes.

$Y = 85\%$, meaning a battery with 85% of the full capacity, where P(Y)=0.95 is the chance of executing the selected action following the current plan.

Thus, if the probability of executing an action is greater or equal to keeping the plan, BN evaluates the plan. BN adds a new child node as the next action and adjusts the previous action as evidence. The process will continue for the whole plan or if a node evaluation demands a re-plan. Figure 23 shows when the probability of success for an action is smaller than the drone's probability of not accomplishing an action in the future. At this point, the system stops the BN process and triggers a re-plan.

In Figure 24, we have a scenario where the system decides to re-plan in two different points after it calls the BN. In Figure 24a, the black arrows represent the original plan, but the BN

Table 2 – Probability Distribution

Battery	Probability	Battery	Probability
0<Y<15	0.20	15<=Y<30	0.50
30<=Y<60	0.75	60<=Y<80	0.85
-	-	80<=Y<=100	0.95

identifies the need to recharge before completing such a plan. This happens since the percentage of battery consumption becomes greater than expected, which means a low available battery as the UAV reaches region R2, leading to the re-planning of the current sequence of actions.

The new plan is shown in blue, where the UAV will go to the base after the action in R4 to recharge the battery, mitigating a future hazard. We assume that the battery is not working as expected, which means the discharge rate is not following the desired behavior, e.g., due to wind resistance, an increase in the previous path to avoid obstacles, or even a battery failure. Thus, when the aircraft arrives at R3 in Figure 24b, ROSPlan dispatches another action, and the BN identifies that it is impossible even to reach R4. Next, a second re-plan is called, with the system deciding on another battery recharge. If a re-planning operation is initiated and the aircraft's battery level is at or below 20%, the system will direct the aircraft to the nearest base for recharging before proceeding with the actual re-planning process.

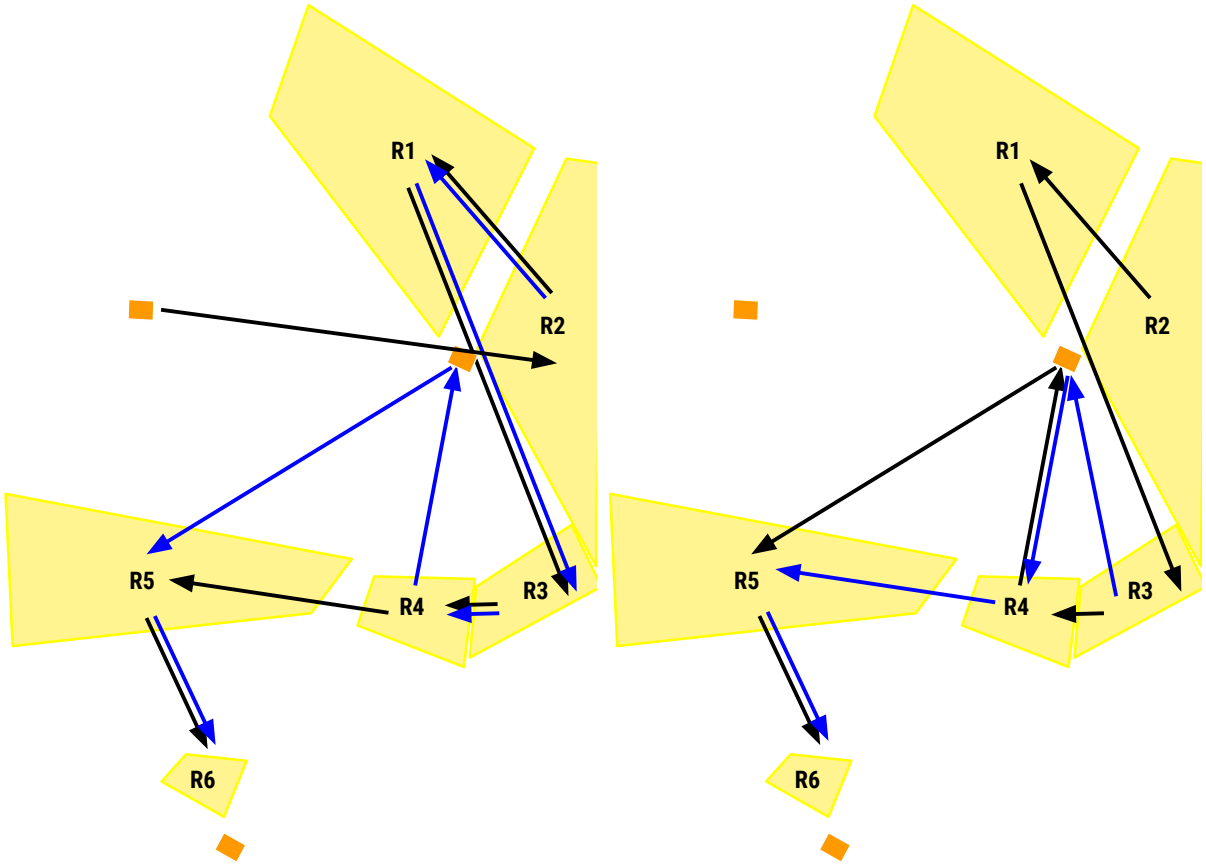
5.1.5 Fault Detection

The erratic flight behavior arises when anomalous values are constantly detected from flight data like roll, pitch, yaw, heading, roll rate, pitch rate, yaw rate, climb rate, and throttle percentage. Those values change for different reasons, such as wind gusts or internal malfunctioning in UAV engines.

PCA and Decision Tree

In our study of fault detection during flights, we simulated 540 flights and introduced noise or errors for 135 of them. Table 3 shows the flight scenarios considered for simulation. An error generator was employed to simulate real-world inaccuracies in measurements and controls. This generator introduces Gaussian noise, modulating the disturbances based on the nature of each parameter and the intended type of error. The Gaussian noise is formulated using the *random.gauss(mean, deviation)* function from Python's standard library. Disturbances and their respective Gaussian specifications are as follows:

- For orientation parameters like `pitch, roll, yaw,` and `heading`: Gaussian noise is added with a mean of 0.0 and a standard deviation of 10.0.
- For rate parameters such as `rollRate, pitchRate, yawRate,` and `altitudeRelative`: The noise has a mean of 0.0 and a standard deviation of 5.0.



(a) First Re-plan (b) Second Re-plan

Figure 24 – Re-planning sequence of actions. The black arrows show the previous sequence of actions, and blue arrows indicate the re-planning one.

- For thrust-related parameters `throttlePct` and `climbRate`: The disturbance is characterized by a mean of 0 and a standard deviation of 0.5.
- For `groundSpeed`: A Gaussian noise with a mean of 0.0 and a standard deviation of 1.5 is utilized.

In a Gaussian distribution with the above specifications, approximately 68.2% of the values will fall within one standard deviation from the mean, indicating that most of the disturbances will be bounded by the specified deviation values.

We propose a machine-learning system that employs Principal Component Analysis(PCA) since such a technique applies a dimensional reduction over the dataset from a higher space to a low dimension space. The advantage is to keep relevant relationships among variables or patterns from the original dataset. PCA also has the advantage of unsupervised learning since it does not need to know the value or label of target variables.

In this work, PCA assesses flight data to extract two components from the variables roll, pitch, yaw, heading, roll rate, pitch rate, yaw rate, climb rate, and throttle percentage, as illustrated in Figure 25.

Flight Type	Main Parameter	Start Altitude(m)	End Altitude(m)
Takeoff and landing	Takeoff Alt: 5 m Takeoff Alt: 15 m Takeoff Alt: 30 m		
Hovering	Hovering Alt: 5 m Hovering Alt: 15 m Hovering Alt: 30 m		
Forward flight	Distance: 10 m Distance: 15 m Distance: 5 m	15 m 50 m 7 m	10 m 30 m 5 m
Backward flight	Distance: 10 m Distance: 15 m Distance: 5 m	15 m 50 m 7 m	10 m 30 m 5 m
Left flight	Distance: 10 m Distance: 15 m Distance: 5 m	15 m 50 m 7 m	10 m 30 m 5 m
Right flight	Distance: 10 m Distance: 15 m Distance: 5 m	15 m 50 m 7 m	10 m 30 m 5 m
Circular flight	Radius: 5 m Radius: 15 m Radius: 20 m		
Climbing and descending		Start: 10 m Start: 20 m Start: 50 m	End: 5 m End: 10 m End: 30 m
Figure-8 flight	Radius 50		

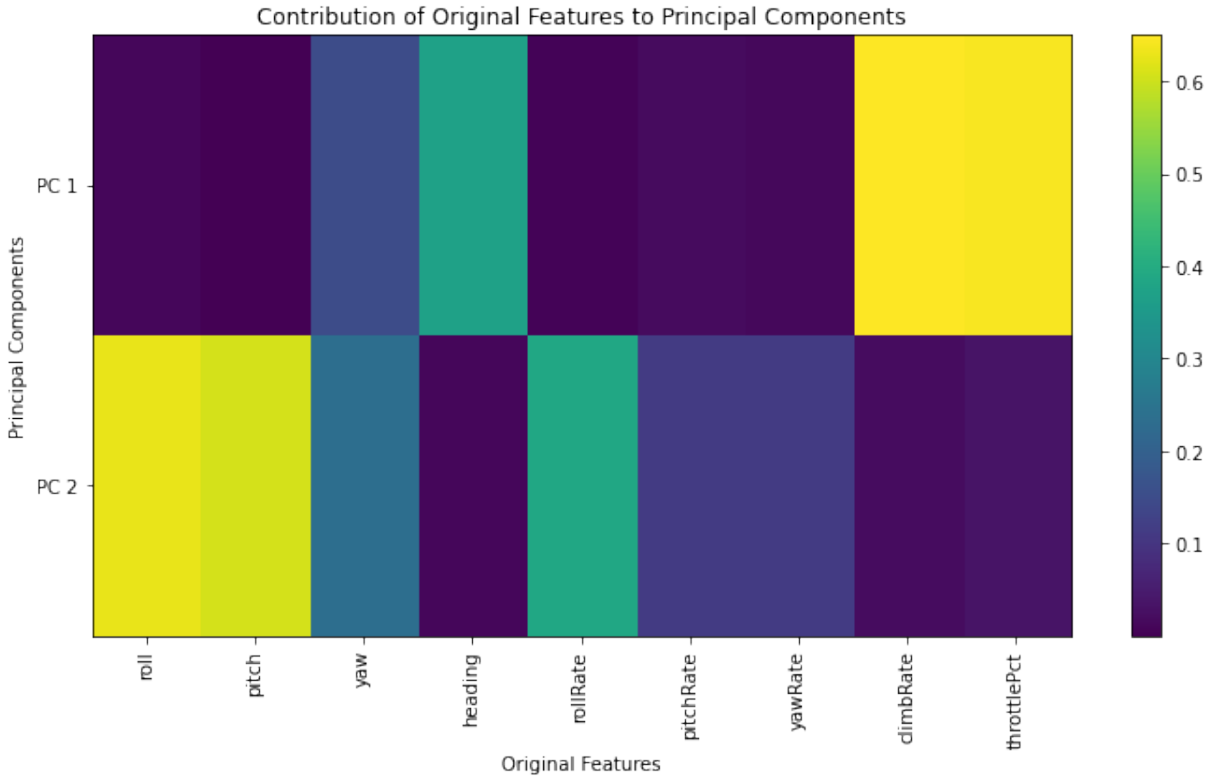
Table 3 – Flight parameters and types

The PCA components allowed us to plot and discern between standard flights and flights with errors. As observed in Figure 26a, even though there are 405 error-free flights, they cluster closely with two centers in the plot. In contrast, the 25% of flights with errors scatter widely. Thus, we defined a distance threshold for each flight data point to determine the severity of the error, as shown in Figure 26b.

Next, we use a decision tree classifier model to identify an error based on the UAV's parameters, as shown in Appendix A. A decision tree is a machine-learning approach that structures the knowledge learned through a hierarchy of decisions, refined until the final classification or regression result is obtained. The structuring approach allows us to better understand the decision-making process in a classification or regression problem.

The decision tree is a supervised learning algorithm, and we evaluated its performance on two distinct datasets: scaled UAV values and PCA-reduction values. Each dataset was partitioned into training and test subsets, allocating 80% of the data for training and the remaining 20% for validation. This configuration thoroughly assessed the model's efficacy on previously unseen data. For the UAV dataset, the classifier achieved:

Figure 25 – PCA components



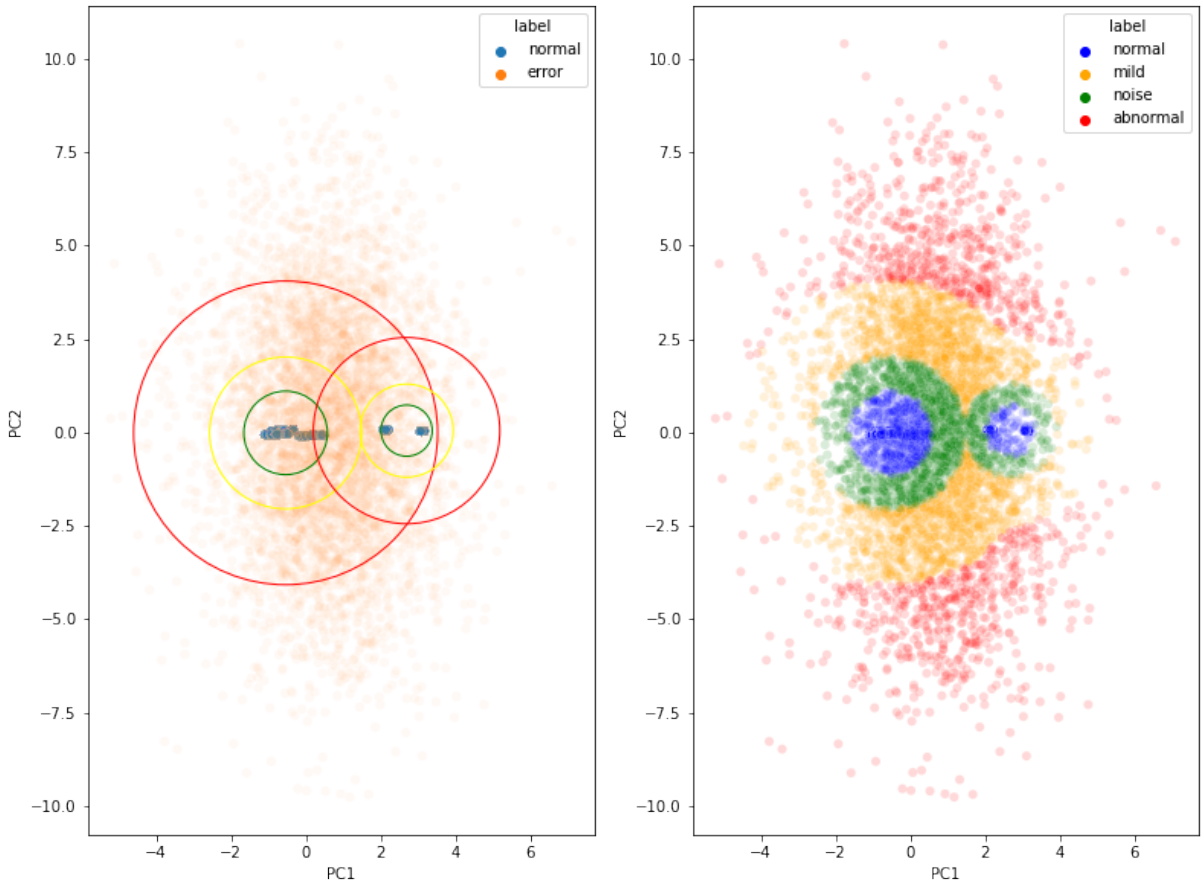
Source: Elaborated by the author.

- Accuracy: 0.9319
- Precision: 0.9323
- Recall: 0.9319
- F1-score: 0.9321

In contrast, when trained on the PCA values, the model's performance exhibited a remarkable improvement:

- Accuracy: 0.9954
- Precision: 0.9954
- Recall: 0.9954
- F1-score: 0.9954

Comparing the outcomes, it is evident that the Decision Tree classifier achieved superior results with the PCA values. This suggests that the PCA transformation might better represent this classification task compared to the raw scaled UAV values.



(a) Simulation values

(b) Class by PCA values

Figure 26 – Re-planning sequence of actions. The black arrows show the previous sequence of actions, and blue arrows indicate the re-planning one.

Fault Detection Module

Our system introduces an additional verification step to mitigate the risk of misidentifying normal conditions as faults (such as when a UAV encounters a warm air pocket) and avoid unwarranted mission interruptions. This module employs two user-defined time windows: Δ_t to appraise the occurrence of a flag given by the PCA-Tree combination and τ to inspect the consistency of percentage errors during the flight. For each time interval Δ_t, the module discerns and quantifies the instances of various error types, categorized as:

- α – percentage of noise.
- γ – percentage of mild conditions.
- β – percentage of abnormal conditions.
- λ – weighting factor of each identified problem.

The decision criteria follows, based on the $(\alpha, \gamma, \beta, \lambda)$ values estimated during Δ_t:

Algorithm 1 – Pseudo code for decision-making in Fault Detection

if $\alpha \leq 0.7, \beta = 0.0$ and $\gamma \leq 0.05$ **then**
    $\Delta_t \leftarrow 0$
else if $\alpha > 0.7, \beta = 0.0,$ and $0.05 < \gamma \leq 0.1$ **then**
    $\Delta_t \leftarrow \lambda^0$
else if $0.1 < \gamma \leq 0.4$ **then**
    $\Delta_t \leftarrow \lambda^1$
else if $\beta < 0.05$ and $0.4 < \gamma \leq 0.6$ **then**
    $\Delta_t \leftarrow \lambda^2$
else if $\beta < 0.1$ or $\gamma > 0.6$ **then**
    $\Delta_t \leftarrow \lambda^3$
else if $\beta < 0.2$ **then**
    $\Delta_t \leftarrow \lambda^4$
else
    $\Delta_t \leftarrow \lambda^5$
end if

The parameter Δ represents a specific λ value, and τ is the secondary time window, with t being the current time and the function $f()$ formulated as:

$$f(t) = \sum_{i=t-\tau}^{t} \Delta_i \tag{5.11}$$

Δ_i denotes the value at time i which will be in a time window from $\tau = t - \tau$ to t. The outcome of $f(t)$ within this window is pivotal for identifying anomalies. If $f(t)$ relies between $\lambda^{4.25}$ and $\lambda^{4.5}$ (exclusive), an anomaly warning is presented. If $f(t)$ escalates to $\lambda^{4.5} + 1$ or beyond, the user is prompted to either continue or check for false positives. If no user action is detected within 30 seconds or if the user agrees to abort, we have the following situations:

- The UAV is instructed to land instantly if $f(t)$ surpasses $1.5 \times \lambda^5$.
- Otherwise, the UAV endeavors to return to its base. In case of any complications with the `go_to_base()` function, an emergency landing is triggered.

5.2 Final Remarks

This chapter further expanded upon the methodologies, tools, and concepts for developing our autonomous system, building upon the previously established foundations of autonomy, code reusability, and programming language independence. Incorporating the KNN classifier underscores the importance of adaptive decision-making in selecting path-planning algorithms according to specific demands for trajectories. The system optimizes its path-planning choices by accounting for variables such as runtime, route length, and quantity of waypoints. The Mission Goal Manager system applies a KNN method to choose better path planners, where KNN and

planner are described. Risk Mitigation considers battery consumption with a BN introduced to forecast the need for re-planning, aiming to recharge the battery. For Fault Detection, we simulated flight scenarios with intentionally introduced errors; thus, it was possible to showcase the robustness and adaptability of our proposed solution. With the Gaussian noise simulations, the explained process of error generator provides a realistic approach to modeling real-world inaccuracies. PCA and Decision Tree algorithms allow our system to achieve error detection precisely.

CHAPTER

6

RESULTS

6.1 Introduction

The primary objective of this research was to comprehensively analyze the performance and behavior of the proposed autonomous system for drones under different mission scenarios. The main idea is to provide valuable insights into how the drone's system can achieve goals, handle errors, and manage faults. The results herein present a detailed overview of these findings, offering a holistic and granular view of the system's capabilities and potential shortcomings.

Harpia's systems are tested through extensive simulations in some scenarios, and we focused on evaluating how Risk Mitigation, Fault Detection, and Mission Goal Manager perform and interact with the other systems under specific challenges. We evaluate Risk Mitigation through different battery health conditions during the flight, a usual situation when dealing with drone missions. Fault Detection is simulated for the drone showing erratic flight behavior, which wind gusts or internal engine problems might provoke. The Mission Goal Manager must deal with changes in the mission during the flight when the ground station decides in real time to add or remove goals, another current demand for drone systems.

We report simulation results for a real-world quad-copter with the following configuration: 9 kg of Maximum take-off weight (MTOW), 5 m/s of topl cruising speed, 40 min of MTOW autonomy, and 7.0 g/w of MTOW Efficiency. Simulations will not replace the potential findings in real-world scenarios, and we tried to reduce such impact by using the real-world model provided by the Gazebo simulator. Thus, Gazebo PX4 Firmware is employed for drone simulation, and the POPF solver is executed to plan actions in PDDL 4.1. The Gazebo simulator executes 220 flights to gather the necessary data for our research, where these flight simulations spanned various scenarios.

The three distinct maps that define our simulation scenarios are in Figure 10, Section 3.1. Table 4 summarizes the number of Regions of Interest (ROI), Non-Fly Zones (NFZ), support

bases (Bases), the average distance in a straight line between all regions of the map (Avg. Dist) and the total of flight simulations for each map (Simulations).

Table 4 – Map information summary

Map	ROI	NFZ	Bases	Avg. Dist. (m)	Simulations
1	12	1	3	2300	60
2	6	15	3	550	80
3	6	6	3	150	80

Table 5 shows the basic configurations applied during simulations for each map. We have the number of Initial Goals (IG) of the mission and the label (number) of each initial goal region (IRG) in the map. Next, it is shown the label of each Added Region Goal (ARG) and Removed Region Goal (RRG) during the mission. Finally, we introduce noise during the flight to simulate fault detection. The last row shows three types of noise named Noise Type (NT): 1, 2, and 3. We explain each type of noise in Section 6.4. For example, let's assume the two configurations below:

C1: [IG, IRG, ARG, RRG, NT] = [2,(2,4),-,-,1]

C2: [IG, IRG, ARG, RRG, NT] = [2,(4,5),(1,2,6),4,-]

Configuration C1 states a simulation with two initial goals (IG) labeled as numbers 2 and 4 within the Map (IRG), and there is a noise type 1 (NT) being applied during the flight. There is no addition or removal of regions in C1. Configuration C2 defines a simulation with two initial goal (IG) labeled as 4 and 5 (IRG) with the addition of regions 1, 2, and 6 (ARG) and the removal of region 4 (RRG). Noise is not applied in C2.

The addition of new regions labeled in ARG always happens after the first initial goal (IG) has been reached and the action to be performed in such region is concluded. The same procedure happens if a configuration only removes a region (RRG) without adding others. However, for configurations simultaneously adding and removing regions, adding new regions happens after the action performed in the first initial goal has been concluded. A re-planning is done to deal with the new regions, and the drone heads to the next goal region. The request to remove regions occurs after the action conclusion in the next goal set in the first re-planning. At this point, another re-planning occurs for the new scenario with removed regions.

Table 5 – Description of the simulation scenarios

IG	2							4							6					
IRG	1,2	2,3	3,4	4,5	1,6	2,4	3,5	1,2,3,4	2,3,4,5	3,4,5,6	4,5,6,1	5,6,1,2	6,1,2,3	1,3,4,6	1,2,3,4,5,6					
ARG	-	1	-	1,2,6	-	-	-	-	1,6	-	2,3	-	-	-	-	-	5	-	-	-
RRG	-	-	3	4	-	-	-	-	-	5,6	2,3,4	-	-	-	-	4,5,6	4,5,6	-	-	-
NT	-	-	-	-	1	2	3	-	-	-	-	1	2	3	-	-	-	1	2	3

6.2 Mission Goal Manager, KNN & Planning Algorithms

The present section reports the results related to the mission planning and path planning problems that Harpia must face in the three maps. The solver POPF is applied to generate feasible plans for the mission planning problem, while KNN is employed to choose the better planner (HGA, PFP, or RRT) to solve the path planning problem. Thus, we evaluate the response provided by the Mission Goal Manager that will also trigger the Mission Planning and Path Planning systems. Next, we summarize the main features considered in Table 5:

- The missions have different numbers of goals, with a subset dedicated to error simulations with errors (noise) incorporated for nine flights.
- Two missions have new goals added by users from the ground station.
- Three flights have users removing original goals from the ground station.
- Three flights encompassing both added and removed goals.
- Three flights have noise added to the simulation.

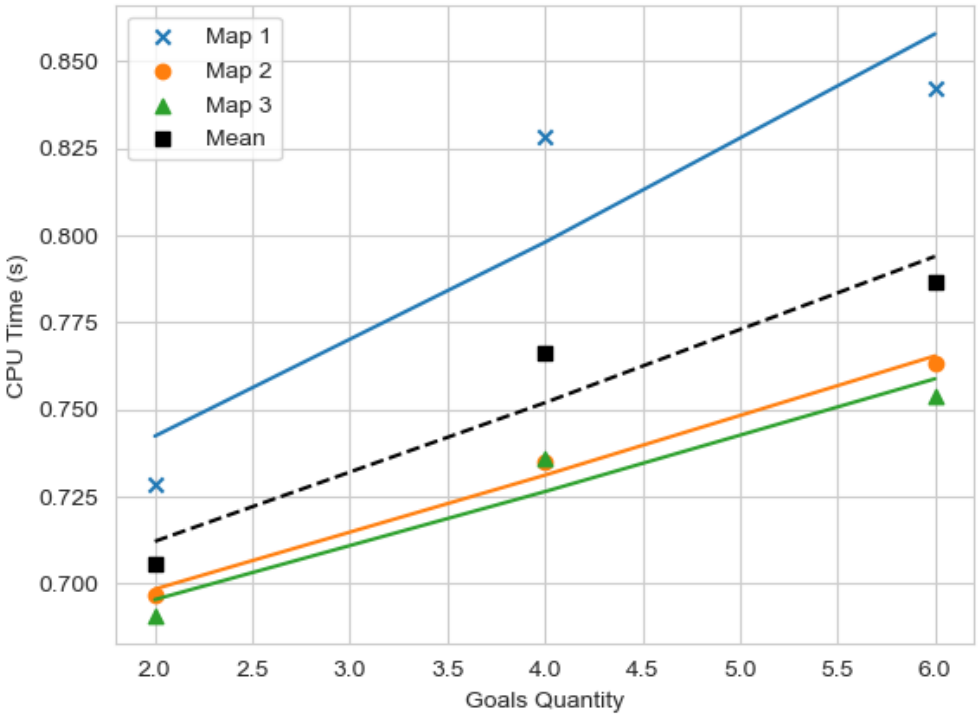
We collected the following data for every simulated flight:

- The plans generated for the mission.
- CPU time required for plan generation.
- Calculated Bayesian Network outcomes.
- Any need for re-planning during the mission.
- Every selection made by the KNN planner and the corresponding input values.
- The efficiency of the routes generated by the planner.

A total of 220 flight simulation was conducted, with 40 missions demanding a re-plan. A critical consideration in planning and re-planning the missions is the time expended by POPF solver formulating a plan. POPF operates as a non-deterministic algorithm, meaning it can not find a plan on the first try. Thus, Harpia will call the planner repeatedly until it generates a feasible plan, which can impact computational time based on the number of interactions between the Mission Goal Manager and POPF solver. Figure 27 and 28 show the average time spent on each map when the number of goals increases and a trend line. Figure 27 has the numbers for all plans and re-plans done, while 28 shows the figures only for re-plans.

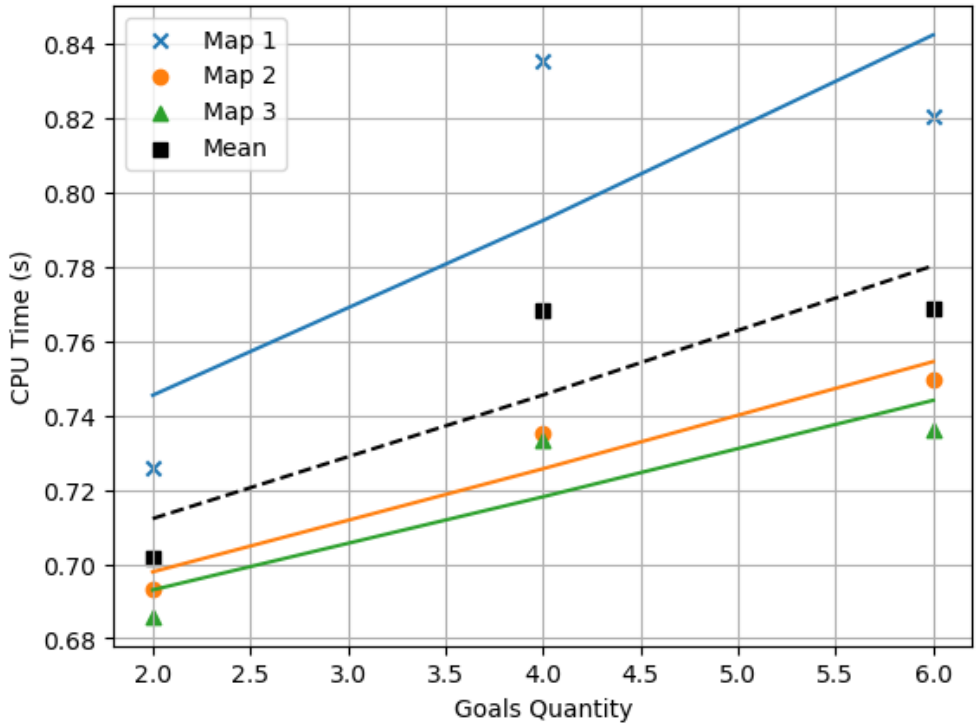
There is no relevant time difference for only re-plans against all plans and re-plans. In both cases, the solver POPF requires more time when time and distance increase, thereby

Figure 27 – CPU Time per Goals, with tendencies lines



Source: Elaborated by the author.

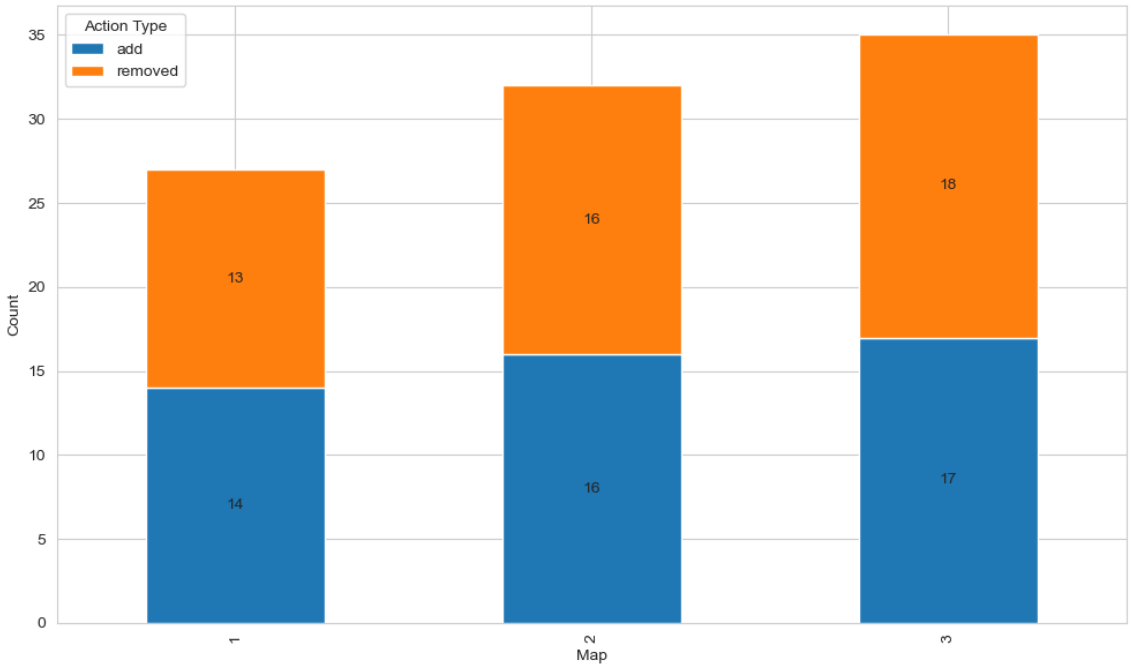
Figure 28 – CPU Time per Goals, with tendencies lines of re-plans



Source: Elaborated by the author.

intensifying the mission problem's complexity. On the other hand, even with the constraints introduced by adding or removing goals, the time dedicated to devising new actions for the drone never exceeds 0.85 seconds. Figure 29 shows the number of re-plans executed when adding or pulling out goals during the mission execution.

Figure 29 – Quantity of mission re-plan per map



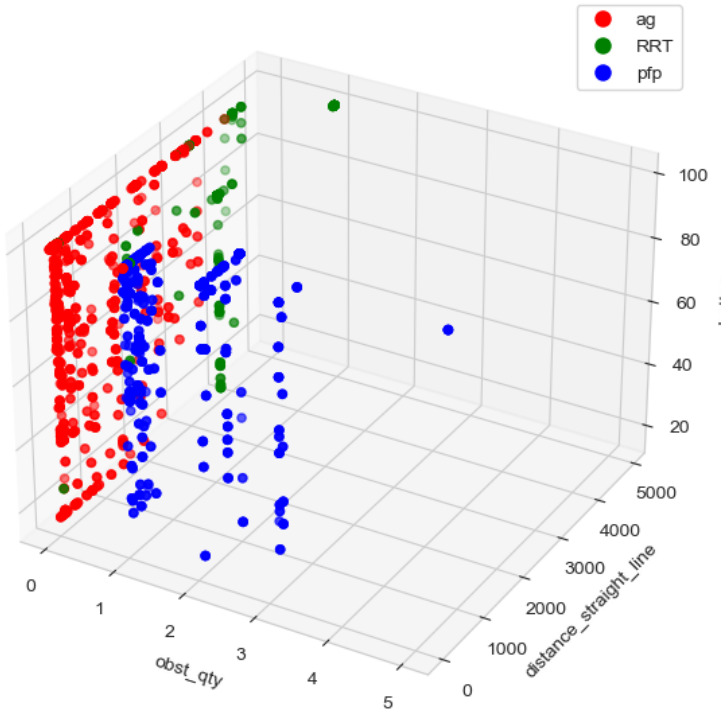
Source: Elaborated by the author.

Map 3 exhibits a higher number of re-plans compared to Maps 1 and 2. This difference arises due to the shorter distances in Map 3, leading to a larger tree of possibilities explored by the planner. Consequently, the planner occasionally encounters dead-ends, demanding additional iterations. However, the number of interactions between the Mission Goal Manager and POPF solver did not threaten the overall computational performance for real-time decision-making.

Figure 30 has the distribution of algorithms selected for each request of the path planning system during the simulation. We counted a total of 2204 total requests for path planning during all simulations. The path planning system, based on KNN classification, chose Hybrid Genetic Algorithm (HGA) for 52.8% of the simulations, followed by PFP with 32.7%, and RRT with 14.5%. The accuracy of KNN algorithm achieved 84.1% with 15.9% infeasible routes returned by the selected planners.

The number of obstacles (obst_qty) and the estimated distance among goals (distance_straight_line) impact the path planner chosen by KNN as shown in Figure 30. KNN chooses HGA more often when there are no obstacles, mainly for short and average distances, while RRT appears to solve path planning problems with obstacles for all distances of goals. Finally, KNN selects PFP when we have obstacles and short distances among goals. However, it's noteworthy that the PFP generated all infeasible routes. Whenever infeasibility arose, the system

Figure 30 – KNN results from simulation



Source: Elaborated by the author.

immediately selected RRT as a replacement planner once it performed better, finding viable trajectories quickly.

Table 6 gives some insights about the infeasible paths returned by PFP, where the method found only infeasible routes in Map 2 and 3. We have the number of infeasible paths (Paths) generated, the average amount of obstacles (Avg. Obst) from the start point in a region to the endpoint in the next region of the path planning, the average distance between regions (Avg. Reg Dist), and the average battery charge (Avg. Battery) at the moment that PFP is executed. The column Avg Reg Dist gives the average values based on a straight line from one region to another. Thus, we can see that PFP could not return a trajectory even with only one or two obstacles being avoided and enough battery to execute a trajectory. The average distance between regions seems not to be the reason once PFP failed for Map 2 with 590.45 m on average and Map 3 with 181.25 m.

Table 6 – Unfeasible PFP paths info

Map	Paths	Avg. Obst	Avg. Reg Dist	Avg Battery
2	149	1.483	590.45	79.342281
3	202	1.504950	181.25	75.262375

Table 7 summarizes the results for each path planner with feasible trajectories across the different maps. **Avg. Obst** shows the average amount of obstacles between two regions, as stated in Table 6. **Avg. Route Dist** gives the average length of the planned path, while **Avg. Battery**

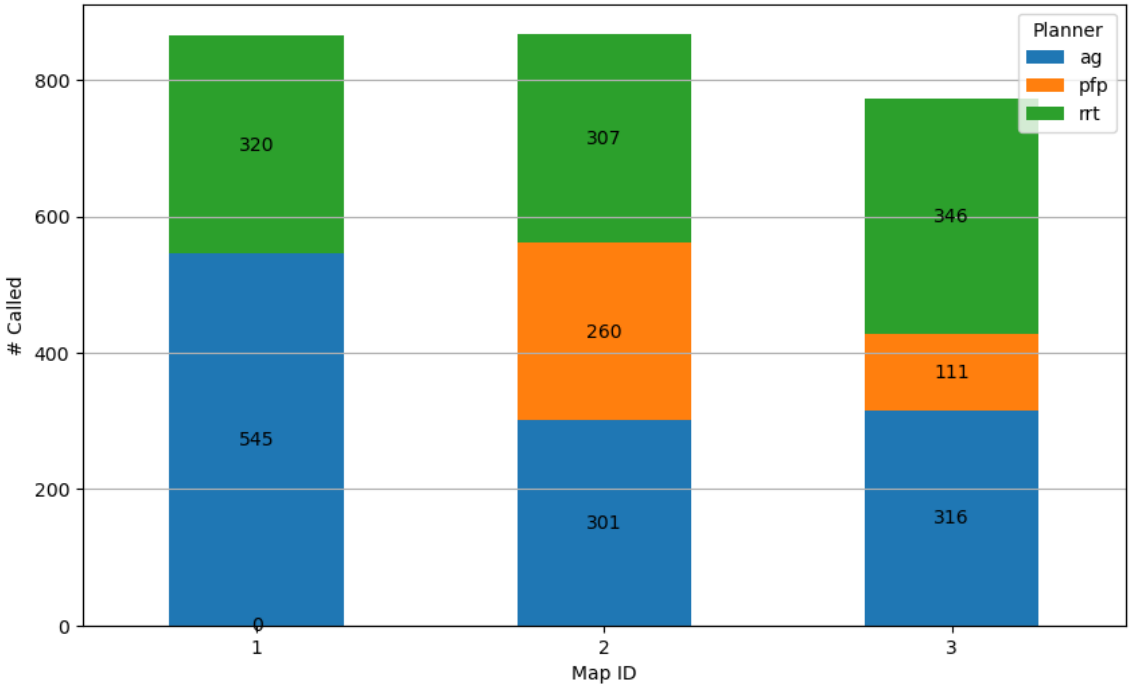
Planner	Map	Avg. # Obst	Avg. Route Dist. (m)	Avg. of Waypoints	Ratio (%)	Planner time(s)
HGA	1	0.0	1781.24	7.7	1.13	120.08
	2	0.0	300.69	6.3	1.16	120.19
	3	0.02	97.36	6.02	1.16	120.18
PFP	2	1.53	707.87	34.9	1.10	23.57
	3	1.54	184.11	10.0	1.16	3.05
RRT	1	0.60	3179.97	154.34	1.25	0.65
	2	1.47	646.24	34.19	1.27	1.02
	3	1.4	182.93	15.78	1.99	1.04

Table 7 – Planning Algorithms results/performance

also shows the battery charge to plan the path. **Avg. of Waypoints** has the mean of waypoints each method spends when returning a trajectory.

The HGA is typically invoked when there are no or fewer obstacles, while KKN triggers RRT and PFP to handle obstacle avoidance. HGA returned, on average, long trajectories for Map 1 that are expected once it is the widest map; however, generates trajectories with few waypoints for all Maps. RRT returns long trajectories for all maps, mainly for Map 3; therefore, building trajectories with many waypoints. These values of RRT are explained by the need to deviate from obstacles once it is the planner more often called for obstacle avoidance. HGA runs for 120 sec, while RRT and PFP run until they find a trajectory. RRT has the advantage of being the fastest planner for all maps. PFP offers a commendable balance between path length, time, and number of waypoints but failed to return feasible solutions within 15.9% of the maps. Figure 31 shows the number of Path Planning system calls from KNN in the Mission Goal system.

Figure 31 – Planners Calls per Map



Source: Elaborated by the author.

KNN did not select PFP for path planning in Map1, making 260 and 111 calls for Map 2 and 3, respectively. Thus, PFP can be a planner not adjusted to handle long distances, which is the case for Map1. KNN employs HGA and RRT in all maps, with HGA being called by a similar number of times and RRT executed more than 500 times for Map 1 and around 300 for Map 2 and 3.

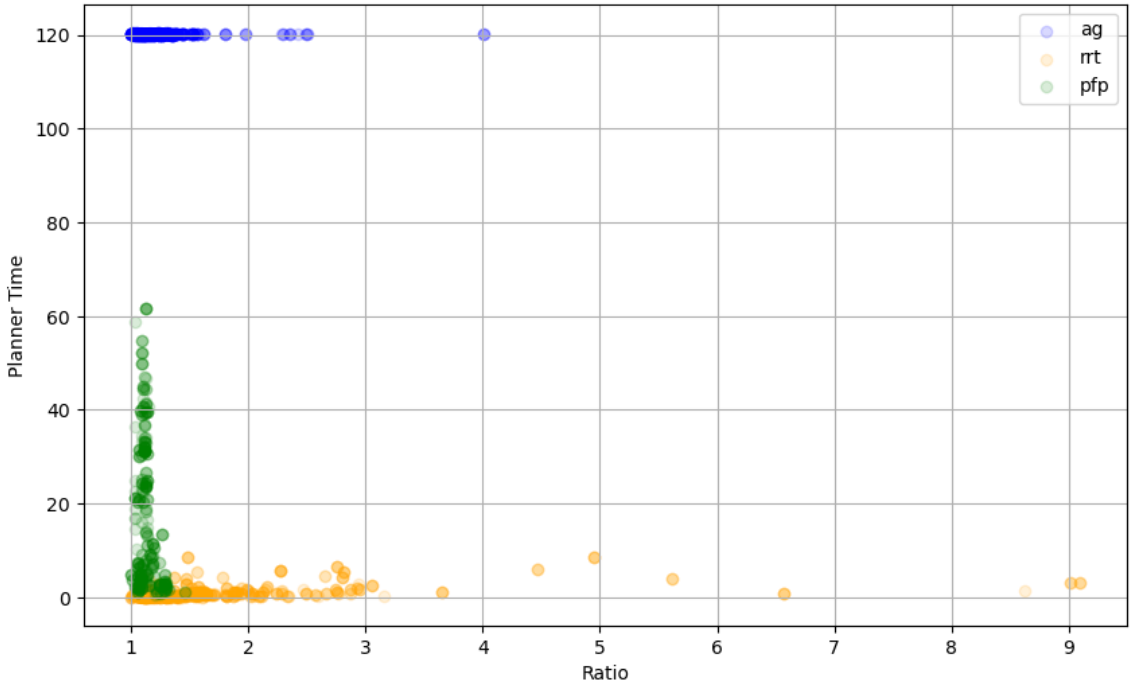
We state Ratio to compare the length of paths planned against the distance using a straight line between two regions, described by Equation 6.1:

$$Ration(\%) = \sum_{i=1}^{R} \sum_{\substack{j=1 \\ i \neq j}}^{R} \frac{1}{\Pi_{i,j}} \left(\sum_{p=1}^{P} \pi_{i,j}^{p} \right) \tag{6.1}$$

- R: set of region in the map.
- P: number of paths.
- $\pi_{i,j}^{p}$: length of path p from region i to j.
- $\Pi_{i,j}$: straight line distance from region i to j.

In Figure 32, we compare the ratio values and the time spent to plan the trajectory. All planners take more time to find paths with better ratios, so more straight trajectories can be time-consuming. HGA is shown but we most remember that it is always executed for 120s.

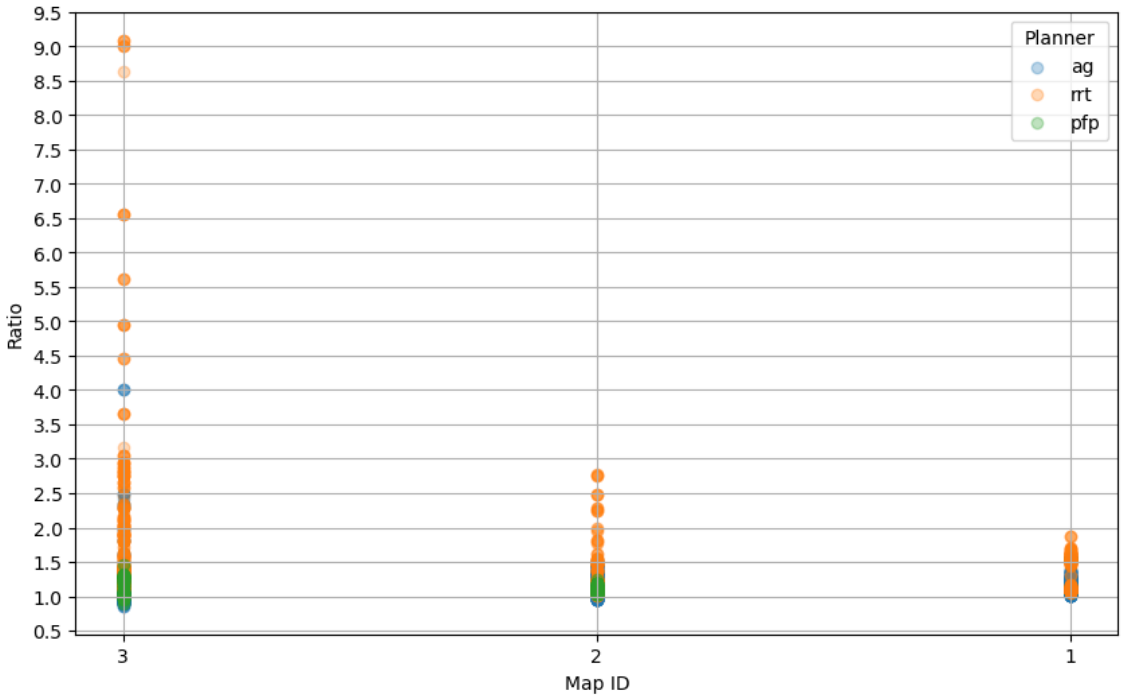
Figure 32 – Planner time per ratio



Source: Elaborated by the author.

The ratio by Map is shown in Figure 33, where most ratio values are below 3.0 for all planners. RRT had some problems finding low ratios for Map 3, which makes sense based on its path lengths shown in Table 7. Thus, the KNN selects a planner that returns a straight trajectory the majority of times. In this case, the planners avoid large detours from obstacles, even for maps with different dimensions and quantities of NFZ and ROI.

Figure 33 – Planners ratio per Map



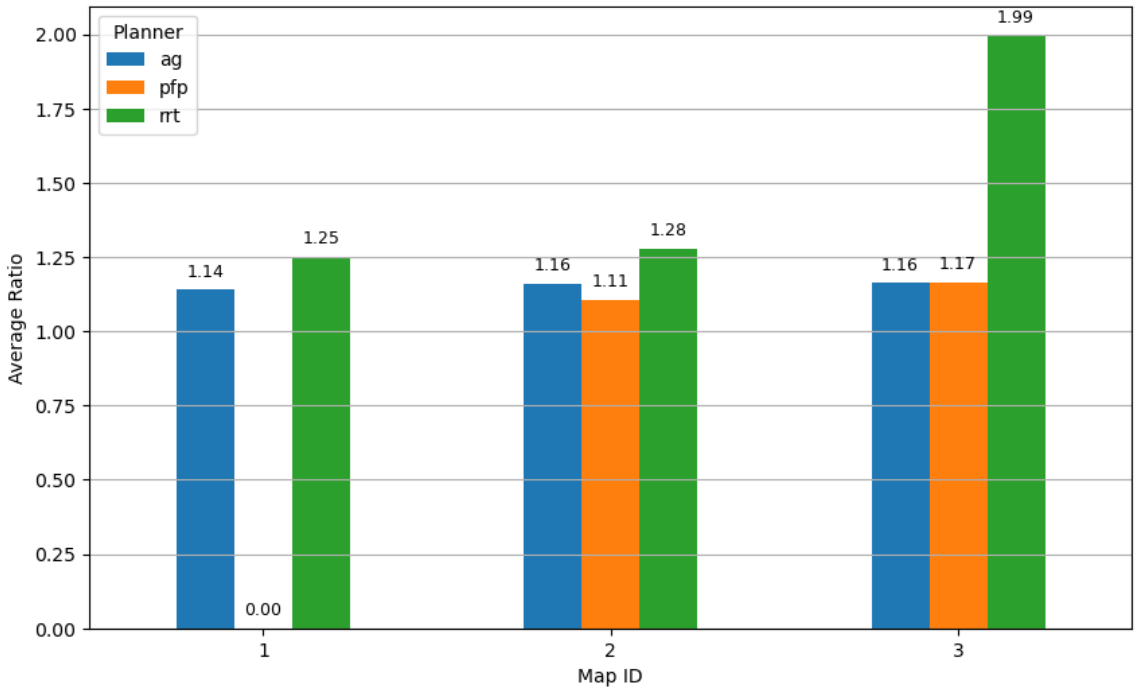
Source: Elaborated by the author.

Figure 33 compares the ratio with the straight-line distances. We can see now why RRT has problems finding a low ratio. RRT has problems finding a more straight trajectory for short distances between regions but has no problem for long distances. HGA also has no problems in returning a straight trajectory for long distances. PFP has a low ratio in most results but is limited to distances below 1000.

Finally, Figure 34 brings the ratio mean for each planner and map. The deviation from a straight line did not exceed 20% on average for HGA and PFP for all maps. RRT is the exception, with a deviation above 20% for Map 1 and 2 and a huge deviation in Map 3.

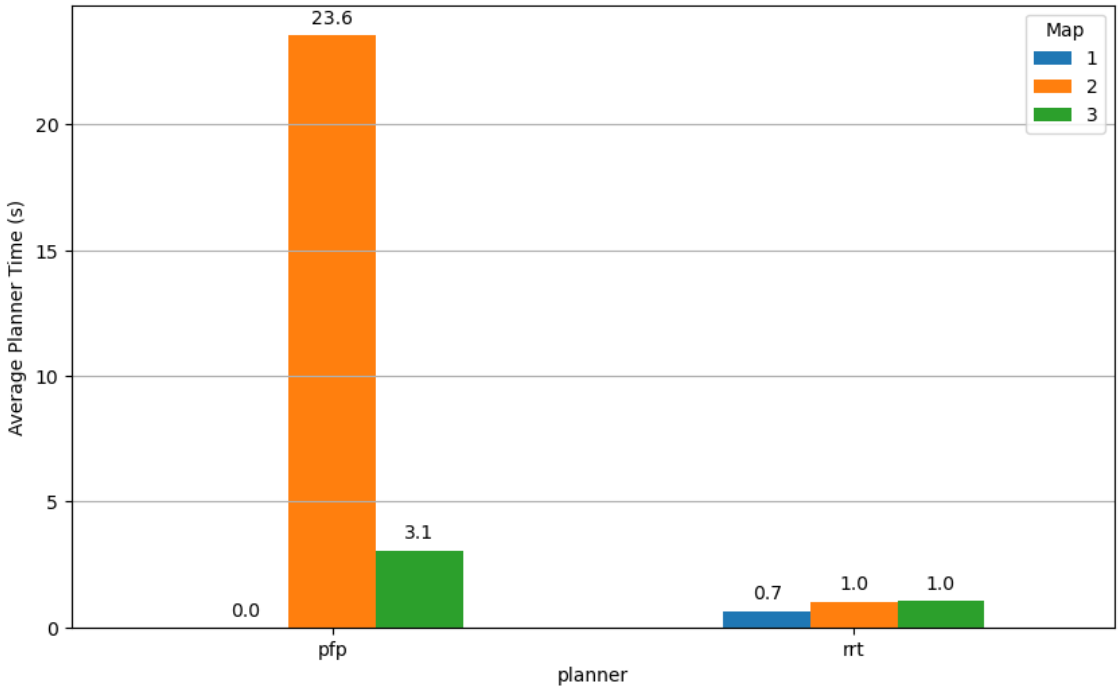
Figure 35 presents the CPU time for each planner and map, with HGA excluded from the analysis due to its imposed time limit of 120s. Furthermore, the PFP was not selected as a planner in Map 1. PFP requires more computation time in Map 2, which is attributed to the higher number of obstacles in that particular map. Conversely, RRT planner exhibits consistent behavior between Map 2 and 3. It has to avoid a single No-Fly Zone in Map 1, generating routes with a marginally shorter execution time, approximately 0.3 seconds less than in maps containing multiple obstacles.

Figure 34 – Average planner ratio per map



Source: Elaborated by the author.

Figure 35 – Time consumed to generate a route per map



Source: Elaborated by the author.

Figure 36 illustrates the path information during a mission. Let's consider C_n as follows: $C_n : [2,(1,2,3,4),-,-,-]$, and the plan generated, represented by the white arrows in Figure 36a. KNN must select a planner to reach the first Region of Interest (ROI), as shown in Figure

36b, with no obstacles to avoid, a battery level of 100%, and a straight-line distance of 323.77m. HGA is chosen and generates a feasible route with a length of 446.64m and a ratio of 37%, shown by the red line.

This process is repeated in Figures 36c and 36f. In Figures 36d and 36g, the KNN invokes the PFP, but it returns an infeasible route, specifically, a route in which the last waypoint's distance to the destination waypoint exceeds 10m. When an infeasible route is returned, the system calls upon the RRT planner, as shown in Figures 36e and 36h.

6.3 Risk Mitigation: Battery-Level & Bayesian Network

Risk Mitigation system in Harpia employs a Bayesian Network (BN) to evaluate the battery consumption and decide the need for a mission re-planning. Thus, battery consumption is evaluated through flight simulations under four distinct battery health conditions: 100%, 70%, 50%, and 30%. Thus, the simulations in Table 5 are executed for each one of the battery conditions. Figure 37 illustrates different temperature conditions that change the battery's capacity.

In our simulation, under a battery health of 100%, the drone can achieve a flight duration of 40 minutes. As the battery health decreases, the flight time is impacted; e.g., the drone's flight duration is reduced to 28 minutes at 70% battery health. When the battery health drops to 50%, the flight time further decreases to 20 minutes, and at 30% battery health, the drone can only keep a flight for 12 minutes. A total of 20 flights were simulated for each of the battery conditions.

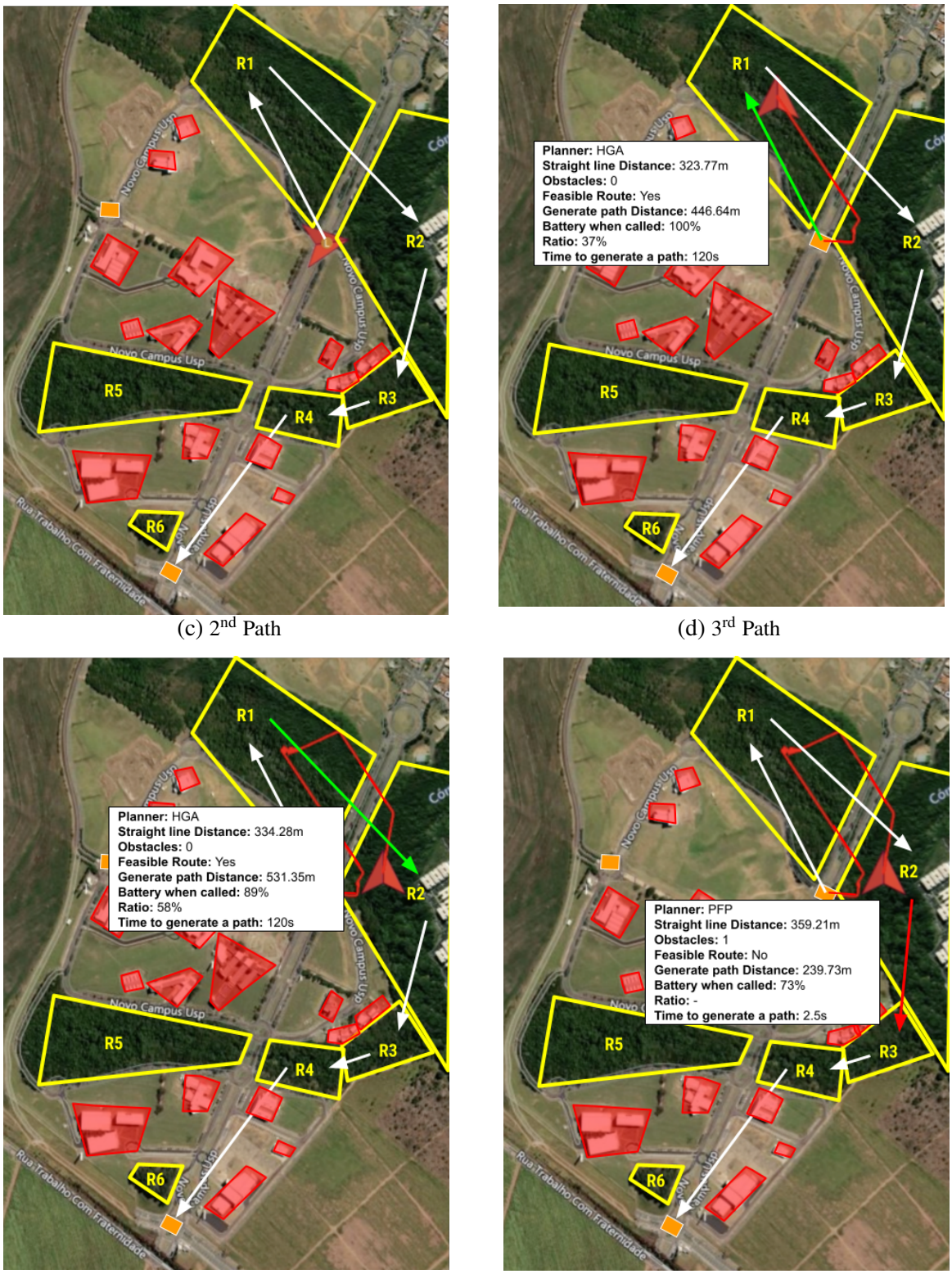
Thus, we assessed the responsiveness of the Mission Manager through ROSPlan (Mission Planning) and BN to detect issues and re-plan actions through four distinct battery health levels. For example, if a drone is designed to fly for 40 minutes but operates with 50% of battery health and a discharge rate twice the usual, its flight duration would be reduced to 20 minutes. However, the minimum distance between the launch site, region, and base is around $3km$ in Map 1, and such distance becomes unfeasible for a drone with only 30% battery health available to reach a base. Therefore, we resume our testing with the 30% battery health scenario only to Map 2 and Map 3.

There is no need for mission re-planning when the drone works with 100% of battery health in all maps, and re-plan did not happen when battery is less than 20%. We assume as a critical level when the battery falls below 20%, which is often for flight simulations with battery health set as 30%. In this case, spending time calling the solver and waiting for the re-planing is not the best decision. Therefore, the system sends the drone to execute a detour to the nearest base, proceeding with the previous plan after recharge.

Figure 38 demonstrates an increase in re-planning calls as battery health diminishes from 70% to 30%. Map 1 has only one NFZ but within a wide area ($2300m$); therefore, the BN and

(a) Plan

(b) 1st Path



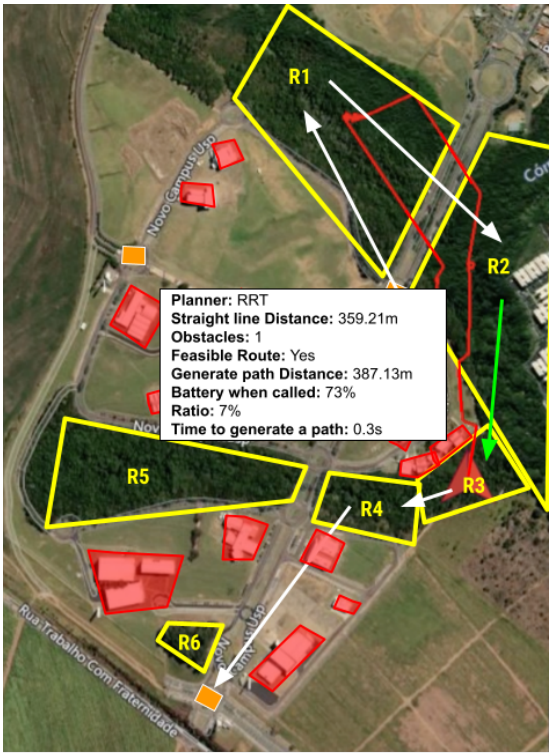
(c) 2nd Path

(d) 3rd Path

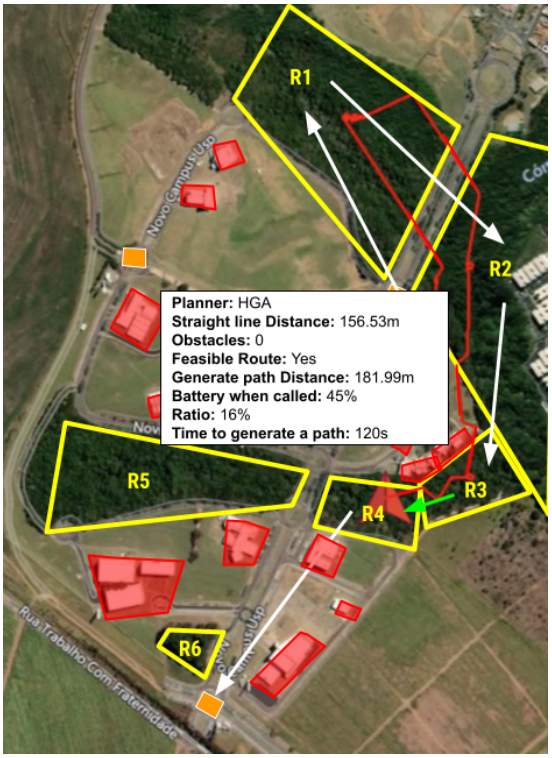
Figure 36 – plan x path plan

re-planning system of Harpia takes a little more than 0.5s to re-plan the mission, which also happens for Map 2, when plenty of battery charge is available (70%) in both cases. This short time for re-planning indicates that POPF solver had no problem easily finding a new plan. Map 3 witnessed no re-planning at 70% battery health since the maximal distance between interest

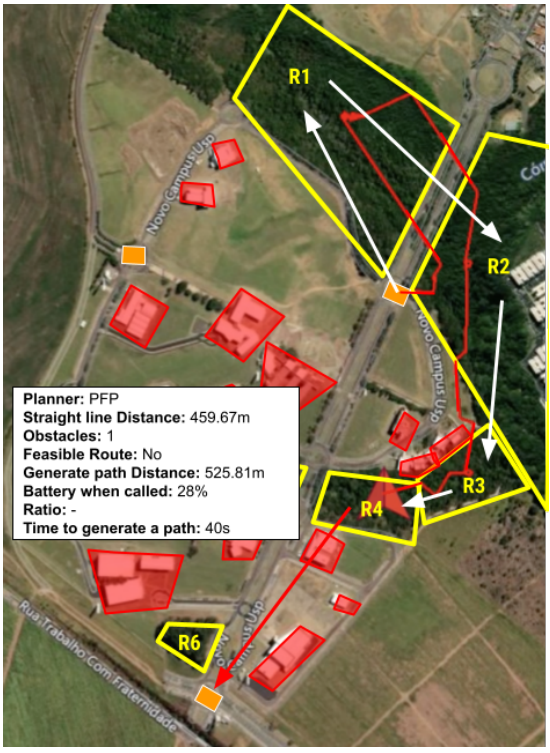
(e) 3rd Path



(f) 4th Path



(g) 5th Path



(h) 5th Path

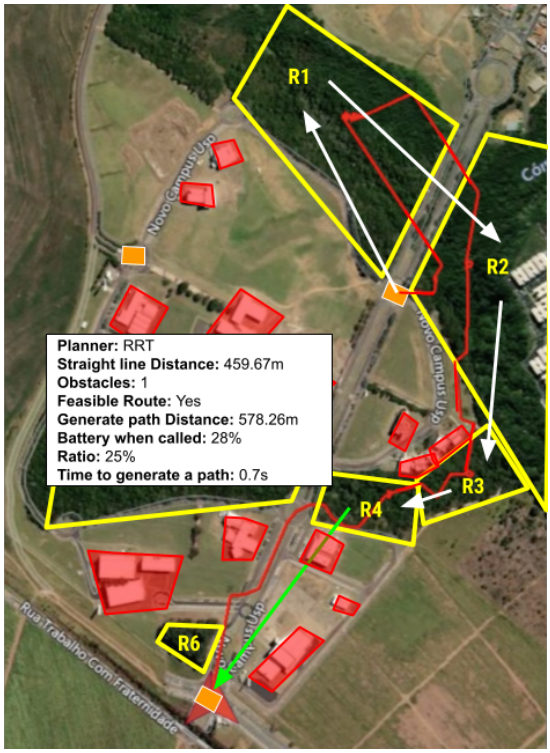
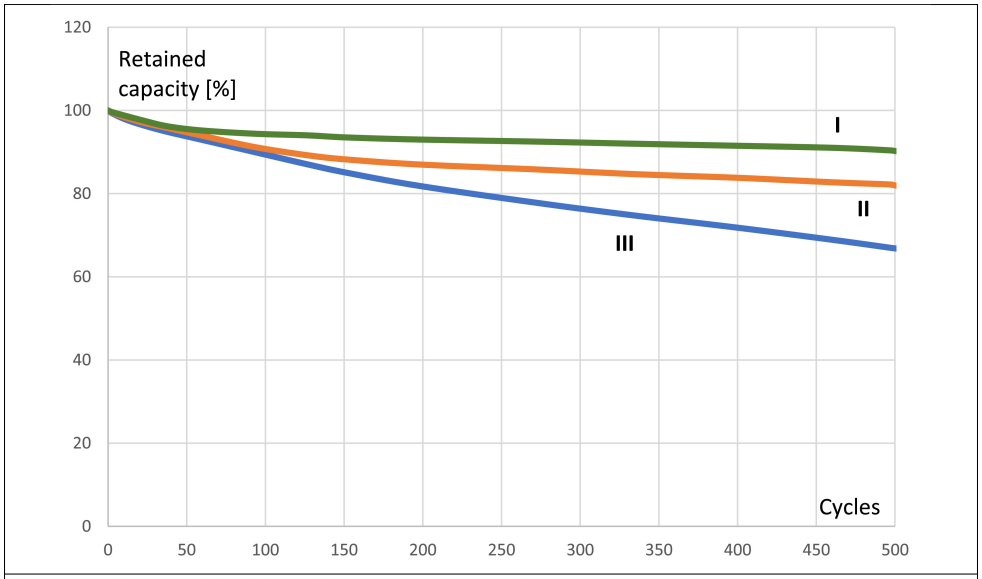


Figure 36 – plan x path plan

zones is only 440m, and the flight capacity stands at 28 minutes.

There is a relevant increase in re-planning for battery health levels of 50% and 30%

Figure 37 – Typical curve of battery degradation under different test conditions

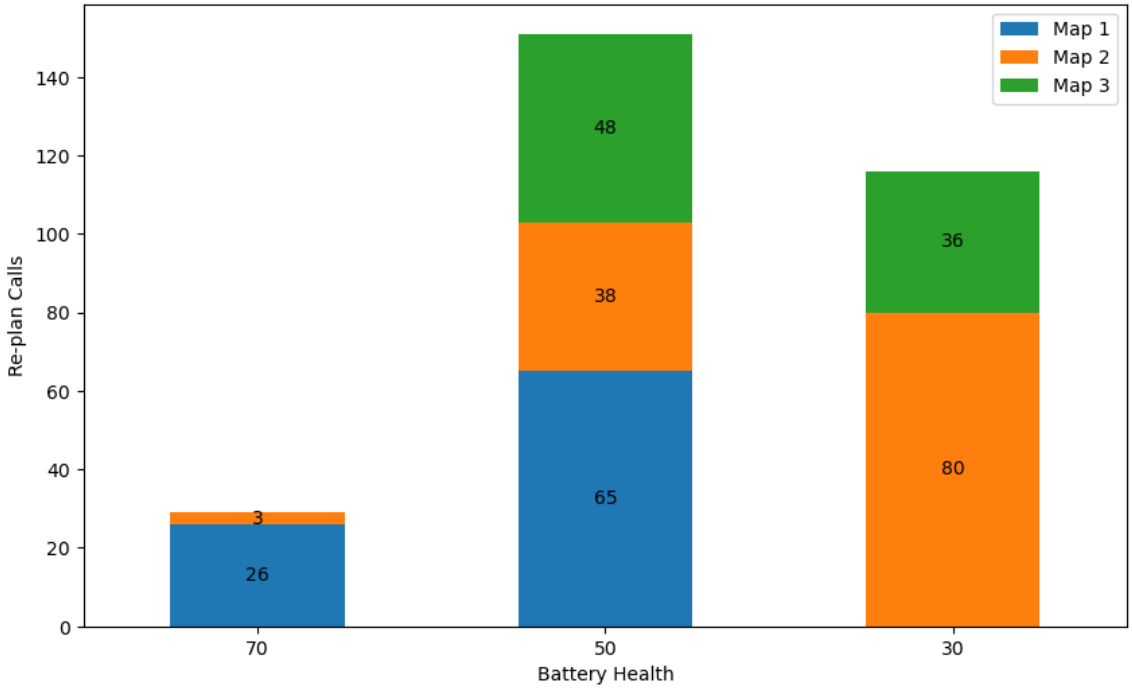


I: Temperature 25°C, charge 0.5C, discharge 1C, partial cycling (90% capacity).
II: Temperature 25°C, charge 0.5C, discharge 1C, full cycling.
III: Temperature 25°C, charge 0.5C, discharge 2C, full cycling.

Source: Zubi *et al.* (2020).

as expected. However, the battery reaches a critical level below 20% more often for flight simulations with 30%, when the BN did not call a re-plan and the drone only makes a detour to the nearest base. This explains the reduction in the calls from 50% to 30% of battery health.

Figure 38 – Mean of Re-plan Calls per Battery Health



Source: Elaborated by the author.

6.4 Fault Detection and Management

Fault Detection system must deal with the drone showing erratic flight behavior, which wind gusts or internal engine problems might provoke. The proposed PCA and Decision tree integration is applied as part of Fault Detection system to identify erratic flights. In situations where faults were detected, we recorded additional data:

- The time of the fault diagnoses.
- Type and percentage of errors for the corresponding time window.
- Historical data from previous time windows.
- The corrective action suggested by the system.
- Any user's feedback, particularly if they decided to continue the mission, signaling that the UAV was not perceived as in any immediate danger. The idea is to identify if the detected error was a false positive.

We introduced different levels of Gaussian noise in some simulations to assess the system's capability when identifying non-expected behaviors. Thus, the standard deviation of the Gaussian distribution was varied to simulate disturbances by adding three types of noise. The Noise Type 1 was added using the parameter specifications next:

- For parameters such as `pitch, roll, yaw,` and `heading`: Gaussian noise with mean 0.0 and standard deviation 3.0.
- For parameters like `rollRate, pitchRate, yawRate,` and `altitudeRelative`: Gaussian noise with mean 0.0 and standard deviation 2.0.
- For parameters `throttlePct` and `climbRate`: Gaussian noise with mean 0 and a standard deviation 0.2.
- For all other parameters: Gaussian noise with mean 0 and standard deviation 0.8.

Noise Type 2 was set as follows:

- For parameters such as `pitch, roll, yaw,` and `heading`: Gaussian noise with mean 0 and standard deviation 7.0.
- For parameters like `rollRate, pitchRate, yawRate,` and `altitudeRelative`: Gaussian noise with mean 0 and standard deviation 5.0.
- For parameters `throttlePct` and `climbRate`: Gaussian noise with mean of 0 and a standard deviation of 0.5.

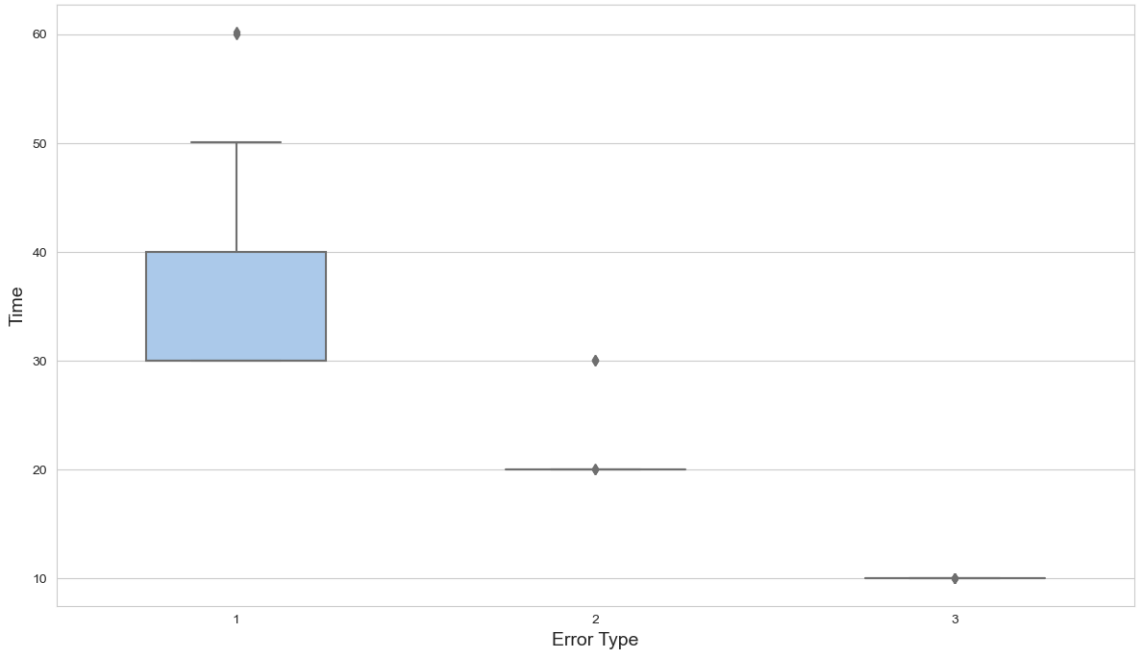
- For all other parameters: Gaussian noise with mean 0 and standard deviation 1.5.

Finally, the Noise Type 3 was considerably accentuated as follows:

- For parameters like `pitch, roll, yaw,` and `heading`: Gaussian noise with mean 0 and standard deviation 15.0 is added.
- For parameters such as `rollRate, pitchRate, yawRate,` and `altitudeRelative`: The Gaussian noise with mean 0 and standard deviation 7.0.
- For parameters `throttlePct` and `climbRate`: Gaussian noise with mean of 0 and standard deviation of 1.0.
- For all other parameters: Gaussian noise with mean 0 and standard deviation 3.0.

Figure 39 has a boxplot about the time the system requires to detect a flight fault. Given the system conditions detailed in Section 5.1.5, Harpia demonstrated an aptitude to identify and respond to flight errors, particularly as error magnitude increased. Some anomalous data in Figure 39 took around 60s to be certified as type 1 noise. Most type 1 noise detection happens from 30 to 40s sec and has a maximum value of 50s. Noises of types 2 and 3 classifications occur more stable, with type 2 being identified around 20s and type 3 around 10s.

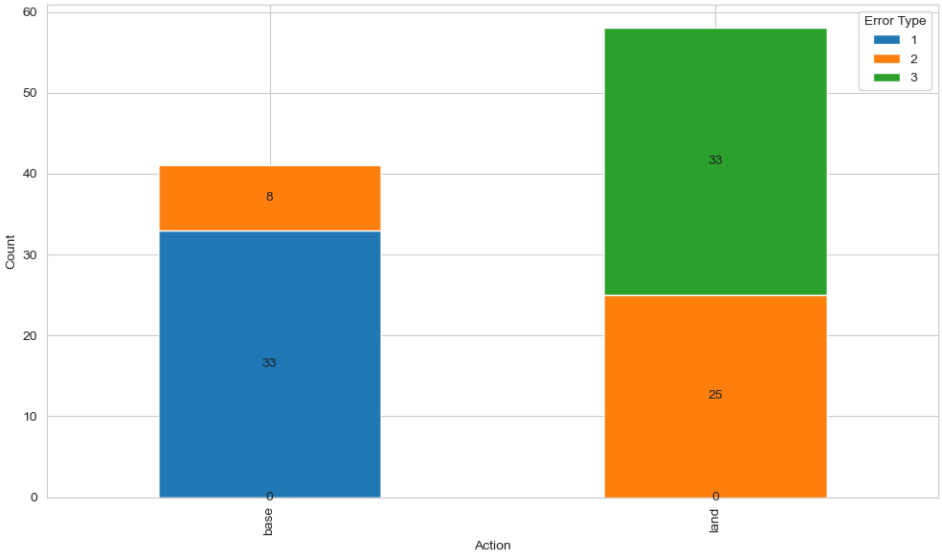
Figure 39 – Time to identify a fault flight per type of error type



Source: Elaborated by the author.

Figure 40 outlines the system's actions for each error type: return to base or land in a nearby area. The system's responses aligned with expectations by selecting the nearest base for all the low noise-affected flights and deciding for immediate landing in the case of severe noise.

Figure 40 – Chosen actions per error type



Source: Elaborated by the author.

In the simulation with Noise Type 1, the system triggers an action return to base 33 times out of 33 simulations with no land action performed. The lack of land action is an expected answer from the system for a low noise level. Noise Type 2 triggered a return to base action eight times and a land action 25 times, which indicates a level of disturbance demanding a land action more often. This behavior is expected since the Gaussian distribution for Type 2 can generate values akin to noise Type 1 and Type 3. Finally, Noise Type 3 produced the opposite result from those seen in simulations with Type 1 noise. There is now a higher need for land action by executing such action 33 times without any return to base action triggered.

6.5 Summary of Results

This chapter reports results by simulating a real-world quad-copter employing Gazebo PX4 Firmware and the POPF solver in a PDDL 4.1 environment. The chosen drone model showcased an efficient velocity of $5.0 m/s$, and a commendable forty minutes of battery autonomy. A total of 220 flights were simulated, covering diverse battery health levels and mission goals within three maps. The aim was to evaluate the Mission Goal Manager, Risk Mitigation, and Fault Detection systems in Harpia.

In the Mission Goal Manager, the planning and re-planning times remained enough to formulate a feasible plan or re-plan and generate new trajectories for such mission changes. The results indicate that the time for planning actions never surpassed 0.85 seconds. Battery health significantly influenced flight feasibility and the need for re-planning, posing a challenge for the Risk Mitigation system. Lower battery health levels (30% and 50%) witnessed more re-planning calls, especially in larger maps. Notably, even with the increase in re-planning calls,

the mean re-planning duration remained consistent, reflecting the system's robustness. Fault Detection system was tested against different types of disturbances, introducing varying levels of Gaussian noise. The results indicate a proactive system response, adeptly handling errors, with actions ranging from proceeding to the nearest base for mild disturbances to initiating immediate landings for severe ones. The classification of error types done by the PCA and Decision tree methods satisfactorily addressed the challenge of identifying possible threats.

KNN is another machine learning approach that satisfactorily addressed the challenge of choosing the path planning planner. The method selected the Hybrid Genetic Algorithm (HGA) that dominated with a 52.8% selection rate, outperforming PFP and RRT. However, the KNN algorithm's accuracy stood at 84.1%, indicating a certain degree of infeasibility. Notably, PFP, despite showing a good balance between time and ratio, failed to produce a feasible route occasionally, which sum up 15.9% of the infeasible routes. On the other hand, besides producing long paths, RRT is the best to find paths, mainly when obstacle avoidance is necessary quickly.

In summary, the advanced drone system exhibits a promising performance profile, especially in challenging mission scenarios. The analysis not only underscores its strengths but also pinpoints areas that could benefit from further enhancements. As drone technology continues to evolve, these insights will be invaluable in ensuring safe, efficient, and reliable UAV operations.

CHAPTER

7

FINAL REMARKS

This doctorate thesis extensively explored an advanced drone system's performance and behavior across various mission scenarios. Our research aimed to bridge the gap between the challenges of developing autonomous drones and the goals of enabling them to adapt to real-time changes with minimal human intervention. In this concluding section, we synthesize the main findings and their implications for the overarching question: Is it possible to develop a system that integrates AI capabilities for autonomous decision-making within different scenarios where real-time changes must be addressed with minimum human intervention?

Our journey began by identifying the critical challenges that autonomous drone systems face. These challenges included optimizing mission management, ensuring efficient path planning, and enhancing fault detection and response mechanisms.

Firstly, our research aimed to comprehensively analyze the impact of varying battery health levels on mission feasibility. The results unequivocally demonstrated that battery health is pivotal to mission success. As battery health declined, the system responded with increased re-planning, showcasing its ability to adapt to changing conditions autonomously.

Secondly, we sought to understand the efficiency of the path-planning algorithms used in autonomous drones. Utilizing a K-nearest neighbors (KNN) classifier for algorithm selection played a central role in achieving this goal. The Hybrid Genetic Algorithm (HGA) emerged as the predominant choice, but the occasional infeasibility uncovered by the KNN, reveals that PFP algorithms underscored the need for further refinements.

Lastly, we aimed to investigate the system's fault detection and response capabilities under various disturbance levels. The simulation results showed the system's aptitude for identifying anomalies and responding accordingly. The system's adaptability was evident in its choice between returning to base or initiating emergency landings, mirroring the severity of the disturbances.

Is it possible to develop a system integrating AI capabilities for autonomous decision-

making within different scenarios with minimal human intervention? Our findings suggest that it is indeed possible. The drone system exhibited a remarkable capacity to autonomously address the challenges posed by real-time changes. As demonstrated in our extensive simulations, the robustness and efficiency of the system's autonomous decision-making capabilities align with the overarching goal of minimizing human intervention.

The system's ability to adapt to fluctuating battery health, select suitable path-planning algorithms, and respond effectively to disturbances illustrates its potential to operate autonomously across diverse scenarios. These results indicate a promising path toward developing advanced drone systems to make real-time intelligent decisions, reducing the need for continuous human oversight.

As we conclude this thesis, we acknowledge that there is still work to transition these findings from simulations to real-world applications. Nevertheless, the foundations here provide a compelling case for the continued development and refinement of autonomous drone systems. These systems hold immense potential in areas such as surveillance and delivery services and in addressing broader societal challenges and opportunities.

In summary, this doctorate thesis has illuminated the path toward autonomous drone systems that can navigate complex scenarios with minimal human intervention. The fusion of AI capabilities and real-time decision-making is within reach, promising a future where drones operate seamlessly and effectively across many dynamic environments.

All the documentation and code implementations are available on GitHub[1] to facilitate the dissemination and utilization of the research outcomes. Interested researchers, developers, and enthusiasts can access the repository to delve into the specifics of the autonomous drone system's modeling and algorithms. This open-access approach encourages collaboration, transparency, and further advancements in autonomous drone systems. The GitHub repository is a valuable resource for those interested in exploring, replicating, or building upon the insights and methodologies presented in this doctoral thesis.

[1] <https://github.com/vvannini/harpia>

BIBLIOGRAPHY

AHMAD, M. W.; AKRAM, M. U.; AHMAD, R.; HAMEED, K.; HASSAN, A. Intelligent framework for automated failure prediction, detection, and classification of mission critical autonomous flights. **ISA Transactions**, v. 129, p. 355–371, 2022. ISSN 0019-0578. Available: <https://www.sciencedirect.com/science/article/pii/S0019057822000209>. Citations on pages 41 and 45.

ALOS, A. M.; DAHROUJ, Z.; DAKKAK, M. A novel technique to assess uav behavior using pca-based anomaly detection algorithm. **International Journal of Mechanical Engineering and Robotics Research**, IJMERR, v. 9, n. 5, May 2020. Citations on pages 41 and 45.

ALSALAM, B. H. Y.; MORTON, K.; CAMPBELL, D.; GONZALEZ, F. Autonomous uav with vision based on-board decision making for remote sensing and precision agriculture. In: **2017 IEEE Aerospace Conference**. [S.l.: s.n.], 2017. p. 1–12. Citations on pages 37, 38, 44, and 45.

ARANTES, J. d. S. **Sistema autônomo para supervisão de missão e segurança de voo em VANTs**. Phd Thesis (PhD Thesis) — Universidade de São Paulo, 2019. Citations on pages 39 and 40.

ARANTES, M.; TOLEDO, C.; WILIAMS, B.; ONO, M. Collision-free encoding for chance-constrained nonconvex path planning. **IEEE Transactions on Robotics**, v. 35, n. 2, p. 433–448, 2019. Citations on pages 34, 43, 45, 47, and 71.

ARANTES, M. da S. **Hybrid qualitative state plan problem e o planejamento de missão com VANTs**. Phd Thesis (PhD Thesis) — Universidade de São Paulo, 2017. Citation on page 51.

ARANTES, M. S.; ARANTES, J. S.; TOLEDO, C. F. M.; WILLIAMS, B. C. A hybrid multi-population genetic algorithm for uav path planning. In: **Genetic and Evolutionary Computation Conference**. [S.l.: s.n.], 2016. Citations on pages 34, 43, and 45.

ARFAOUI, A. Unmanned aerial vehicle: Review of onboard sensors, application fields, open problems and research issues. **International Journal of Image Processing (IJIP)**, v. 11, n. 1, p. 12, 2017. Citation on page 27.

BADRA, A. A.; AïELLO, O.; CHAUDEMAR, J.-C. Applying a model-based systems engineering approach to model an unmanned aerial vehicle mission. In: **2023 IEEE International Systems Conference (SysCon)**. [S.l.: s.n.], 2023. p. 1–7. Citation on page 35.

BALDINI, A.; D'ALLEVA, L.; FELICETTI, R.; FERRACUTI, F.; FREDDI, A.; MONTERIù, A. Uav-fd: a dataset for actuator fault detection in multirotor drones *. In: **2023 International Conference on Unmanned Aircraft Systems (ICUAS)**. [S.l.: s.n.], 2023. p. 998–1004. Citations on pages 43 and 45.

BASKAYA, E.; BRONZ, M.; DELAHAYE, D. Fault detection & diagnosis for small uavs via machine learning. In: **2017 IEEE/AIAA 36th Digital Avionics Systems Conference (DASC)**. [S.l.: s.n.], 2017. p. 1–6. Citations on pages 42 and 45.

BAVLE, H.; MANTHE, S.; PUENTE, P. de la; RODRIGUEZ-RAMOS, A.; SAMPEDRO, C.; CAMPOY, P. Stereo visual odometry and semantics based localization of aerial robots in indoor environments. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 1018–1023. Citations on pages 27 and 34.

BIAN, J.; HUI, X.; ZHAO, X.; TAN, M. A novel monocular-based navigation approach for uav autonomous transmission-line inspection. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 1–7. Citation on page 33.

BLASCH, E. Autonomy in use for information fusion systems. In: IEEE. **NAECON 2018-IEEE National Aerospace and Electronics Conference**. [S.l.], 2018. p. 1–8. Citations on pages 56 and 63.

BODIN, F.; CHARRIER, T.; QUEFFELEC, A.; SCHWARZENTRUBER, F. Generating plans for cooperative connected uavs. In: **IJCAI**. [S.l.: s.n.], 2018. p. 5811–5813. Citation on page 29.

BOUBETA-PUIG, J.; MOGUEL, E.; SÁNCHEZ-FIGUEROA, F.; HERNÁNDEZ, J.; PRECIADO, J. C. An autonomous uav architecture for remote sensing and intelligent decision-making. **IEEE Internet Computing**, IEEE, v. 22, n. 3, p. 6–15, 2018. Citation on page 38.

CAMPO, L. V.; LEDEZMA, A.; CORRALES, J. C. Optimization of coverage mission for lightweight unmanned aerial vehicles applied in crop data acquisition. **Expert Systems with Applications**, Elsevier, v. 149, p. 113227, 2020. Citations on pages 35, 44, and 45.

CARRIO, A.; VEMPRALA, S.; RIPOLL, A.; SARIPALLI, S.; CAMPOY, P. Drone detection using depth maps. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 1034–1037. Citation on page 34.

CASHMORE, M.; FOX, M.; LONG, D.; MAGAZZENI, D.; RIDDER, B.; CARRERA, A.; PALOMERAS, N.; HURTOS, N.; CARRERAS, M. Rosplan: Planning in the robot operating system. In: **Twenty-Fifth International Conference on Automated Planning and Scheduling**. [S.l.: s.n.], 2015. Citations on pages 29 and 62.

CHUNG, T. H.; CLEMENT, M. R.; DAY, M. A.; JONES, K. D.; DAVIS, D.; JONES, M. Live-fly, large-scale field experimentation for large numbers of fixed-wing uavs. In: IEEE. **2016 IEEE International Conference on Robotics and Automation (ICRA)**. [S.l.], 2016. p. 1255–1262. Citations on pages 34 and 57.

COHN, P.; GREEN, A.; LANGSTAFF, M.; ROLLER, M. Commercial drones are here: The future of unmanned aerial systems. **McKinsey & Company**, 2017. Citation on page 66.

DENG, D.; PALLI, P.; SHU, F.; SHIMADA, K.; PANG, T. Heterogeneous vehicles routing for water canal damage assessment. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 2375–2382. Citations on pages 27, 32, and 57.

FALOMIR, E.; CHAUMETTE, S.; GUERRINI, G. A mobility model based on improved artificial potential fields for swarms of uavs. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 8499–8504. Citation on page 34.

FIGUEIRA, N.; FREIRE, I.; TRINDADE, O.; OES, E. S. Mission-oriented sensor arrays and uavs - a case study on environmental monitoring. **ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences**, p. 305–312, 2015.

Available: <http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-1-W4/305/2015/>. Citations on pages 36 and 37.

FIGUEIRA, N.; TRINDADE, O.; MATTEI, A. L. P.; NERIS, L. Mission oriented sensor arrays – an approach towards uas usability improvement in practical applications. In: **5th European Conference for Aeronautics and Space Sciences (EUCASS)**. [S.l.: s.n.], 2013. Citation on page 36.

FIGUEIRA, N. M. **Arranjos de sensores orientados à missão para a geração automática de mapas temáticos em VANTs**. Phd Thesis (Tese de Doutorado) — Universidade de Sao Paulo (USP), mar 2016. Sao Carlos, SP. Citation on page 39.

FOX, M.; LONG, D. Pddl2. 1: An extension to pddl for expressing temporal planning domains. **Journal of artificial intelligence research**, v. 20, p. 61–124, 2003. Citations on pages 58 and 69.

GHASEMLOU, S.; OKANE, J. M.; SHELL, D. A. Delineating boundaries of feasibility between robot designs. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 422–429. Citation on page 34.

GONZALEZ, L. F.; MONTES, G. A.; PUIG, E.; JOHNSON, S.; MENGERSEN, K.; GASTON, K. J. Unmanned aerial vehicles (uavs) and artificial intelligence revolutionizing wildlife monitoring and conservation. **Sensors**, Multidisciplinary Digital Publishing Institute, v. 16, n. 1, p. 97, 2016. Citation on page 29.

GUNETTI, P.; DODD, T.; THOMPSON, H. A software architecture for autonomous uav mission management and control. **AIAA Infotech@Aerospace 2010**, 2010. Citation on page 35.

HAKSAR, R. N.; SCHWAGER, M. Distributed deep reinforcement learning for fighting forest fires with a network of aerial robots. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 1067–1074. Citation on page 32.

HAMAZA, S.; GEORGILAS, I.; RICHARDSON, T. Towards an adaptive-compliance aerial manipulator for contact-based interaction. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 1–9. Citation on page 33.

HIRECHE, C.; DEZAN, C.; DIGUET, J.-P.; MEJIAS, L. Bfm: a scalable and resource-aware method for adaptive mission planning of uavs. In: IEEE. **2018 IEEE International Conference on Robotics and Automation (ICRA)**. [S.l.], 2018. p. 6702–6707. Citations on pages 40, 43, and 45.

JIANG, H.; ELBAUM, S.; DETWEILER, C. Reducing failure rates of robotic systems though inferred invariants monitoring. In: IEEE. **2013 IEEE/RSJ International Conference on Intelligent Robots and Systems**. [S.l.], 2013. p. 1899–1906. Citation on page 57.

JUNG, S.; SONG, S.; YOUN, P.; MYUNG, H. Multi-layer coverage path planner for autonomous structural inspection of high-rise structures. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 1–9. Citation on page 32.

KENMOGNE, I.-F.; DREVELLE, V.; MARCHAND, E. Interval-based cooperative uavs pose domain characterization from images and ranges. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 6349–6356. Citation on page 34.

KHAN, S.; LIEW, C. F.; YAIRI, T.; MCWILLIAM, R. Unsupervised anomaly detection in unmanned aerial vehicles. **Applied Soft Computing**, v. 83, p. 105650, 2019. ISSN 1568-4946. Available: <https://www.sciencedirect.com/science/article/pii/S1568494619304302>. Citations on pages 42, 44, and 45.

KIM, J.; KIM, S.; JU, C.; SON, H. I. Unmanned aerial vehicles in agriculture: A review of perspective of platform, control, and applications. **IEEE Access**, IEEE, v. 7, p. 105100–105115, 2019. Citation on page 31.

KOURIS, A.; BOUGANIS, C.-S. Learning to fly by myself: A self-supervised cnn-based approach for autonomous navigation. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 1–9. Citation on page 33.

KOŁODZIEJCZAK, M.; PUCHALSKI, R.; BONDYRA, A.; SLADIC, S.; GIERNACKI, W. Toward lightweight acoustic fault detection and identification of uav rotors. In: **2023 International Conference on Unmanned Aircraft Systems (ICUAS)**. [S.l.: s.n.], 2023. p. 990–997. Citation on page 43.

KUHNER, D.; ALDINGER, J.; BURGET, F.; GÖBELBECKER, M.; BURGARD, W.; NEBEL, B. Closed-loop robot task planning based on referring expressions. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 876–881. Citation on page 62.

LI, M.; LI, G.; ZHONG, M. A data driven fault detection and isolation scheme for uav flight control system. In: **2016 35th Chinese Control Conference (CCC)**. [S.l.: s.n.], 2016. p. 6778–6783. Citations on pages 42 and 45.

LINDQVIST, B.; KARLSSON, S.; KOVAL, A.; TEVETZIDIS, I.; HALUŠKA, J.; KANELLAKIS, C.; AGHA-MOHAMMADI, A.-a.; NIKOLAKOPOULOS, G. Multimodality robotic systems: Integrated combined legged-aerial mobility for subterranean search-and-rescue. **Robotics and Autonomous Systems**, Elsevier, v. 154, p. 104134, 2022. Citations on pages 35, 44, and 45.

LUIS, C. E.; VUKOSAVLJEV, M.; SCHOELLIG, A. P. Online trajectory generation with distributed model predictive control for multi-robot motion planning. **IEEE Robotics and Automation Letters**, IEEE, v. 5, n. 2, p. 604–611, 2020. Citations on pages 34 and 43.

MA, J.; LI, Y.; LIU, H.; WU, Y.; ZHANG, L. Towards improved accuracy of uav-based wheat ears counting: A transfer learning method of the ground-based fully convolutional network. **Expert Systems with Applications**, Elsevier, v. 191, p. 116226, 2022. Citations on pages 31, 44, and 45.

MATTEI, A. L. P. **Consciência situacional em voo de sistemas aéreos não tripulados**. Phd Thesis (PhD Thesis) — Universidade de São Paulo, 2015. Citation on page 39.

MATTEI, A. L. P.; TOLEDO, C. F. M.; ARANTES, J. d. S.; JUNIOR, O. T. Unmanned aerial vehicles flight safety improvement using in-flight awareness. **Intelligent Information Management**, v. 13, n. 2, p. 97–123, 2021. Citations on pages 39, 40, 43, and 45.

MENG, X.; HE, Y.; LI, Q.; GU, F.; YANG, L.; YAN, T.; HAN, J. Contact force control of an aerial manipulator in pressing an emergency switch process. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 2107–2113. Citation on page 32.

MOHAIMENIANPOUR, S.; VAUGHAN, R. Hands and faces, fast: Mono-camera user detection robust enough to directly control a uav in flight. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 5224–5231. Citations on pages 33 and 57.

NOUACER, R.; HUSSEIN, M.; ESPINOZA, H.; OUHAMMOU, Y.; LADEIRA, M.; CASTIÑEIRA, R. Towards a framework of key technologies for drones. **Microprocessors and Microsystems**, Elsevier, v. 77, p. 103142, 2020. Citation on page 66.

ONO, M.; WILLIAMS, B. C.; BLACKMORE, L. Probabilistic planning for continuous dynamic systems under bounded risk. **Journal of Artificial Intelligence Research**, v. 46, p. 511–577, 2013. Citation on page 52.

POBKRUT, T.; EAMSA-ARD, T.; KERDCHAROEN, T. Sensor drone for aerial odor mapping for agriculture and security services. In: IEEE. **2016 13th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology (ECTI-CON)**. [S.l.], 2016. p. 1–5. Citation on page 28.

POIESI, F.; CAVALLARO, A. A distributed vision-based consensus model for aerial-robotic teams. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 169–176. Citation on page 39.

PRODAN, I.; OLARU, S.; BENCATEL, R.; SOUSA, J. B. de; STOICA, C.; NICULESCU, S.-I. Receding horizon flight control for trajectory tracking of autonomous aerial vehicles. **Control Engineering Practice**, v. 21, n. 10, p. 1334 – 1349, 2013. ISSN 0967-0661. Available: <http://www.sciencedirect.com/science/article/pii/S0967066113001020>. Citations on pages 36, 44, and 45.

QUIGLEY, M.; CONLEY, K.; GERKEY, B.; FAUST, J.; FOOTE, T.; LEIBS, J.; WHEELER, R.; NG, A. Y. Ros: an open-source robot operating system. In: KOBE, JAPAN. **ICRA workshop on open source software**. [S.l.], 2009. v. 3, n. 3.2, p. 5. Citation on page 57.

RADOGLOU-GRAMMATIKIS, P.; SARIGIANNIDIS, P.; LAGKAS, T.; MOSCHOLIOS, I. A compilation of uav applications for precision agriculture. **Computer Networks**, Elsevier, v. 172, p. 107148, 2020. Citation on page 31.

RAMASAMY, S.; SABATINI, R.; GARDI, A.; LIU, J. LIDAR obstacle warning and avoidance system for unmanned aerial vehicle sense-and-avoid. **Aerospace Science and Technology**, v. 55, p. 344 – 358, 2016. ISSN 1270-9638. Available: <http://www.sciencedirect.com/science/article/pii/S1270963816301900>. Citations on pages 36, 44, and 45.

RAMIREZ-ATENCIA, C.; RODRÍGUEZ-FERNÁNDEZ, V.; GONZALEZ-PARDO, A.; CAMACHO, D. New artificial intelligence approaches for future uav ground control stations. In: IEEE. **Evolutionary Computation (CEC), 2017 IEEE Congress on**. [S.l.], 2017. p. 2775–2782. Citation on page 29.

RASTGOO, M.; DEMONCEAUX, C.; SEULIN, R.; MOREL, O. Attitude estimation from polarimetric cameras. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 8397–8403. Citation on page 27.

ROELOFSEN, S.; GILLET, D.; MARTINOLI, A. Reciprocal collision avoidance for quadrotors using on-board visual detection. In: IEEE. **2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2015. p. 4810–4817. Citation on page 34.

RUSSELL, S. J.; NORVIG, P. **Artificial intelligence: a modern approach**. [S.l.]: Malaysia; Pearson Education Limited,, 2016. Citations on pages 51, 55, 63, and 72.

SADHU, V.; ZONOUZ, S.; POMPILI, D. On-board deep-learning-based unmanned aerial vehicle fault cause detection and identification. In: IEEE. **2020 IEEE International Conference on Robotics and Automation (ICRA)**. Paris, France, 2020. Citations on pages 41, 44, and 45.

SAKAI, A.; INGRAM, D.; DINIUS, J.; CHAWLA, K.; RAFFIN, A.; PAQUES, A. Python-robotics: a python code collection of robotics algorithms. **arXiv preprint arXiv:1808.10703**, 2018. Citations on pages 50 and 69.

SAMPEDRO, C.; BAVLE, H.; RODRIGUEZ-RAMOS, A.; PUENTE, P. de la; CAMPOY, P. Laser-based reactive navigation for multirotor aerial robots using deep reinforcement learning. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 1024–1031. Citations on pages 39 and 57.

SANELLI, V.; CASHMORE, M.; MAGAZZENI, D.; IOCCHI, L. Short-term human-robot interaction through conditional planning and execution. In: **Twenty-Seventh International Conference on Automated Planning and Scheduling**. [S.l.: s.n.], 2017. Citations on pages 64 and 66.

SAPKOTA, K. R.; ROELOFSEN, S.; ROZANTSEV, A.; LEPETIT, V.; GILLET, D.; FUA, P.; MARTINOLI, A. Vision-based unmanned aerial vehicle detection and tracking for sense and avoid systems. In: IEEE. **2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2016. p. 1556–1561. Citations on pages 34 and 57.

SARTORI, D.; ZOU, D.; PEI, L.; YU, W. A revisited approach to lateral acceleration modeling for quadrotor uavs state estimation. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 5711–5718. Citations on pages 33 and 57.

Shakhatreh, H.; Sawalmeh, A. H.; Al-Fuqaha, A.; Dou, Z.; Almaita, E.; Khalil, I.; Othman, N. S.; Khreishah, A.; Guizani, M. Unmanned aerial vehicles (uavs): A survey on civil applications and key research challenges. **IEEE Access**, v. 7, p. 48572–48634, 2019. Citations on pages 27, 31, and 32.

SONG, B. D.; PARK, H.; PARK, K. Toward flexible and persistent uav service: Multi-period and multi-objective system design with task assignment for disaster management. **Expert Systems with Applications**, Elsevier, v. 206, p. 117855, 2022. Citations on pages 35, 44, and 45.

SOUZA, G. M.; TOLEDO, C. F. M. Genetic algorithm applied in uav's path planning. In: IEEE. **2020 IEEE Congress on Evolutionary Computation (CEC)**. [S.l.], 2020. p. 1–8. Citations on pages 35, 43, 45, 50, and 69.

STEENSTRA, L. Pddl-based task planning of survey missions for autonomous underwater vehicles: A generic planning system, taking into account location uncertainty and environmental properties. 2019. Citation on page 62.

STRUPKA, G.; LEVCHENKOV, A.; GOROBETZ, M. Automated situation analysis as next level of unmanned aerial vehicle artificial intelligence. In: SPRINGER. **International Conference on Applied Human Factors and Ergonomics**. [S.l.], 2017. p. 25–37. Citation on page 29.

SUN, F.; WANG, X.; ZHANG, R. Task scheduling system for uav operations in agricultural plant protection environment. **JOURNAL OF AMBIENT INTELLIGENCE AND HUMANIZED COMPUTING**, Springer, 2020. Citations on pages 35, 44, and 45.

SVACHA, J.; MOHTA, K.; WATTERSON, M.; LOIANNO, G.; KUMAR, V. Inertial velocity and attitude estimation for quadrotors. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 1–9. Citation on page 33.

TABOR, S.; GUILLIARD, I.; KOLOBOV, A. Ardusoar: an open-source thermalling controller for resource-constrained autopilots. In: IEEE. **2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2018. p. 6255–6262. Citation on page 27.

USACH, H.; VILA, J. A.; TORENS, C.; ADOLF, F. Architectural design of a safe mission manager for unmanned aircraft systems. **Journal of Systems Architecture**, Elsevier, v. 90, p. 94–108, 2018. Citation on page 38.

VANEGAS, F.; CAMPBELL, D.; EICH, M.; GONZALEZ, F. Uav based target finding and tracking in gps-denied and cluttered environments. In: IEEE. **2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2016. p. 2307–2313. Citation on page 57.

VEMULAPALLI, K. **Investigating the use of digital twins in networked commercial UAVs**. Phd Thesis (PhD Thesis) — Massachusetts Institute of Technology, 2019. Citation on page 28.

WANG, B.; PENG, X.; JIANG, M.; LIU, D. Real-time fault detection for uav based on model acceleration engine. **IEEE Transactions on Instrumentation and Measurement**, IEEE, v. 69, n. 12, p. 9505, 2020. Citations on pages 41 and 44.

WANG, D.; WILLIAMS, B. tburton: A divide and conquer temporal planner. In: **Twenty-Ninth AAAI Conference on Artificial Intelligence**. [S.l.: s.n.], 2015. Citation on page 45.

WOPEREIS, H. W.; FUMAGALLI, M.; STRAMIGIOLI, S.; CARLONI, R. Bilateral human-robot control for semi-autonomous uav navigation. In: IEEE. **2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [S.l.], 2015. p. 5234–5240. Citations on pages 36, 37, and 57.

WRIGHT, D. Drones: Regulatory challenges to an incipient industry. **Computer Law & Security Review**, v. 30, n. 3, p. 226 – 229, 2014. ISSN 0267-3649. Citation on page 29.

XUE, X.; LAN, Y.; SUN, Z.; CHANG, C.; HOFFMANN, W. C. Develop an unmanned aerial vehicle based automatic aerial spraying system. **Computers and Electronics in Agriculture**, v. 128, p. 58 – 66, 2016. ISSN 0168-1699. Available: <http://www.sciencedirect.com/science/article/pii/S0168169916305282>. Citations on pages 36, 44, and 45.

YAN, F.; LI, S.; CHEN, Q. Hardware and software design of highly reliable integrated navigation system for long-endurance uav. In: IEEE. **2018 Chinese Control And Decision Conference (CCDC)**. [S.l.], 2018. Citation on page 38.

ZHANG, C.; KOVACS, J. M. The application of small unmanned aerial systems for precision agriculture: a review. **Precision agriculture**, Springer, v. 13, n. 6, p. 693–712, 2012. Citation on page 27.

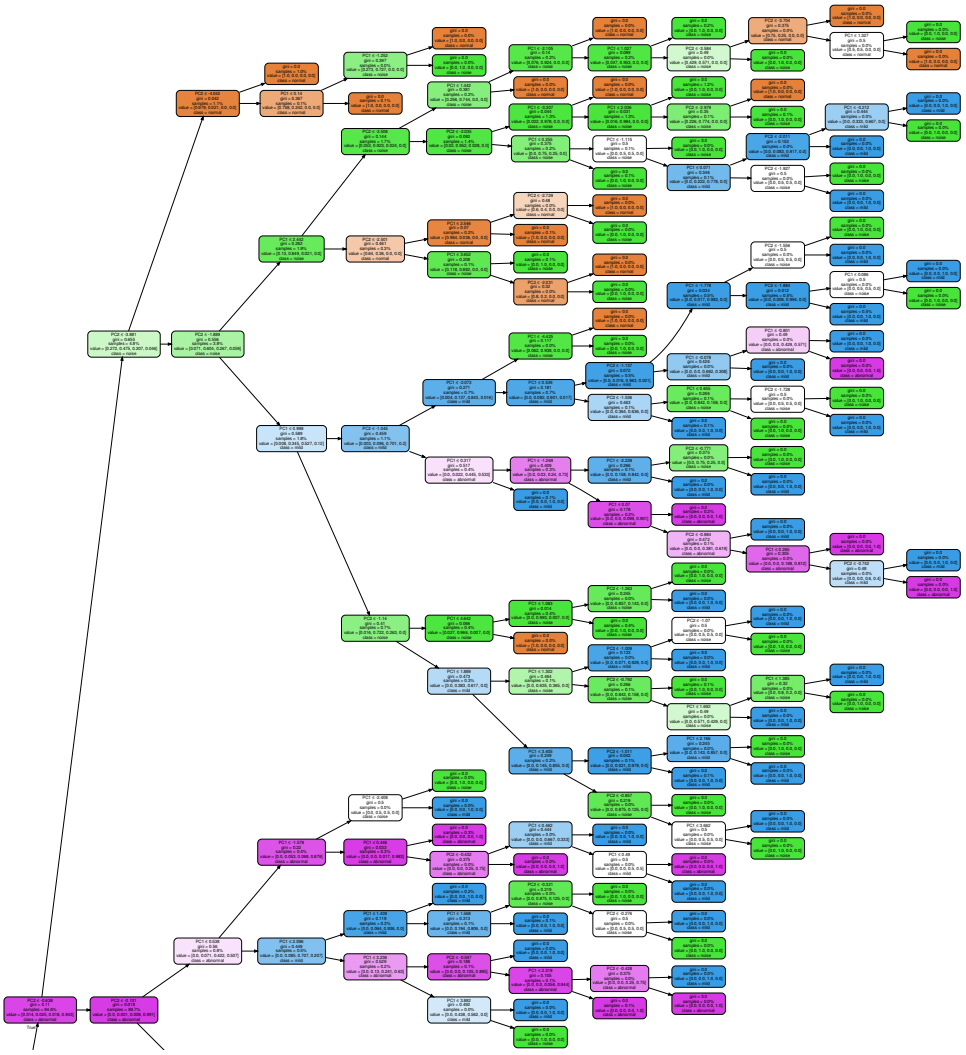
ZHANG, S.; LI, Y.; DONG, Q. Autonomous navigation of uav in multi-obstacle environments based on a deep reinforcement learning approach. **Applied Soft Computing**, Elsevier, v. 115, p. 108194, 2022. Citations on pages 44 and 45.

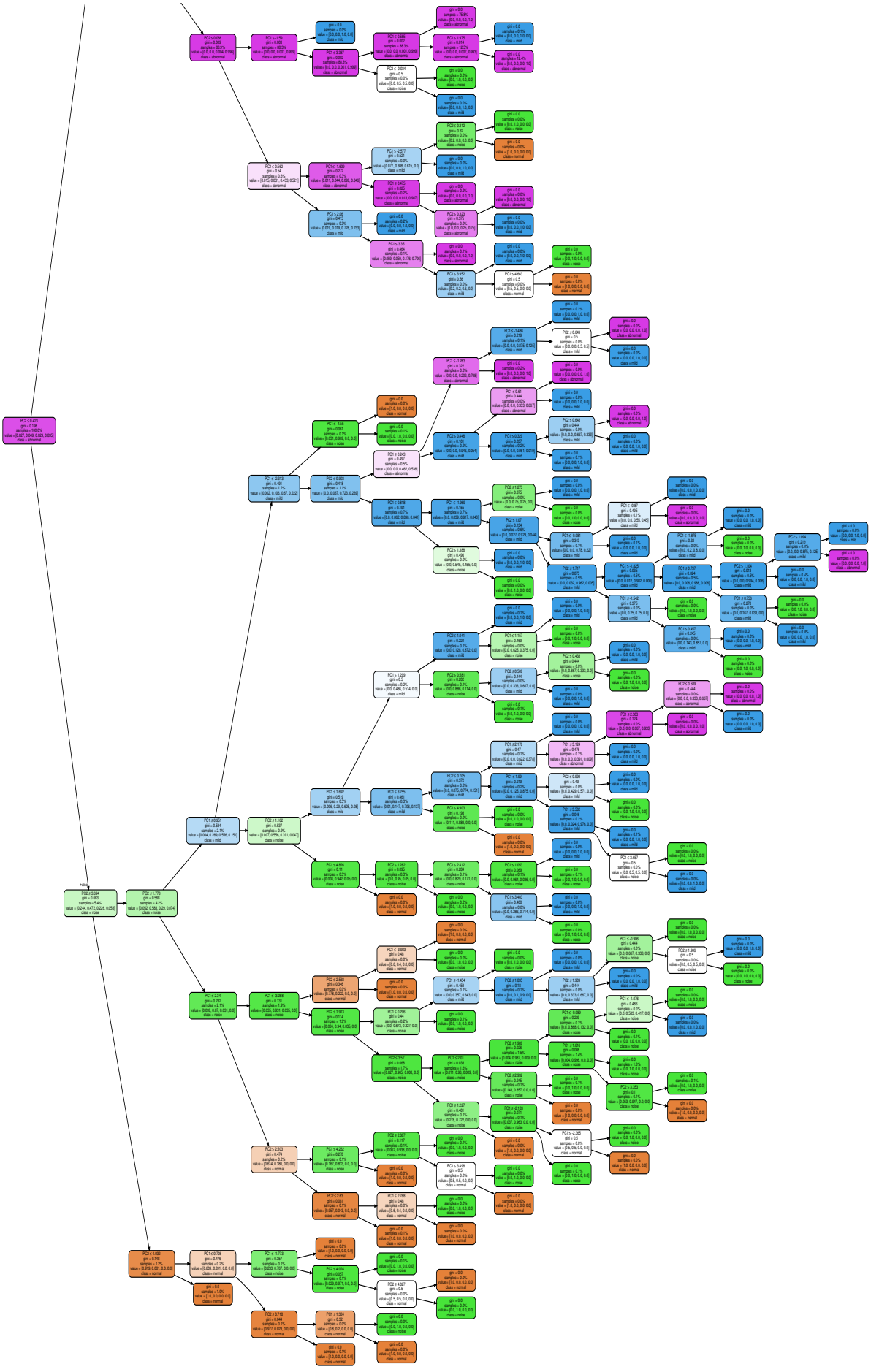
ZUBI, G.; ADHIKARI, R. S.; SáNCHEZ, N. E.; ACUñA-BRAVO, W. Lithium-ion battery-packs for solar home systems: Layout, cost and implementation perspectives. **Journal of Energy Storage**, v. 32, p. 101985, 2020. ISSN 2352-152X. Available: <https://www.sciencedirect.com/science/article/pii/S2352152X2031820X>. Citation on page 96.

APPENDIX

A

GENERATED TREE





APPENDIX

B

PROBLEM MODELING IN PDDL

Source code 6 – Domain Code

```
1: (define (domain harpia)
2:
3:     (:requirements :typing :strips
   :disjunctive-preconditions :equality )
4:
5:     (:types
6:         region - object
7:         base - region)
8:
9:
10:
11:    (:functions
12:
13:
14:        ;; Variavel q controla bateria em porcentagem
15:        (battery-amount)
16:        ;; quantidade de insumo
17:        (input-amount)
18:        ;;velocidade de carregar a bateria em porcentagem por
   segundos
19:        (recharge-rate-battery)
20:        ;;velocidade de descarregar a bateria
21:        (discharge-rate-battery)
22:        ;;capacidade maxima bateria
23:        (battery-capacity)
```

```
24:         ;;capacidade maxima de insumo
25:         (input-capacity)
26:         ;;velocidade de reabastecer o insumo
27:         (recharge-rate-input)
28:         ;;distancia entre regioes em metros
29:         (distance ?from-region - region ?to-region - region)
30:         ;;velocidade em m/s
31:         (velocity)
32:         (picture-path-len ?region - region)
33:         (pulverize-path-len ?region - region)
34:         (total-goals)
35:         (goals-achived)
36:         (mission-length)
37:
38:     )
39:
40:     (:predicates
41:
42:         (been-at ?region - region)
43:         ;;se esta carregando um insumo
44:         (carry)
45:         ;;esta em uma regiao
46:         (at ?region - region)
47:         ;; se pode pulverizar
48:         (can-spray)
49:         ;;se pode carregar/descarregar
50:         (can-recharge)
51:         ;se já tirou a foto
52:         (taken-image ?region - region)
53:         ;se pulverizou
54:         (pulverized ?region - region)
55:         ; (canGo)
56:         (can-take-pic)
57:         (its-not-base ?region - region)
58:         (pulverize-goal ?region - region)
59:         (picture-goal ?region - region)
60:         (hw-ready ?from - region ?to - region)
61:
62:         ; (can-go-to-base)
```

```
63:         ; (has-pulverize-goal)
64:         ; (has-picture-goal)
65:         ; (at-move)
66:
67:     )
68:
69:
70:     (:action go_to
71:         :parameters (
72:              ?from-region - region
73:              ?to-region - region)
74:         :precondition (and
75:             (at ?from-region)
76:             (> (battery-amount) (+ (* (/ (distance ?from-region
       ?to-region) (velocity)) (discharge-rate-battery)) 15))
77:         )
78:         :effect (and
79:                 (not (at ?from-region))
80:                 (been-at ?to-region)
81:                 (at ?to-region)
82:                 (decrease (battery-amount )
83:                       (*
84:                           (/
85:                               (distance ?from-region ?to-region
       )
86:                               (velocity)
87:                           )
88:                           (discharge-rate-battery)
89:                       )
90:
91:                 )
92:                 (increase (mission-length) (distance ?
       from-region ?to-region))
93:                 )
94:     )
95:
96:     (:action take_image
97:         :parameters (
98:             ?region - region
```

```
99:         )
100:        :precondition(and
101:            (at ?region)
102:            (picture-goal ?region)
103:            (> (battery-amount)
104:                (*
105:                    (/
106:                        1000
107:                        (velocity)
108:                    )
109:                    (discharge-rate-battery)
110:                )
111:            )
112:        )
113:        :effect(and
114:            (taken-image ?region)
115:            (increase (mission-length) 1000)
116:            (decrease (battery-amount)
117:                (*
118:                    (/
119:                        1000
120:                        (velocity)
121:                    )
122:                    (discharge-rate-battery)
123:                )
124:            )
125:        )
126:    )
127:    (:action pulverize_region
128:        :parameters (
129:            ?region - region)
130:        :precondition(and
131:            (at ?region)
132:            (pulverize-goal ?region)
133:            (> (input-amount) 0)
134:            (> (battery-amount)
135:                (*
136:                    (/
137:                        314
```

```
138:                         (velocity)
139:                     )
140:                     (discharge-rate-battery)
141:                 )
142:             )
143:         )
144:         :effect(and
145:             (pulverized ?region)
146:             (increase (mission-length) 314)
147:             (decrease (input-amount) 1)
148:             (decrease (battery-amount)
149:                 (*
150:                     (/
151:                         314
152:                         (velocity)
153:                     )
154:                     (discharge-rate-battery)
155:                 )
156:             )
157:         )
158:     )
159:     (:action recharge_battery
160:         :parameters (?base - base)
161:         :precondition (and
162:             (at ?base)
163:             ;(< (battery-amount) 60)
164:         )
165:         :effect
166:         (and
167:             (assign (battery-amount) (battery-capacity))
168:         )
169:     )
170:
171:     (:action recharge_input
172:         :parameters (?base - base)
173:         :precondition (and
174:             (at ?base)
175:             (< (input-amount) (/ (input-capacity) 2))
176:         )
```

```
177:         :effect
178:         (and
179:             (assign (input-amount) (input-capacity))
180:         )
181:     )
182: )
```

Source code 7 – Example of Problem Code

```
 1: (define (problem task)
 2: (:domain harpia)
 3: (:objects
 4:     region_1 region_2 region_3 region_4 region_5 region_6
    region_7 region_8 region_9 region_10 region_11 region_12 -
    region
 5:     base_1 base_2 base_3 base_4 - base
 6: )
 7: (:init
 8:     (been-at region_2)
 9:     (been-at region_3)
10:
11:
12:     (at region_3)
13:
14:
15:
16:     (taken-image region_2)
17:
18:
19:
20:
21:
22:     (picture-goal region_2)
23:     (picture-goal region_3)
24:     (picture-goal region_1)
25:
26:
27:     (= (battery-amount) 79)
28:
29:     (= (input-amount) 0)
```

```
30:
31:
32:     (= (discharge-rate-battery) 0.042)
33:
34:     (= (battery-capacity) 100)
35:
36:     (= (input-capacity) 3)
37:
38:
39:     (= (distance region_2 region_3) 1352.25)
40:     (= (distance region_2 base_1) 1835.59)
41:     (= (distance region_2 base_2) 1870.26)
42:     (= (distance region_2 base_3) 2292.66)
43:     (= (distance region_2 base_4) 1043.42)
44:     (= (distance region_3 region_2) 1352.25)
45:     (= (distance region_3 base_1) 2775.42)
46:     (= (distance region_3 base_2) 1123.67)
47:     (= (distance region_3 base_3) 3241.38)
48:     (= (distance region_3 base_4) 821.239)
49:     (= (distance base_1 region_2) 1835.59)
50:     (= (distance base_1 region_3) 2775.42)
51:     (= (distance base_1 base_2) 2474.12)
52:     (= (distance base_1 base_3) 3513.11)
53:     (= (distance base_1 base_4) 2828.67)
54:     (= (distance base_2 region_2) 1870.26)
55:     (= (distance base_2 region_3) 1123.67)
56:     (= (distance base_2 base_1) 2474.12)
57:     (= (distance base_2 base_3) 4100.55)
58:     (= (distance base_2 base_4) 1872.06)
59:     (= (distance base_3 region_2) 2292.66)
60:     (= (distance base_3 region_3) 3241.38)
61:     (= (distance base_3 base_1) 3513.11)
62:     (= (distance base_3 base_2) 4100.55)
63:     (= (distance base_3 base_4) 2454)
64:     (= (distance base_4 region_2) 1043.42)
65:     (= (distance base_4 region_3) 821.239)
66:     (= (distance base_4 base_1) 2828.67)
67:     (= (distance base_4 base_2) 1872.06)
68:     (= (distance base_4 base_3) 2454)
```

```
69:     (= (distance region_1 region_2) 2659.31)
70:     (= (distance region_1 region_3) 2474.48)
71:     (= (distance region_1 base_1) 2137.48)
72:     (= (distance region_1 base_2) 1421.39)
73:     (= (distance region_1 base_3) 4930.21)
74:     (= (distance region_1 base_4) 3092.56)
75:     (= (distance region_2 region_1) 2659.31)
76:     (= (distance region_3 region_1) 2474.48)
77:     (= (distance base_1 region_1) 2137.48)
78:     (= (distance base_2 region_1) 1421.39)
79:     (= (distance base_3 region_1) 4930.21)
80:     (= (distance base_4 region_1) 3092.56)
81:
82:     (= (velocity) 3.5)
83:
84:     (= (picture-path-len region_1) 1000)
85:
86:
87:
88:
89:     (= (mission-length) 0)
90:
91: )
92: (:goal (and
93:     (taken-image region_3)
94:     (at base_1)
95:     (taken-image region_1)
96: ))
97: (:metric minimize (mission-length))
98: )
```


ICMC
USP
SÃO CARLOS