



AN AERIAL ROBOT COGNITIVE NETWORK ARCHITECTURE FOR SWARM APPLICATIONS DEVELOPMENT: ARCOG-NET

Gabryel Silva Ramos

Thesis submitted to the Postgraduate Program in Instrumentation and Applied Optics of the Federal Center for Technological Education of Rio de Janeiro, CEFET/RJ, as partial fulfillment of the requirements for the degree of doctor.

Advisor: Milena Faria Pinto
Co-advisor: Diego Barreto Haddad

Rio de Janeiro,

November, 2025

AN AERIAL ROBOT COGNITIVE NETWORK ARCHITECTURE FOR SWARM APPLICATIONS DEVELOPMENT: ARCOG-NET

Thesis submitted to the Postgraduate Program in Instrumentation and Applied Optics of the Federal Center for Technological Education of Rio de Janeiro, CEFET/RJ, as partial fulfillment of the requirements for the degree of doctor.

Gabryel Silva Ramos

Examining jury:

President, Milena Faria Pinto, D.Sc. (CEFET/RJ) (Advisor)

Diego Barreto Haddad, D.Sc. (CEFET/RJ) (Coadvisor)

Amaro Azevedo de Lima, D.Sc. (CEFET/RJ)

Accacio Ferreira dos Santos Neto, D.Sc. (CEFET/MG)

Mario Antonio Ribeiro Dantas, D.Sc. (INESC/UFJF)

Ricardo Queiroz de Araujo Fernandes, D.Sc. (GSI)

Rio de Janeiro,

November, 2025

DEDICATION

Aos meus pais, que com muito sacrifício me permitiram sonhar e chegar onde estou.

Ao meu filho, Daniel. Quando optei pela engenharia, o fiz sonhando em criar grandes coisas. Agora, porém, sei que nada que eu venha a fazer irá te superar. Você é, e sempre será, minha maior criação.

ACKNOWLEDGMENTS

O presente trabalho contou com o apoio da Coordenação de Aperfeicionamento de Pessoal de Nível Superior - Brasil (CAPES) - Para participação e publicação de artigo no Simpósio Latino-Americano de Robótica 2023.

À minha orientadora Milena Faria Pinto por todas as contribuições com o projeto, além da disponibilidade e boa vontade ao longo do processo de pesquisa. Obrigado por compartilhar comigo não apenas seu conhecimento e experiência, mas sobretudo por me inspirar coragem para dar passos além do que eu imaginei que meus pés alcançassem.

Ao meu co-orientador Diego Barreto Haddad pelas instruções, apoio no estágio de docência e relevantes contribuições na produção científica decorrente deste doutorado.

Aos professores Amaro Azevedo de Lima e Felipe da Rocha Henriques pelas relevantes contribuições e orientações durante o período de doutorado.

Aos demais professores e servidores do CEFET-RJ e UFF por sua competência e dedicação. A qualidade da pesquisa e ensino das instituições públicas, das quais sinto orgulho em ter sido discente, é inigualável no Brasil. O ensino público é o alicerce da minha vida e certamente de outros milhares de brasileiros, e esse resultado eu só posso atribuir ao trabalho de vocês. Muito obrigado.

À banca examinadora, pelo convite aceito para avaliação deste trabalho.

"Passe adiante o que você aprendeu. Força, maestria, mas fraqueza, insensatez, fracasso também. Sim: fracasso, acima de tudo. O maior professor, o fracasso é."

Yoda

RESUMO

Uma arquitetura cognitiva de robôs aéreos em rede para desenvolvimento de aplicações de enxame: ARCog-NET

Esta tese propõe a ARCog-NET, uma arquitetura cognitiva distribuída para robôs aéreos em rede, voltada à coordenação autônoma de enxames em ambientes dinâmicos e incertos. Integrando percepção, planejamento, tomada de decisão e aprendizado adaptativo, a arquitetura permite cooperação entre drones com ou sem intervenção humana, operando sobre um modelo *edge-fog-cloud* que distribui funções conforme a capacidade computacional: os nós *edge* executam decisões locais em tempo real, os *fog* realizam coordenação regional e fusão de mapas, e o *cloud* executa análises complexas e armazenamento. A arquitetura foi validada em simulação e em experimentos práticos com seis drones DJI Ryze Tello em missão de reconstrução 3D no Laboratório de Controle e Automação (LACEA). O sistema apresentou erro médio absoluto (MAE) de 0,04 m em X e 0,07 m em Y, RMSE médio de 0,10 m, e erro absoluto de trajetória (ATE) médio de 0,13 m, valores compatíveis com sistemas visuais-inerciais de referência. A reconstrução do ambiente de 80 m² foi concluída em poucos minutos, com cobertura de 87% e tempo médio de convergência de 14 s, demonstrando capacidade de navegação cognitiva cooperativa e realinhamento dinâmico entre agentes. Em simulações com até 1000 UAVs (*Unmanned Aerial Vehicle*), o modelo manteve desempenho estável com aumento controlado de latência e consumo energético (70 J por formação, 9% de overhead de comunicação, e 88% de robustez a atrasos) evidenciando escalabilidade até o ponto de saturação da rede, quando novos agentes não trazem ganhos significativos de desempenho. A distribuição hierárquica da carga entre *edge*, *fog* e *cloud* mostrou-se essencial para mitigar congestionamento e otimizar o uso de recursos. Por fim, o trabalho disponibiliza publicamente todos os pacotes, dados de simulação e experimentos em repositório GitHub, promovendo transparência, reprodutibilidade e estímulo à pesquisa colaborativa na área de robótica cognitiva e SLAM (*Simultaneous Location and Mapping*) cooperativo.

Palavras-chave: Arquitetura Cognitiva; Robôs Aéreos; Autonomia de Robôs Aéreos; Processamento de Dados Distribuídos; Sistemas de Enxame; Tecnologias Edge-Fog-Cloud

ABSTRACT

An Aerial Robot Cognitive Network Architecture for Swarm Applications Development: ARCog-NET

This thesis proposes ARCog-NET, a distributed cognitive architecture for networked aerial robots designed to enable autonomous swarm coordination in dynamic and uncertain environments. By integrating perception, planning, decision-making, and adaptive learning, the architecture supports cooperative behavior among drones with or without human intervention, operating over an edge-fog-cloud computational model: edge nodes execute real-time local decisions, fog nodes perform regional coordination and map fusion, and the cloud layer handles complex analytics and long-term data management. The architecture was validated both in simulation and through real-world experiments using six DJI Ryze Tello drones performing a 3D reconstruction mission inside the Control and Automation Laboratory (LACEA). The system achieved an average absolute error (MAE) of 0.04 m in the X-axis and 0.07 m in the Y-axis, a mean RMSE of 0.10 m, and an average Absolute Trajectory Error (ATE) of 0.13 m, values comparable to established visual-inertial methods. The reconstruction of an 80 m^2 indoor environment was completed within minutes, achieving 87% coverage with a convergence time of 14 s, demonstrating cooperative cognitive navigation and dynamic inter-agent alignment. In large-scale simulations with up to 1000 UAVs (Unmanned Aerial Vehicle), ARCog-NET maintained stable performance with controlled increases in latency and energy consumption (approximately 70 J per formation, 9% communication overhead, and 88% robustness to delays) showing scalability up to the point of network saturation, beyond which additional agents no longer yield performance gains. The hierarchical distribution of computation across edge, fog, and cloud layers proved critical for mitigating congestion and optimizing resource utilization. Finally, all software packages, simulation datasets, and experimental recordings are publicly available in a GitHub repository, ensuring transparency, reproducibility, and promoting further research in cognitive robotics and cooperative SLAM (Simultaneous Location and Mapping).

Keywords: Aerial Robots; Autonomy of Aerial Robots; Cognitive Architecture; Distributed Data Processing; Edge-Fog-Cloud Technologies; Swarm Systems

LIST OF FIGURES

Figure 1 – Publications with the term "Cognitive Architecture." 32

Figure 2 – Publications with the term "Collaborative Robot System." 32

Figure 3 – Aerial Robot Cognitive Architecture In Network (ARCog-NET) basic representation. 41

Figure 4 – ARCog-NET message structure representation. 42

Figure 5 – ARCog-NET complete algorithm flowchart. 43

Figure 6 – Simplified edge node algorithm. 45

Figure 7 – Simplified fog node algorithm. 47

Figure 8 – Simplified cloud layer algorithm. 49

Figure 9 – Cognitive learning process in ARCog-NET. 51

Figure 10 – Task Allocation (TA) algorithm flowchart. 60

Figure 11 – Path Planning (PP) algorithm flowchart. 67

Figure 12 – Formation Control (FC) algorithm flowchart. 72

Figure 13 – World 3D point in monocular 2D image. 74

Figure 14 – Epipolar geometry from point matching. 76

Figure 15 – Triangulation to form the 3D point cloud from matching image points and camera parameters. 77

Figure 16 – ARCog-NET simplified dataflow representation. 82

Figure 17 – Standard hardware representation of ARCog-NET. 84

Figure 18 – ARCog-NET basic components. 85

Figure 19 – Active ROS nodes and topics for 2 Unmanned Aerial Vehicle (UAV) and a serve in simulation. 86

Figure 20 – Swarm of 10 Typhoon H480 in simulation. 86

Figure 21 – World created in Gazebo for simulating autonomous predictive maintenance on a vertical wind field mission. 87

Figure 22 – World created in Gazebo for simulating the image transmission system on an offshore mission. 88
Figure 23 – World created in Gazebo for simulating using the UAV swarm as WiFi routers for communication emergencies. 89
Figure 24 – World created in Gazebo for simulating using the UAV swarm in search and rescue indoor missions. 90
Figure 25 – Laboratory test setup with 6 DJI Ryze Tello connected to ARCog-NET. 92
Figure 26 – Autoencoder Structure. 99
Figure 27 – UAVs flying between wind turbines. 104
Figure 28 – Number of decisions processed on each ARCog-NET layer in the wind field simulation. 104
Figure 29 – Pareto Graph of the average number of tasks by type in the wind farm simulation 105
Figure 30 – Average ARCog-NET latency and power consumption by number of UAVs at each time in the wind farm simulation. 109
Figure 31 – Average ARCog-NET transmitted data and throughput in wind farm simulation. 110
Figure 32 – Compression example of a picture of corrosion found at a wind turbine shaft. 112
Figure 33 – PSNR, SSIM, and MS-SSIM variation for interlinked compression in wind field simulation. 112
Figure 34 – Average ARCog-NET packet loss and throughput in wind farm simulation. 113
Figure 35 – Group of 10 UAVs flying to search for oil spills in an offshore simulation. 114
Figure 36 – Number of decisions processed on each ARCog-NET layer in the offshore simulation. 117
Figure 37 – Pareto Graph of the average number of tasks by type in the offshore simulation. 117
Figure 38 – Average ARCog-NET latency by number of UAVs at each time in the offshore simulation. 118

Figure 39 – PSNR, SSIM, and MS-SSIM variation for interlinked compression in offshore simulation. 120
Figure 40 – Compression example of a picture of an oil spill found at the water surface. 120
Figure 41 – Average ARCog-NET transmitted data and throughput in offshore simulation. 121
Figure 42 – Average ARCog-NET packet loss and throughput in offshore simulation. 121
Figure 43 – 2D points occupied by each UAV in city simulation and cluster coverage. 123
Figure 44 – 3D points occupied by each UAV in city simulation and cluster coverage. 124
Figure 45 – Response of each UAV to X coordinate setpoint variation in city simulation. 124
Figure 46 – Response of each UAV to Y coordinate setpoint variation in city simulation. 125
Figure 47 – Number of decisions processed on each ARCog-NET layer in the city simulation. 126
Figure 48 – Pareto Graph of the average number of tasks by type in the city simulation. 127
Figure 49 – Average ARCog-NET latency by number of UAVs at each time in the city simulation. 130
Figure 50 – Average ARCog-NET transmitted data and throughput in city simulation. 130
Figure 51 – Average ARCog-NET packet loss and throughput in city simulation. 131
Figure 52 – Comparison between trajectories calculated by ARCog-NET and executed by each UAV for a simulation with 10 robots in an industrial plant environment. 132
Figure 53 – Response of each UAV to X coordinate setpoint variation in industrial plant simulation. 132
Figure 54 – Response of each UAV to Y coordinate setpoint variation in industrial plant simulation. 133

Figure 55 – Number of decisions processed on each ARCog-NET layer in all simulations of the industrial plant environment. 134
Figure 56 – Pareto graph of the average number of tasks by type in the industrial plant simulation. 135
Figure 57 – Average ARCog-NET latency by number of UAVs at each time in the industrial plant simulation. 138
Figure 58 – Average ARCog-NET transmitted data and throughput in industrial plant simulation. 139
Figure 59 – Average ARCog-NET packet loss and throughput in industrial plant simulation. 140
Figure 60 – PSNR, SSIM, and MS-SSIM variation for interlinked compression in industrial plant simulation. 141
Figure 61 – Ablation study of the simulated ARCog-NET cognitive architecture. 142
Figure 62 – Swarm picture before the take off. 143

LIST OF TABLES

Table 1 – Feature comparison among ARCog-NET and recent UAV swarm and distributed SLAM frameworks. 37
Table 2 – Comparison of Related Works with Key Aspects. 39
Table 3 – Complete Workflow Descriptions. 42
Table 4 – Power Consumption of Different Hardware Components. 90
Table 5 – Communication and Processing Power Specifications. 91
Table 6 – Task Allocation Comparison in Wind Turbine Inspection Scenario. 107
Table 7 – Path Planning Comparison in Wind Turbine Inspection Scenario. 107
Table 8 – Formation Control Comparison in Wind Turbine Inspection Scenario. 108
Table 9 – Path Planning Comparison in Offshore Scenario. 115
Table 10 – Formation Control Comparison in offshore scenario. 116
Table 11 – Task Allocation Comparison in offshore scenario. 118
Table 12 – Comparison of MAE and RMSE for X and Y coordinates for each UAV. 125
Table 13 – Task Allocation Comparison in city scenario. 127
Table 14 – Path Planning Comparison in city scenario. 128
Table 15 – Formation Control Comparison in city scenario. 128
Table 16 – MAE and RMSE for X and Y trajectories, and Time Shift for UAVs. 133
Table 17 – Path Planning Comparison in Industrial Plant Scenario. 136
Table 18 – Formation Control Comparison in Industrial Plant Scenario. 137
Table 19 – Task Allocation Comparison in Industrial Plant Scenario. 137

LIST OF ALGORITHMS

Algorithm 1 –	Task Allocation Process in ARCog-NET.	59
Algorithm 2 –	Particle Swarm Optimization for ARCog-NET navigation event trajectory path planning.	66
Algorithm 3 –	ARCog-NET PP Extension for Formation Control.	71

LIST OF ABBREVIATIONS

ARCOG	Aerial Robotics Cognitive Architecture
ARCOG-NET	Aerial Robot Cognitive Architecture In Network
CBR	Case-Based Reasoning
DL	Deep Learning
DM	Data Mining
EC	Evolutionary Computation
EFC	Edge-Fog-Cloud
ES	Expert System
FCU	Flight Controller Unit
GUI	Graphic User Interface
HITL	Human In The Loop
MAVLINK	Micro Air Vehicle Link
MAVROS	MAVLink ROS
ML	Machine Learning
MS-SSIM	Mean Squared Structural Similarity
NN	Neural Network
PSNR	Peak Signal To Noise Ratio
PSO	Particle Swarm Optimization
QOS	Quality Of Service
RL	Reinforcement Learning
ROS	Robotics Operating System
SSIM	Structural Similarity
UAV	Unmanned Aerial Vehicle

CONTENTS

Introduction **14**

1 Introduction **14**

1.1 Main Contributions 20

1.2 Organization 23

2 Background and Related Works **24**

2.1 Cognitive Aspects 24

2.1.1 Cognition in ARCog-NET 26

2.1.2 Decision-Making 27

2.2 Data Compression 29

2.3 Task Allocation 29

2.4 Path Planning and Formation Control 30

2.5 Related Works 31

2.5.1 Cognitive Architecture 32

2.5.2 Multi-Robot Collaboration 34

2.5.3 Simultaneous Location and Mapping (SLAM) in Distributed Robot Systems 36

2.6 The Proposed Architecture 38

3 Methodology **40**

3.1 Network architecture 40

3.1.1 Edge nodes design 43

3.1.2 The Fog layer 46

3.1.3 Cloud computing, control and supervision 48

3.2 Theoretical modeling, cognition and flight algorithms 50

3.2.1 Task Allocation 51

3.2.2 Path Planning 60

3.2.3 Formation Control 69
3.2.4 Monocular Vision SLAM 72
3.3 Simulation framework 80
3.3.1 Simulation architecture 81
3.3.2 Multi-UAV simulation with ROS and Gazebo 84
3.4 Laboratory SLAM test 90
3.5 Network evaluation 93
3.5.1 Latency 93
3.5.2 Throughput 94
3.5.3 Quality of service and packet loss 95
3.5.4 Energy consumption and efficiency 97
3.5.5 Data compression 98
3.5.6 Reliability and scalability 100

4 Results and Discussion 103
4.1 Environment 1: Vertical turbines wind energy field surveillance 103
4.2 Environment 2: Offshore environmental monitoring 114
4.3 Environment 3: Provide mobile phone coverage to a neighborhood 123
4.4 Environment 4: Industrial plant indoor search and rescue 131
4.5 Collaborative indoor mapping experiment results 142

5 Conclusions 145

6 Scientific production 148

References 149

Attachments 164

1- Introduction

Unmanned Aerial Vehicles (UAVs) are one of the most versatile robot platforms due to their wide range of applications (Parikh and Basu, 2021; Ramos et al., 2022b). To cite some uses, there is surveillance (Faria Pinto et al., 2017), naval operations assistance (Ramos et al., 2022a), search and rescue (Rodin et al., 2018), inspection (Berger et al., 2023), package delivering (Pugliese et al., 2020), environmental monitoring (Green et al., 2019), topography (Manfreda et al., 2019), among others. The UAV systems also have improved their capacities when combined in groups or swarms, where coordinated vehicles can share the same tasks or individually execute parts of a bigger mission, gaining efficiency or even being able to accomplish operations where a single robot wouldn't be enough (Tahir et al., 2019a). Coordinated UAVs can share tasks or execute parts of a larger mission individually, achieving greater efficiency or enabling operation beyond the capacity of a single robot (Tahir et al., 2019a; Alqudsi and Makaraci, 2025; Luthra, 2023).

Despite their potential, research and development of UAV swarms still face gaps. As reported by Abdelkader et al. (Abdelkader et al., 2021; Javed et al., 2024), most existing aerial swarm systems are validated through simulations, with experimental implementations limited to small groups in controlled indoor settings. Besides, outdoor experiments with autonomous UAV groups are scarce, revealing scalability challenges in real-world environments. These challenges are primarily attributed to uncertainties in individual robot localization, relative positioning within the swarm, and high communication demands required for coordinated and safe operation.

Recent systematic reviews emphasize the potential of UAV swarms for different kind of applications but highlight persistent obstacles such as scalability, energy management, security, communication reliability, and regulatory compliance (Javed et al., 2024). Future research must focus on integrating artificial intelligent (AI) models, advanced communication systems, sustainable energy solutions, and robust ethical and regulatory frameworks. Additionally, the development of generic swarm implementations that are adaptable to various application requirements remains a critical priority (Ramos et al., 2024; Alqudsi and Makaraci, 2025).

The integration of complex algorithms within aircraft systems has led also to the

development of highly specialized solutions tailored to specific applications (Du et al., 2014). Many of these systems excel in optimizing particular functions, only a limited few contribute to advancing cognitive capabilities, particularly in decision-making processes critical to aircraft operations. Cognitive architecture plays a pivotal role in organizing data and information flow within an agent, thereby facilitating intelligent behavior emergence. However, existing research primarily focuses on low-level data interfaces like sensors and actuators, neglecting the broader abstraction of hardware (Pinto et al., 2020a).

According to Pinto et al. (2020a), one of the major challenges in robotics is the development of architectures that can coherently integrate multiple cognitive capabilities within a unified framework. Core aspects such as knowledge representation, decision-making, and mission-driven reasoning require practical and structured mechanisms to embed artificial cognition into autonomous systems. For robots to operate autonomously and reliably, these components must be interconnected and supported by an organizational structure that enables the emergence of intelligent behaviors and seamless interoperation among agents. Cognitive architectures address these needs by providing mechanisms for adaptability, scalability, and coordinated action, as highlighted in previous works on cognitive frameworks for autonomous robotics (Pinto et al., 2019; Seleckỳ et al., 2015).

An important aspect of the UAV multi-vehicle systems is the communication strategy between each part. The vehicles need to be able to send and receive data to know each other's position, velocity, orientation, and flight autonomy and communicate with the Ground Control Stations (GCS) of the mission (Asaamoning et al., 2021). These network systems may be designed as IoT-based (Internet of Things) architectures or even be nodes of an IoT architecture itself, like fog (Gupta and Gupta, 2022), fog-cloud (Alharbi et al., 2022), or Edge-Fog-Cloud (EFC) networks (Chen et al., 2020). The present work is particularly interested in EFC designs due to its versatility and low exploration in robotics applications.

An EFC network represents a hierarchically structured computing framework that orchestrates data processing across multiple network layers to enhance IoT and smart application performance. "Edge" computing allows data processing to occur at the source of data generation, effectively reducing response times and conserving bandwidth by not sending all data to the central cloud (Yi et al., 2015). The "fog" layer acts as an intermediary that offers localized processing and storage, facilitating lower latency and network traffic, and supporting mobility and geographical distribution (Chiang and Zhang, 2016). Finally,

the “cloud” provides a backbone with substantial computational power and storage for non-time-critical and complex processing tasks (Vaquero and Rodero-Merino, 2014). This tripartite model benefits from the immediacy of edge computing, the geographic distribution of fog computing, and the extensive resources of cloud computing, presenting a scalable and efficient solution for modern computing challenges (Dastjerdi and Buyya, 2016).

Edge computing is an emergent paradigm that brings computational capabilities to the periphery of the network, closer to data sources like IoT devices and sensors. This approach is instrumental in reducing latency, enhancing data processing speed, and mitigating bandwidth issues typically associated with centralized cloud computing (Shi et al., 2016; Satyanarayanan, 2017). By preprocessing data locally, edge computing allows online analytics and swift decision-making, which are crucial in applications such as autonomous vehicles, smart cities, and healthcare monitoring systems (Ahmed and Rehmani, 2017; Premsankar et al., 2018). In this research, robotics groups functioning as edge nodes in a network are modeled as the closest agents to the user (or operation goal). These robots are, in fact, the ones executing the mission they’re applied for, and dealing with this operation’s events. These events can be navigation-related (a constraint for the robot continuing the trajectory) or mission-related (finding a goal or receiving a command from another robot), for example:

- Environmental Monitoring: Detecting changes in environmental conditions, like temperature, humidity, or the presence of certain chemicals or gases, using onboard sensors.
- Obstacle Detection: Identifying potential hazards or obstacles in the path of a drone or robot, necessitating a reroute or other immediate action.
- Data Collection: Gathering specific data points from the field, such as images, sounds, or samples, and determining whether to process this data locally or send it to the fog or cloud layer for further analysis.
- Health Monitoring: Checking the status of the drone’s or robot’s systems and components, like battery levels, propeller function, or sensor health.
- Security Breach: Recognizing unauthorized access or breach in the monitored area, potentially requiring an alert to be sent to a central system or human operators.

- Task Completion: Completing a given task such as delivering an item, and then receiving the next set of instructions or moving on to the next phase of a mission.
- Communication Requests: Handling requests for communication from other nodes or the central system, which may include the transmission of data or coordination instructions.
- Anomaly Detection: Identifying abnormalities in data patterns that could signify critical events such as system malfunctions, environmental disasters, or unusual activity.

Fog computing is a distributed computing infrastructure that extends cloud capabilities to the edge of the network, enabling a new breed of applications and services. By bringing the advantages of the cloud closer to where data is produced and acted upon, fog computing facilitates low-latency, network-efficient data processing, which is essential in cases such as the Internet of Things (IoT), 5G networks, and online analytics (Bonomi et al., 2012; Yi et al., 2015). The decentralization inherent in fog computing allows for localized decision-making, which is critical in scenarios where immediate action is required, such as smart transportation systems and emergency services (Stojmenovic and Wen, 2014; Vaquero and Rodero-Merino, 2014).

Cloud computing has significantly influenced the field of robotics, enabling enhanced computational power, vast storage capabilities, and advanced robotics applications (Kuffner, 2010; Wan et al., 2016). Through the cloud, robots can access powerful processing resources to perform complex tasks, parallel collaboration, and utilize machine learning algorithms for improved autonomy and adaptability (Kehoe et al., 2015; Chen et al., 2018). This integration has given rise to the concept of 'cloud robotics', which leverages the cloud for offloading heavy computation and providing services like mapping, navigation, and vision recognition, dramatically expanding the potential of robotic systems (Qureshi, 2016; Saha et al., 2018). The present research is also interested in applying this computing concept in robotics alongside cognition for autonomous optimized and evolutive decision-making and control.

The concept of cognition in computing refers to the integration of cognitive capabilities into computer systems, enabling them to understand, learn from, and respond to their environment in a manner akin to human cognition (Langley et al., 2009; Modha et al., 2011). This approach leverages artificial intelligence and machine learning algorithms to

process and analyze vast amounts of data, make decisions, and solve problems more efficiently. The goal is to create systems that are not only reactive but also proactive and adaptive, capable of anticipating needs and offering solutions without explicit programming for every possible scenario (Cox, 2005; Reggia, 2013). Cognition in robotics, on the other hand, focuses on imbuing robots with cognitive functions to enhance their autonomy and interaction capabilities (Chella et al., 2009; Vernon, 2016). Cognitive robotics combines AI, machine learning, computer vision, and natural language processing to enable robots to perceive their environment, understand context, learn from experiences, and make informed decisions. Such capabilities are essential for complex tasks like navigation, manipulation in unstructured environments, and human-robot interaction, pushing the boundaries of what robots can achieve independently (Hawes et al., 2017; Krichmar, 2018).

In robotic systems, cognition refers to the computational capability to perceive, interpret, and act upon uncertain environments through adaptive reasoning processes such as learning, inference, and memory integration. Cognitive models enable robots to deal with incomplete sensing, dynamic conditions, and ambiguous information by continuously updating internal world representations and selecting actions that maximize mission objectives under uncertainty (Langley et al., 2020; Kotseruba and Tsotsos, 2020). In multi-robot systems, cognition supports decentralized coordination, local autonomy, and robust decision-making, allowing agents to cooperate despite communication delays, sensor noise, or unexpected events. For ARCog-NET, cognition is essential because it provides the foundation for adaptive trajectory planning, cooperative SLAM, and distributed decision-making across the edge–fog–cloud hierarchy. By embedding cognitive mechanisms into its architecture, ARCog-NET achieves higher robustness, scalability, and mission resilience than classical swarm-control and SLAM frameworks, particularly in dynamic or partially observable scenarios (González-Santamarta et al., 2025).

To cite some possible cognitive-network applications trends in robotics (Kinsner et al., 2022; Wang et al., 2022): *(i)* intelligent data processing (network layers can perform context-aware processing, understanding the significance of the data in relation to the robots' mission objectives); *(ii)* reduced latency for online decisions (it facilitates near-instantaneous decision-making by processing data in close physical proximity to the edge devices, essential for the autonomous operation of robots); *(iii)* bandwidth management and data prioritization (by intelligently analyzing data, the network - mainly fog layer - can

prioritize its transit to the cloud, managing network resources efficiently); *(iv)* autonomous local storage and caching (it decides what data should be stored locally and what should be discarded or sent to the cloud, optimizing network and storage resources); *(v)* decentralized cognitive functions (the fog and edge layers enable distributed cognition, allowing a robot swarm to adapt to dynamic environments through collaborative intelligence. The fog nodes may also aid in the decentralized coordination of drones, for example, helping to manage communication, navigation, and operational algorithms among the swarm); *(vi)* enhanced resilience and redundancy (contributes to the robustness of the network by maintaining operations during cloud outages, utilizing its cognitive capabilities to manage a robot swarm independently); *(vii)* proactive security measures (cognitive network nodes, mainly in fog layer, can identify potential security threats and initiate protective protocols without waiting for cloud-based analysis); *(viii)* energy-efficient operations (distributed EFC optimizes the energy consumption of robots by processing data locally or in intermediary network layers, reducing the need for long-range communications for every processing load).

Some applications, such as surveillance, may also demand that the vehicles send and receive sensing data, for example, lidar readings, thermal images, or simple video footage frames (Scherer and Rinner, 2016). Therefore, for designing reliable network communication systems for multi-UAV applications, some factors must be considered, such as communication latency, data throughput, package loss, interference, processing delay in network nodes, and power constraints (Pinto et al., 2019). Given the significant volume of data that must flow through this kind of system, efficient compression strategies are very important to reduce data volume without losing too much information.

According to (Jayasankar et al., 2021), thousands of data compression research papers are published in the most important scientific bases. The author categorizes the techniques in compression methodologies based on data quality, coding schemes, data types, and applications. The present work mainly concerns compression based on application (wireless sensor networks) and data quality (lossy compression). Advances in Machine Learning (ML) and Deep Learning (DL) allowed the development of versatile models for data compression (Park et al., 2018).

Once DL is mainly based on training a model to extract features from input data (coding) and then reconstructing the input at the output based on those reduced features (decoding), it is possible to use those properties to reduce (code) the data generated in

network nodes, and then recreate it on its destination (decode). If necessary, the same node may send the compressed data by coding and decoding the original data, with some loss, a methodology described as autoencoder (Zhai et al., 2018). This paper aims to propose a linear autoencoder architecture based on a DL model and evaluate its training, test, and validation results. The trained autoencoder is integrated with a multi-UAV control simulation, which captures video data, evaluating the results of transmitting compressed and uncompressed image data during the simulation workout.

1.1- Main Contributions

In order to fill gaps in generic UAV swarm architectures to serve as a base for developing multi-aerial robots, proven both in simulation using real-world data and practical experimentation, therefore proposing solutions for the challenges in communication, control, and scalability pointed out in literature (Tahir et al., 2019b; Zhou et al., 2020; Khamis et al., 2021; Luthra, 2023; Abro et al., 2025; Checker et al., 2025; Alqudsi and Makaraci, 2025), the Aerial Robot Cognitive Network Architecture (ARCog-NET) (Ramos et al., 2024, 2025b) is proposed. It consists of a distributed cognitive framework designed to coordinate autonomous aerial robots operating collaboratively across Edge–Fog–Cloud (EFC) layers. It integrates online decision-making, visual perception, and reinforcement-based learning to support adaptive trajectory planning and cooperative task allocation and formation control among UAV swarms. By modeling knowledge reuse and hierarchical communication between agents, ARCog-NET enables resilient, context-aware behavior in dynamic and GPS-denied environments, bridging individual autonomy with collective intelligence.

Designed as a cognitive control framework, ARCog-NET enhances autonomous decision-making in swarm robotics by leveraging an Edge-Fog-Cloud (EFC) model. The system follows a decentralized loop-based operation, where multiple UAVs function as intelligent nodes executing parallel tasks at different levels of the network hierarchy. This distributed processing approach improves operational coherence, enabling UAVs to dynamically adapt to mission requirements while maintaining collective efficiency. The architecture integrates key functionalities such as online data management, inter-UAV

communication, intelligent task allocation, adaptive decision-making, and autonomous mission execution. By incorporating fog and cloud computing, it optimizes mission supervision, data analytics, and robotic coordination while ensuring scalability for large UAV formations. Communication efficiency within the network is the main focus, addressing critical parameters like latency, bandwidth utilization, interference mitigation, and power optimization to enhance overall swarm performance. The contributions of this work can be summarized as follows:

- Development of ARCog-NET, a modular, reconfigurable, and programmable framework for UAV swarm applications, that integrates cognitive-based decision-making processes for flexible and adaptive swarm behavior. The first published study of ARCog-NET (Ramos et al., 2024) addressed cognitive path planning decision making affecting navigation. The research evolved to present a generalization of the first algorithm in Ramos et al. (2025b), adapting the cognitive distributed structure to implement a dynamic decision-making process. The decisions were categorized into several types, as network, data, and navigation tasks. Then navigation gained two special cognitive subroutines for path planning and formation control, improving the study presented before. This thesis addresses the challenges in scalability, communication reliability, and energy efficiency through innovative swarm design and provides a generic framework adaptable to various UAV swarm applications, covering gaps in swarm literature.

- The proposed system uses a distributed Edge-Fog-Cloud (EFC) computing model to optimize data processing and support online decision-making across all network layers. In this model, computational tasks are intelligently allocated between the edge (onboard UAVs), fog (intermediate processing nodes), and cloud (high-capacity servers). By processing time-sensitive data locally at the edge and delegating more complex or less urgent computations to higher layers, the system minimizes latency, reduces network congestion, and enhances scalability. This hierarchical distribution ensures efficient resource utilization while maintaining the responsiveness required for dynamic and mission-critical UAV swarm operations.

- The architecture is intrinsically designed to handle large volumes of data through a dynamic compression mechanism tightly integrated with the cognitive decision-making process. This mechanism prioritizes the transmission of critical information

by selectively applying higher compression (and therefore tolerating greater data loss) to less relevant data, while preserving the quality of essential data streams. The data selection for compression may be realized automatically, depending on the mission's objectives or UAV status, or it may be previously programmed by the user. This adaptive strategy optimizes bandwidth usage and ensures that mission-critical information is transmitted with minimal degradation. The compression is implemented using a built-in autoencoder-based model. This Deep Learning (DL) model offers a versatile and low computational cost solution for data compression. It is suitable for use with various image formats within the UAV swarm system and is a built-in asset to ARCog-NET communication features. This integration enhances the efficiency of data transmission and processing within the multi-UAV framework. The implementation and testing are detailed in Ramos et al. Ramos et al. (2023).

- Use of real flight controller firmware in simulations to validate the architecture's performance in diverse scenarios, such as environmental monitoring, search and rescue, and infrastructure inspection, proving ARCog-NET robust and flexible design. Once ARCog-NET is integrated with Robotic Operating System (ROS), a widely used framework for developing robotic applications, this integration enables communication and interaction between ARCog-NET and ROS-based UAV systems, facilitating the implementation and testing of cognitive algorithms within real-world robotic environments. By simulating the behavior of UAVs and their interactions with ARCog-NET in a virtual environment, software-in-the-loop (SITL) testing allows for comprehensive testing and validation of the cognitive architecture's capabilities before deployment in physical UAV systems.

- A new approach on 3D mapping intelligence integrating multi-layer cognitive decision-making and adaptive trajectory optimization, while prior studies have emphasized only onboard resource constraints, visual-inertial consistency, or communication-efficient collaboration.

- The open science contribution, making all important ARCog-NET packages, simulation and experiment data available at the author's GitHub page (Ramos, 2025a), aims to foster transparency, reproducibility, and further research development. By sharing both the software modules and real experimental datasets, other researchers can replicate the results, validate the proposed methods, and extend the architecture to

new applications. This public availability also supports collaborative advancements in distributed robotic cognition and SLAM research, ensuring that ARCog-NET can serve as a practical foundation for future swarm intelligence studies.

1.2- Organization

The rest of this document is organized as follows. Chapter 2 describes the related works and the state-of-art. Besides, it introduces the necessary characteristics for cognitive architectures and decision-making process, with are essential for comprehending the ARCOG-net framework.

Chapter 3 presents the overall proposed architecture for creating the compression model and integrating it into the multi-robot simulation and the mathematical foundations for validating the methodology.

Chapter 4 presents the results in simulations and discusses the adopted algorithms. It shows the ARCOG-net being applied in three different scenarios.

The final Chapters 5 concludes the paper and presents ideas for continuing this research and improve ARCog-NET development and 6 presents the complete scientific production elaborated during the proposed architecture's development.

2- Background and Related Works

2.1- Cognitive Aspects

The field of robotics has seen significant advancements, evolving from simple mechanical structures to complex autonomous systems capable of interacting with their environment in increasingly sophisticated ways. Central to this evolution is the development of cognitive capabilities that enable robots to perceive, understand, and act effectively within their surroundings (Pinto et al., 2020a). According to Dominey and Warneken (2011), the fields of psychology and computer science are embracing a phase of collaboration driven by advancements in robotics technology. Contrary to simulations and classic AI applications, which are limited to virtual environments, robots possess the sensory and motor skills necessary to navigate and operate within the human physical realm, enabling direct interaction.

This section outlines the essential aspects of cognition that a robot must possess to achieve a higher level of autonomy and interaction. These aspects include perception, learning and adaptation, decision-making and problem-solving, and memory (Langley et al., 2009; Pinto et al., 2020a). By integrating these cognitive faculties, robots can better understand their environment, make informed decisions, learn from experiences, and interact with other robots meaningfully.

Perception is the foundational cognitive aspect, enabling robots to acquire and process information in order to take action (Haazebroek et al., 2011). Effective perception involves not just the passive collection of data but also the active interpretation of this data to understand the context and significance of environmental stimuli. This perception data is part of the cognition process, which also involves attention, association, memory, reasoning, judgment, imagination, thought, and language (Rumelhart, 2017)

For robots to operate autonomously in changing environments, they must be able to learn from experiences and adapt their behavior accordingly. Machine learning algorithms, including supervised, unsupervised, and reinforcement learning, are crucial for this cognitive aspect (Ogata et al., 2022). Adaptive learning enables robots to improve

their performance over time based on feedback from their actions and environmental changes. Autonomous robots must be capable of making decisions and solving problems to navigate complex environments and achieve their goals. This involves evaluating possible actions, predicting outcomes, and selecting the optimal course of action based on current objectives. Decision-making and problem-solving require synthesizing information from various cognitive processes, including perception and memory (Pinto et al., 2020a).

Note that this level of autonomy is crucial for robots operating in dynamic or unstructured environments, such as search and rescue operations and other applications (Pinto et al., 2019, 2020b). With cognitive functions, robots can learn from their experiences and adapt their behaviors to suit their tasks and environments better. Adaptability extends the utility of robots to a broader range of tasks and conditions, improving their efficiency and effectiveness in changing scenarios. Besides, cognitive processes enable robots to identify problems, generate potential solutions, evaluate their feasibility, and implement the most suitable ones. This capability is essential for robots tasked with complex missions where predefined algorithms may not suffice, such as navigating unknown terrains or handling unexpected obstacles in manufacturing processes.

Several works from the literature proposed architectures with cognition aspects to enhance the engagement between robotic systems and their environment (Langley et al., 2009; Yang et al., 2014; Pinto et al., 2020a; Laird et al., 2012; Insaurralde, 2014). The Intelligent Vehicle Control Architecture (IVCA) introduced in Insaurralde (2015) facilitates cooperation among various aerial vehicles. While IVCA is designed to be adaptable, its primary focus is on collaborative aircraft operations. It employs ontologies for its foundation but lacks details on hardware specifics or learning mechanisms. In contrast, Seleckỳ et al. (2015) developed a UAV framework emphasizing autonomous operations reliant on sensor data, yet it falls short in incorporating learning functionalities and its hardware needs vary with project scope. Aerostack (Sanchez-Lopez et al., 2017), stands out by offering a structured system for commanding and managing autonomous, diverse teams directly from the aircraft, focusing on navigation and avoiding obstacles. Despite its advantages, Aerostack's framework has room for enhancement in Human In The Loop (HITL) integration, team collaboration, and making autonomous decisions. There is still a clear demand for frameworks that incorporate cognitive abilities and effectively manage interactions between multiple robots and the HITL process.

2.1.1 Cognition in ARCog-NET

The implementation of cognition is essential to ARCog-NET for three main reasons. First, it ensures adaptive navigation, enabling UAVs to refine their trajectories over time instead of following static patterns typical of conventional SLAM-based swarms (Kinsner et al., 2022). Second, it improves cooperative efficiency, since agents can transfer knowledge across layers and avoid redundant exploration or unsafe decisions previously encountered by peers (Zuo and Xin, 2025). Third, cognition enhances scalability, allowing large swarms to maintain coordinated behavior through local decision-making reinforced by distributed memory, rather than centralized task allocation (González-Santamarta et al., 2025). As it will be demonstrated in both simulation and physical experiments, these cognitive mechanisms reduce trajectory drift, accelerate multi-UAV convergence, improve global map consistency, and maintain stable performance even as the number of agents increases.

In ARCog-NET, cognition is not a metaphorical or philosophical construct; it is a computational process that enables each UAV to adapt its navigation and decision-making based on prior experiences, shared knowledge, and online environmental cues. Cognition emerges from the interaction of three algorithmic components: *i.* A memory system storing past trajectory outcomes, coverage history, and map information; *ii.* A reinforcement-based decision model that updates influence weights according to the quality of actions taken by the UAV or its peers; and *iii.* A hierarchical information-sharing mechanism across the edge–fog–cloud layers. Together, these elements allow UAVs to evaluate the effectiveness of previous decisions, reuse successful strategies, and avoid repeating ineffective or unsafe behaviors.

Operationally, cognition in ARCog-NET is implemented as symbolic reasoning through two core structures: the knowledge base κ_{L_i}, which records historical coverage and environmental information, and the decision-weight tensor ω_{L_i}. These two structures encode, respectively, the coverage history matrix $H_{L_i}(t)$ and the decision-weight matrix $W_{L_i}(t)$ across time. The knowledge tensor stores symbolic representations of which regions were explored, by which agents, and under which conditions, while the weight tensor stores symbolic associations between agents and the effectiveness of their past recommendations. Symbolic reasoning is used to infer which past situations resemble the

current one, which nodes demonstrated reliable behavior under similar conditions, and which trajectory adaptations are likely to maximize performance (Himabindu et al., 2023).

This form of symbolic cognitive processing, tightly coupled with reinforcement learning, provides a hybrid reasoning capability similar to models proposed for cognitive multi-robot control and hierarchical learning architectures The reinforcement mechanism updates influence weights according to the observed reward (e.g., improved coverage, reduced collision risk, lower latency), enabling the system to converge toward progressively more efficient and safer behavior (Himabindu et al., 2023). This process constitutes the cognitive cycle: perceive the environment, evaluate past experience, select an action biased by learned knowledge, observe the outcome, and update the internal belief model.

Cognition plays a structural role in the distributed architecture. At the edge layer, cognition enables UAVs to adapt their trajectories without relying exclusively on external commands, reducing reaction latency in dynamic environments. At the fog layer, cognition supports consistent cluster-level coordination by allowing fog nodes to reuse prior map alignments, conflict-free flight patterns, or avoidance maneuvers learned by their subordinate UAVs. At the cloud layer, cognition enables long-term reasoning by aggregating multi-agent histories into global patterns, which can later guide the swarm during new missions. Because cognition is distributed rather than centralized, the architecture avoids single-point bottlenecks and maintains robustness against communication degradation, intermittent connections, or partial agent failure.

2.1.2 Decision-Making

Decision-making in the context of robots refers to the processes and mechanisms that allow a robot to autonomously select actions based on its perceptions of the environment, its objectives, and its internal state. Having a decision-maker on board allows UAVs to operate effectively in highly dynamic settings, even in the absence of centralized coordination. This capability enables UAVs to adapt seamlessly to any alterations in their surroundings. Traditional approaches to autonomous decision-making typically rely on extensive training with large datasets, which limits their ability to provide a form of intelligence that can be applied across a wide spectrum of scenarios (Das, 2016).

In sectors with critical demands like the aerospace industry, it's essential for robotic systems to not only understand the decision-making process but also exhibit predictable and stable actions. Several techniques meet these criteria, with the most commonly employed algorithms for decision-making including Neural Network (NN), Expert System (ES), Evolutionary Computation (EC), Data Mining (DM), and Case-Based Reasoning (CBR) (Liao, 2005). DM is crucial for decision support systems, as it involves modeling and identifying patterns to extract knowledge effectively. Meanwhile, EC is adept at handling numerous criteria, making it ideal for addressing intricate problems in the real world. CBR is advantageous due to its minimal computational demands and strong representation skills, yet it falls short in generating decisions for previously unseen scenarios. NN have shown the most remarkable outcomes, though they necessitate a substantial volume of data for training. This study employs a Reinforcement Learning (RL) approach for making decisions.

RL is a machine learning method that teaches an agent to make optimal decisions through experiments. It focuses on the agent's interaction with an environment to achieve a specific goal, without explicit instructions on how to achieve it. RL, was introduced by Sutton and Barto (2018), is one of the most popular research fields in the context of machine learning, effectively addresses various problems and challenges of artificial intelligence.

By using RL, the robot is an agent that learns to make decisions through a system of rewards and penalties. Each action that the agent performs results in feedback from the environment, which can be positive (reward) or negative (penalty). The agent uses this information to formulate a strategy that maximizes rewards over time. According to Leonetti et al. (2016), managing execution and online planning involves dealing with models that may not always be reliable. Besides, identifying and responding to failures may not suffice for a robot. Thus, it's essential for an intelligent agent to not only recognize its errors but also to learn from them and minimize similar mistakes in the future. RL enables this capability, and it the algorithm that this thesis uses as a decision-making approach to the proposed framework ARCOG-net.

2.2- Data Compression

Regarding data compression, Liu et al. (2021) present an autoencoder model for compressing scientific floating-point data without losing key information in data. A comprehensive study on autoencoders was conducted, but the author's model still needs to be tuned to reach higher compression rates without losing crucial data. In La Grassa et al. (2022) and Zebang and Sei-ichiro (2019), it was proposed two models of linear autoencoders. In the first work, the model is trained with satellite multispectral and hyperspectral RGB images. The second citation used a densely connected autoencoder for data compression, which was trained with high-resolution images to get richer feature extraction, then used a U-net network to decrease the distortion caused by compression. Both models demonstrated improvements compared to other methods (such as traditional JPG) but were slow for parallel data compression, which is necessary for the present work proposal.

2.3- Task Allocation

Despite significant progress in UAV swarm architectures, existing models still face challenges in balancing scalability, real-world applicability, and robust autonomous decision-making. For instance, task allocation in UAV swarms has been extensively studied to enhance autonomous decision-making and optimize mission execution. Various approaches, including market-based mechanisms (Yang et al., 2020), auction-based models (Shin and Bang, 2017), and bio-inspired strategies (Gupta et al., 2016), have been proposed to address dynamic and cooperative task assignments. Additionally, heuristic methods such as task clustering and auction-based assignments have shown effectiveness in distributing workloads while minimizing computational complexity. Reinforcement learning has also emerged as a promising solution, allowing UAVs to adapt to environmental changes and resource constraints autonomously. However, these strategies often lack cognitive adaptability and do not explicitly integrate utility-based decision models or hierarchical coordination mechanisms.

2.4- Path Planning and Formation Control

Regarding path planning, PSO (Kennedy and Eberhart, 1995) and A* (Hart et al., 1968) algorithms remain popular for dynamic route optimization (Zhao et al., 2021), but many implementations focus on static environments or rely on centralized processing, limiting their responsiveness and scalability in swarm systems. Similarly, formation control techniques, including leader-follower and behavior-based models, have been explored for maintaining UAV formations (Liu et al., 2022), yet often disregard network dynamics, latency handling, and decision decentralization. Few architectures combine these components within an adaptive framework that also accounts for distributed cognition, resource awareness, and mission-level decision-making. Architectures like Aerostack (Sanchez-Lopez et al., 2017) and IVCA (Emel'yanov et al., 2016) provide modular solutions for UAV coordination but do not integrate multi-layer learning or edge-fog-cloud processing. In contrast, this work builds upon these insights by proposing ARCog-NET, an enhanced UAV swarm architecture that leverages cognitive computing, edge-fog-cloud processing, and reinforcement learning for improved autonomy and operational efficiency

The ARCog-NET performance is evaluated and compared with the literature models above for task allocation, path planning, and formation control. Those models are implemented in the ARCog-NET framework and run in parallel with ARCog-NET models presented in Subsections 3.2.1, 3.2.2, and 3.2.3. For Task Allocation, the evaluated parameters are efficiency (number of successful allocated tasks per total number of assigned tasks in a given time window), energy consumption per task (average energy used in executing a task), communication overhead (ratio between transmitted data for network commands or requests and total transmitted data in a given time window), adaptability (average time taken in reallocating tasks) and scalability (the number of connected UAV that don't saturate fog layer in transmitting data). Path Planning is evaluated and compared using convergence time (how fast the models find a solution), trajectory coverage (the reason between the traveled path and the calculated path), computational cost (time taken in each iteration of path algorithms), and adaptability and scalability. Finally, Formation Control is evaluated through formation error (mean absolute error of UAV in formation), energy consumption, adaptability, communication overhead, and robustness to delays (measures the ability of UAV in formation to maintain itself despite communication or

control signal delays between it).

It's also important to notice that some models can handle changes in dynamic environments. For the algorithms that are designed for static environments, ARCog-NET automatically deals with changes by "resetting" the routines when an event happens. For example, if a UAV detects an obstacle in a calculated path using the algorithm A*, to avoid this obstacle a new path will be calculated using A* from scratch, using only the new knowledge about neighborhood collected by the UAV and the network. The versatility of ARCog-NET is based on including such information in previous calculations, without needing to remake the path from scratch. Using cognitive weighting factors history in calculation also helps the ARCog-NET performance in task allocation, path planning, and formation control. And finally, depending on the UAV status when an event happens, the algorithms will not be even executed as the knowledge base may be able to provide a solution already calculated by this event in the knowledge base.

2.5- Related Works

This section aims to present correlated works and compare our proposed approach with recent trends in research. With advancements in fields like artificial intelligence, embedded intelligence, and neuroscience, researchers are increasingly exploring sophisticated models and frameworks to mimic cognitive processes, thereby driving innovation across various domains such as robotics, automation, and human-computer interaction. Besides, the escalating number of publications concerning collaborative robot systems signifies a paradigm shift towards multiple robot collaboration, where robots seamlessly interact and collaborate with humans and other machines in shared workspaces.

In this sense, with all this information, Figure 1 and Figure 2 display the number of articles in IEEE and Elsevier databases that present the terms "Cognitive Architecture" and "Collaborative Robot System." What is remarkable is that there's an apparent rise in the number of these articles over time. This suggests that these topics are becoming more popular in research. This is important because it shows that our work is in line with what many other researchers are interested in.

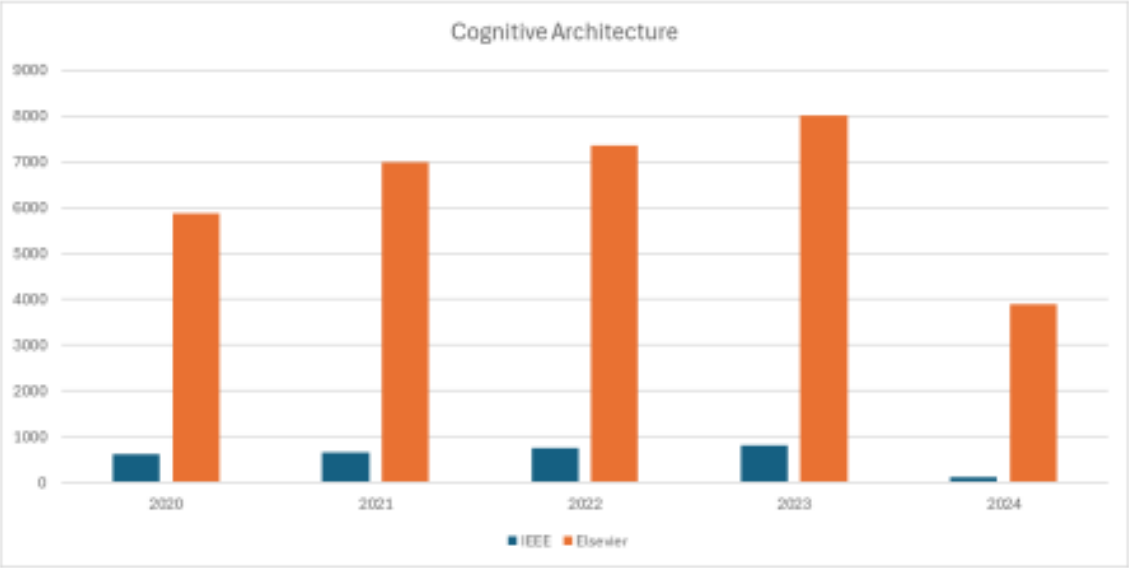


Figure 1 – Publications with the term "Cognitive Architecture."

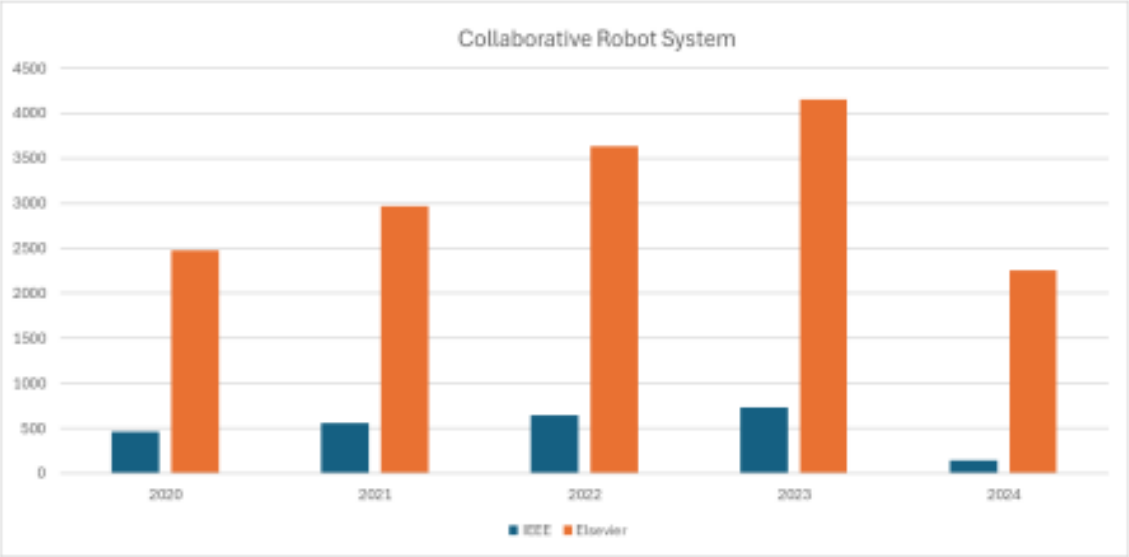


Figure 2 – Publications with the term "Collaborative Robot System."

2.5.1 Cognitive Architecture

In cognitive systems, a crucial aspect is data representation, which often manifests as a dichotomy between symbolic and sub-symbolic paradigms (Kelley, 2003). Symbolic representation involves direct symbols such as words and ontologies, facilitating knowledge verification and human understanding of decisions, making it particularly suitable for aircraft applications. Conversely, the sub-symbolic paradigm employs parallel data

representations, like neural network weights, which are not directly accessible (Pinto et al., 2020a).

Generic architectures like SOAR (Laird et al., 2012) and ACT-R (Ball, 2013) initially lack specific definitions for knowledge representation. However, subsequent works have proposed structures capable of both symbolic Laird et al. (2012) and sub-symbolic representations (Butz, 2016). For this research, symbolic representation primarily facilitates block interaction.

Another critical aspect is the system's ability to handle natural language. Extensions like NL-SOAR (Tweedale, 2014) and CHREST (Laird et al., 2012) enhance general architectures to comprehend and generate natural language outputs, supporting both simulated and physical environments. Our proposed architecture utilizes symbolic knowledge for block interaction, ensuring comprehensive understanding of reasoning processes.

Effective problem-solving and decision-making within dynamic environments are crucial for autonomous systems. Learning from previous experiences enhances the system's adaptability and reduces programming workload. Tweedale (2014) reviews decision-making methodologies in SOAR and ACT-R, utilizing expert knowledge and situational memory combined with rule-based logic. Similar approaches are seen in Selecký et al. (2015) and Sanchez-Lopez et al. (2017), incorporating heuristic methods and centralized activity management for tasks distribution and path planning. While some implementations, such as Sampedro et al. (2016), focus on specific tasks, broader and general mission decision-making capabilities remain underdeveloped in current research. The present work aims to address this gap by integrating comprehensive decision-making mechanisms suitable for diverse mission objectives.

In aerial applications, the need for consistent and predictable algorithms is essential to ensure proper verification and validation. Aerial Robotics Cognitive Architecture (ARCog), proposed by Pinto et al. (2020a) presents a new architecture focused on HITL and team coordination for autonomous operations. Their approach relied on language for information representation and decision-making processes. ARCog incorporates a functional block utilizing the Case-based Reasoning approach (Roth-Berghofer et al., 2012) to learn and make decisions. Their architecture is designed for compatibility with Linux-based hardware, enabling as close to online data processing onboard the aircraft.

2.5.2 Multi-Robot Collaboration

Several works presented ideas for simulating multi-UAV systems and their communication network integration. In Luna et al. (2022) is presented a multi-UAV application for area sensing with a novel path planning algorithm, with both simulation and field test robust results. However, besides presenting the network architecture and communication protocols, the research needs to consider the communication network constraints (such as latency) that influence the mapping mission and focus on something other than this issue when modeling the algorithm.

In (Alfeo et al., 2019) it is also presented a strategy for UAV swarm coordination based on the biomimetic behavior of insect groups. The research has a complete approach regarding group control modeling, and the authors presented a nice literature review regarding the theme. Still, it also needs more information on practical and physical conditions for those systems, hardware constraints, and communication architecture, presenting only mathematical simulations and their results.

The research in multi-UAV systems that most approaches and inspired the present work are based in Robot Operating System (ROS) (Quigley et al., 2009), with Gazebo (Gazebo, 2021) physics simulator integration. ROS is a collection of software frameworks for robot development, which provide the functionality of an operating system on a heterogeneous cluster of computers. ROS also provides standard operating system services, such as hardware abstraction, low-level device control, the implementation of commonly used functionality, message passing between processes, and package management.

Sets of running ROS processes are represented in a graph architecture (will be shown ahead) where the processing takes place in nodes that can receive and send messages, such as multiplex sensors, control, state, planning, actuator, and others. The system is used in the present work to develop the multi-UAV control packages and send and receive data between each node (or robot). The developed ROS packages also run the autoencoder (which will be detailed ahead) for the data compression in the simulation.

The ROS also have a package named MAVROS (ROS, 2022), which is an extendable communication node for ROS with a proxy for Ground Control Station. This package allows the developer to build applications using MAVLINK (MAVLINK, 2022), which is a very lightweight messaging protocol for communicating with drones and between onboard

drone components. This composition provides functionality for building abstraction of controllers, sensors, and actuators for real drones with PX4 Autopilot (PX4, 2022) firmware, simulating physics conditions in Gazebo and generating ready-to-deploy routines in real Flight Controller Units (FCUs).

The Gazebo is an open-source 3D robotics simulator. It integrates the Dynamic Open Engine (ODE) physics engine, OpenGL rendering, and code for sensor simulation and actuator control. The Gazebo can use various high-performance physics engines like ODE, and Bullet (Erez et al., 2015). It provides realistic rendering of environments, including high-quality lighting, shadows, and textures. It can model sensors that "see" the simulated environment, such as laser range finders, cameras (including fish-eye), Kinect-style sensors, etc. In the present work, Gazebo simulates the environment, the main sensor (an RGB camera), and the physics acting on the proposed framework.

In work presented in (Anagnostopoulos et al., 2022), a framework combining ROS and network simulators (Artery/OMNET++ (OMNET++, 2022)) is proposed, focusing on establishing a platform for cooperative autonomous vehicles system developments. This is a significant contribution since neither ROS nor Gazebo has built-in support for complex communication simulations (covering the OSI model). However, the paper needs more application examples and an evaluation of communication indicators within the proposed simulation framework.

Similarly to the previous citation, (Mohini, 2019) also presented a framework for integrating a ROS/Gazebo simulation with a benchmark network simulator NS-3 (NS-3, 2022). The author used the PX4 Autopilot (Ardupilot, 2021a,b) packages for ROS, building a multi-UAV system simulation that can be adapted for several communication systems between nodes or robots of the group (such as Wi-Fi or GSM). The authors also studied the effects of communication losses and delays, taking into account the system modeling robustness. Note that one of the contributions of the present work regarding this is the data compression, which the author did not cover.

Following a similar path, (Acharya et al., 2020) and (Calvo-Fullana et al., 2021) proposed similar frameworks for integrating ROS and network simulators for multi-robot systems development. Both types of research deal with time synchronization, as ROS/Gazebo simulations are based on continuous iterations and network simulators are based on discrete event time steps. Finally, (Pinto et al., 2019) presented a framework for analyzing fog-cloud computing cooperation applied to information processing in UAVs. The results

are based on mathematical data processing models in a multi-robot communication system and communication pipeline parameters. Still, they lack physical simulation results of real robots since the presented results rely only on mathematical modeling.

2.5.3 Simultaneous Location and Mapping (SLAM) in Distributed Robot Systems

Recent years have witnessed substantial advances in distributed SLAM (Simultaneous Location and Mapping) for UAV swarms. For example, Friess et al. proposed a fully onboard SLAM framework for nano-UAVs, capable of generating coherent grid maps using only low-power microcontrollers and tight sensing capabilities and achieving mapping accuracies of around 12 cm with minimal infrastructure dependence (Friess et al., 2023). This work demonstrates impressive scalability and autonomy. However, it lacks higher-level cognitive coordination and dynamic trajectory optimization, which ARCog-NET provides.

Xu et al. introduced D^2SLAM (Xu et al., 2022), a decentralized and distributed visual-inertial SLAM system designed for aerial swarms. It supports both near-field relative estimation and far-field global consistency, with robustness to network delays and scalability improvements via distributed optimization. While D^2SLAM focuses on SLAM robustness, ARCog-NET enriches this by integrating distributed cognitive decision-making, trajectory adaptation, and reinforcement learning. In the LiDAR domain, Zhong et al. presented Distributed Collaborative LiDAR SLAM (DCL-SLAM) (Zhong et al., 2022), a collaborative framework for robotic swarms that enables distributed loop closure and pose graph optimization using lightweight descriptors over ad-hoc network. Their architecture emphasizes communication efficiency and inter-robot consistency, yet doesn't address higher-level navigation and mapping synergy across ARCog-NET's Edge-Fog-Cloud layers.

Moreover, task offloading strategies have gained traction in recent literature. Aldossary et al. formulated a Mixed-Integer Linear Programming (MILP)-based collaborative offloading strategy, showing significant reductions in service latency (up to 59%) and energy consumption (up to 12%) via optimized allocation across UAVs, fog, and cloud nodes (Aldossary et al., 2024). Similarly, Hu et al. proposed a swarm-enabled mobile

edge computing system combining task offloading and data compression, using deep reinforcement learning (PER-DDPG) to minimize latency and energy costs by approximately 32% (Hu et al., 2025). These works align with ARCog-NET's multi-layer cognition, though ARCog-NET uniquely integrates SLAM-derived mapping and trajectory adaptation with cognitive reinforcement. Lastly, Tahir's comprehensive survey on edge and fog robotics highlights the emerging importance of fog robotics in multi-robot systems, providing low-latency offloading, local autonomy, and scalable coordination, foundational concepts that align with the design philosophy of ARCog-NET (Tahir and Parasuraman, 2025). In this scenario, ARCog-NET advances over survey-level architectures by demonstrating real UAV swarm deployment with integrated SLAM, cognitive navigation, and experimental validation.

To contextualize the contributions of ARCog-NET, it is compared with recent distributed SLAM and cognitive UAV frameworks. While prior approaches have emphasized onboard resource constraints, visual-inertial consistency, or communication-efficient collaboration, ARCog-NET uniquely integrates multi-layer cognitive decision-making, adaptive trajectory optimization, and 3D mapping through swarm intelligence. Table 1 summarizes the key parameters considered in each study.

Table 1 – Feature comparison among ARCog-NET and recent UAV swarm and distributed SLAM frameworks.

Study	**Mapping**	**Odometry**	**Cognitive Planning**	**Trajectory Adaptation**	**Communication Model**	**Multi-layer Processing**
Friess et al.	✓	✓	–	–	Local-only	–
D^2SLAM	✓	✓	–	–	Decentralized	–
DCL-SLAM	✓	✓	–	–	Ad-hoc	–
MILP-based	–	–	✓	–	Fog-Cloud	✓
Hu et al.	–	–	✓	✓	Fog-Cloud	✓
ARCog-NET	✓	✓	✓	✓	Edge-Fog-Cloud	✓

Overall, while existing works have substantially advanced distributed SLAM and multi-layer offloading, ARCog-NET combines cognitive trajectory decision-making, SLAM-based mapping, and hierarchical Edge-Fog-Cloud architecture in a real-world multi-UAV deployment context.

2.6- The Proposed Architecture

The present presents the development and validation of a multi-UAV simulation framework with ROS and Gazebo packages and integrate it with a data (image) compression DL model, evaluating its robustness within the robot system, physically implementing and deploying it on a collaborative indoor mapping as well. Regarding this and the related works previously presented, Table 2 provides a comparative overview of some of the cited works alongside key characteristics relevant to the proposed approach. In this table, "Yes" means the work addresses the respective aspect, and the symbol "-" means the aspect is not applicable or not mentioned in the work.

Most cited works demonstrate some decision-making capability, albeit with different approaches and methodologies. While some explicitly address this aspect, others do not focus extensively. Integration with ROS and/or Gazebo for simulation purposes is a common theme among the works, facilitating the development and testing of robotic systems in simulated environments.

However, fewer works concentrate on physical robot simulation, simulating real-world hardware components, and interactions. This aspect is critical for validating algorithms and behaviors in realistic settings. Similarly, data compression techniques, although not a primary focus in many works, are addressed in some ways to manage and transmit sensor data efficiently, especially in resource-constrained systems.

Table 2 – Comparison of Related Works with Key Aspects.

Work	Cognitive Features	Decision-Making	ROS/ Gazebo	Physical Simulation	Swarm Focus	Data Compression
Quigley et al. (2009) ROS	-	-	Yes	-	-	-
Laird et al. (2012) SOAR	Yes	Yes	-	-	-	-
Ball et al. (2013) ACT-R	Yes	Yes	-	-	-	-
Tweedale et al. (2014)	Yes	Yes	-	-	-	-
Yang et al. (2013)	Yes	Yes	-	-	Yes	-
Selecky et al. (2015)	Yes	Yes	-	-	Yes	-
Sampedro et al. (2016)	-	Yes	-	-	Yes	-
Butz et al. (2016)	Yes	-	-	-	-	-
Das et al. (2016)	Yes	Yes	-	-	-	-
Shin et al. (2017)	-	Yes	-	-	Yes	-
Sánchez-Rada et al. (2017) Aerostack	-	Yes	-	-	Yes	-
Pinto et al. (2019)	Yes	Yes	Yes	-	Yes	-
Pinto et al. (2020) ARCog	Yes	Yes	Yes	Yes	Yes	-
Yang et al. (2020)	-	Yes	-	-	Yes	-
Christos et al. (2021)	-	-	Yes	-	Yes	-
Acharya et al. (2021)	-	-	Yes	-	Yes	-
Calvo et al. (2021)	-	-	Yes	-	Yes	-
Liu et al. (2021)	-	-	-	-	-	Yes
Zhao et al. (2021)	-	Yes	-	-	Yes	-
Lunas et al. (2022)	-	-	-	Yes	Yes	-
Antonio et al. (2022)	-	-	-	Yes	Yes	-
Lagrassa et al. (2022)	-	-	-	-	-	Yes
Gupta et al. (2016)	-	Yes	-	-	Yes	-
Liu et al. (2022)	-	Yes	-	-	Yes	-
Mohini et al. (2023)	-	-	Yes	-	Yes	-
Proposed Approach (ARCog-NET)	Yes	Yes	Yes	Yes	Yes	Yes

3- Methodology

As previously defined, this thesis aims to provide a tool for developing swarm applications so that future developers don't have to worry about network communication, processing, storage, organization, and group functioning issues. The advantage of this is that researchers would only have to worry about control algorithms, sensors, and methodological strategies for the applications in which they use the ARCog-NET framework. This way, it is necessary to offer a reliable structure within the architecture package based on the proposed EFC concept that covers both the cognitive group and network control. This section will present implemented methodologies for building this framework, its validation assumptions, and comparison metrics with other works.

3.1- Network architecture

The basic cycle of ARCog-NET functioning is a loop-oriented collective algorithm in which each iteration happens in parallel at different network nodes (where a network node can be understood as an active robot of the swarm) that occupy different levels of the structure. Each level has specific tasks to accomplish to keep the whole network working while the robots execute the mission that the swarm is tasked with. In general, data structures, flows, and filtering, information exchange, cognitive decision-making for network control and operation control, robot control actions, data processing, data storage, and human interaction possibilities for each network level that should run at each node. Figure 3 presents a basic representation of ARCog-NET with two groups of edge agents coordinated by two fog coordinators each and connected to a cloud mainframe for mission control and supervision.

In the context of the EFC cognitive UAV swarm network generated by ARCog-NET, various communication issues can significantly impact the system's overall efficacy and performance. To cite some:

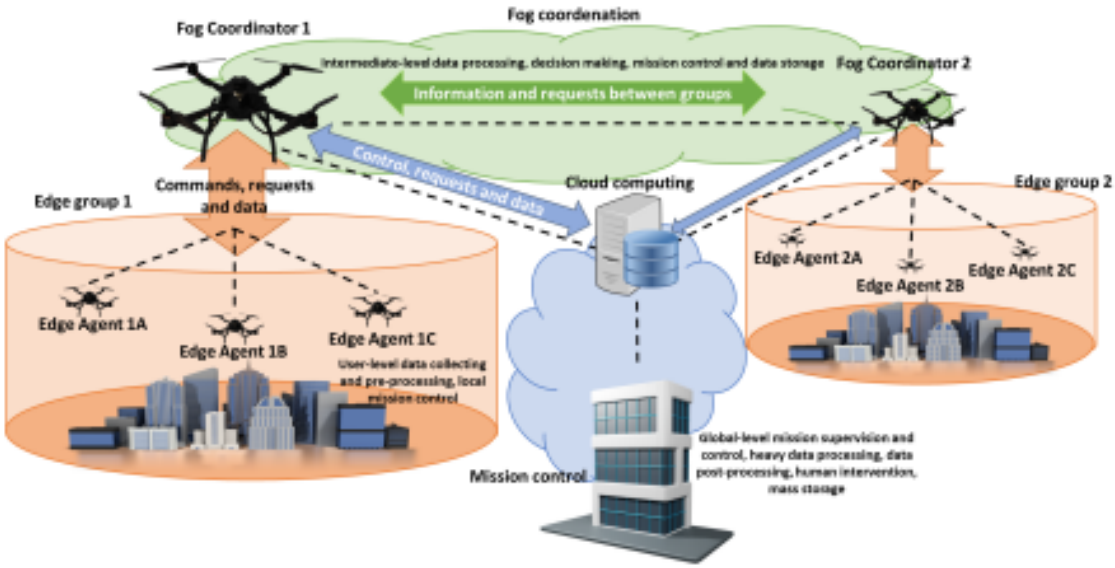


Figure 3 – ARCog-NET basic representation.

- Latency: Critical for online operations, latency becomes a concern when data must traverse multiple layers from edge to cloud, potentially delaying decision-making processes.
- Bandwidth Limitations: The drone's high volume of data can exceed the available bandwidth, leading to transmission bottlenecks, especially during peak data transfer periods.
- Interference: Communication can be disrupted by environmental factors, electronic interference, or competing signals, leading to data loss or miscommunication.
- Security and Privacy: Open wireless communication channels present vulnerabilities to security breaches and privacy intrusions, necessitating robust encryption and security protocols.
- Scalability: As the drone network expands, the communication infrastructure must scale accordingly without compromising performance or increasing latency.
- Energy Constraints: Drones are limited by their battery life, with communication protocols consuming significant energy. Thus, efficient energy use is paramount for sustained operations.

Overcoming these communication challenges is crucial for deploying effective and autonomous cognitive drone swarm networks, and the proposed framework is designed to

simulate those constraints, as will be discussed in the Subsection 3). ARCog-NET was also designed with a specific message format that keeps the same structure regardless of the dataflow between layers or nodes and is always broadcast through the MAVLink protocol. This message structure is basically an array of arrays following the representation presented in Figure 4.

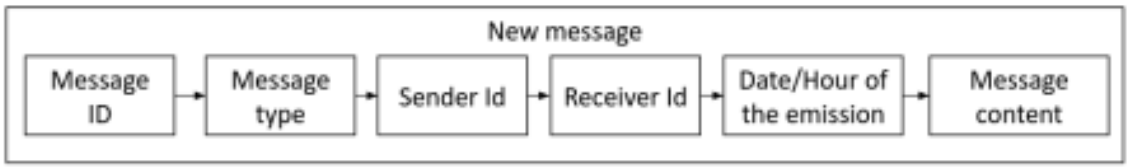


Figure 4 – ARCog-NET message structure representation.

Message content is the core of exchanging information, passing commands through the network, or requests addressed downstream or upstream. In the present architecture, these messages were divided into categories (or workflows), which represent the kind of message (data, processed data, request, or command) and the node (agent, coordinator, or server) they're being sent to or coming from. Table 3 represents these workflow descriptions, which will be important to understand the data structures of the layers in the following subsections dealing with the implemented architecture's algorithms and data. It is important to notice that future developers can add workflows to the basic structure.

Workflow type	Workflow description
A1	Agent data
A2	Agent processed data
A3	Agent request to coordinator
A4	Agent request to server
C1	Coordinator data
C2	Coordinator processed data
C3	Coordinator request to agent
C4	Coordinator request to server
C5	Coordinator command to agent
S1	Server data
S2	Server processed data
S3	Server request to agent
S4	Server request to coordinator
S5	Server command to coordinator
S6	Server command to agent

Table 3 – Complete Workflow Descriptions.

The simplified ARCog-NET functioning algorithm is presented in Figure 5. The

algorithm will be explained in parts in the next subsections.

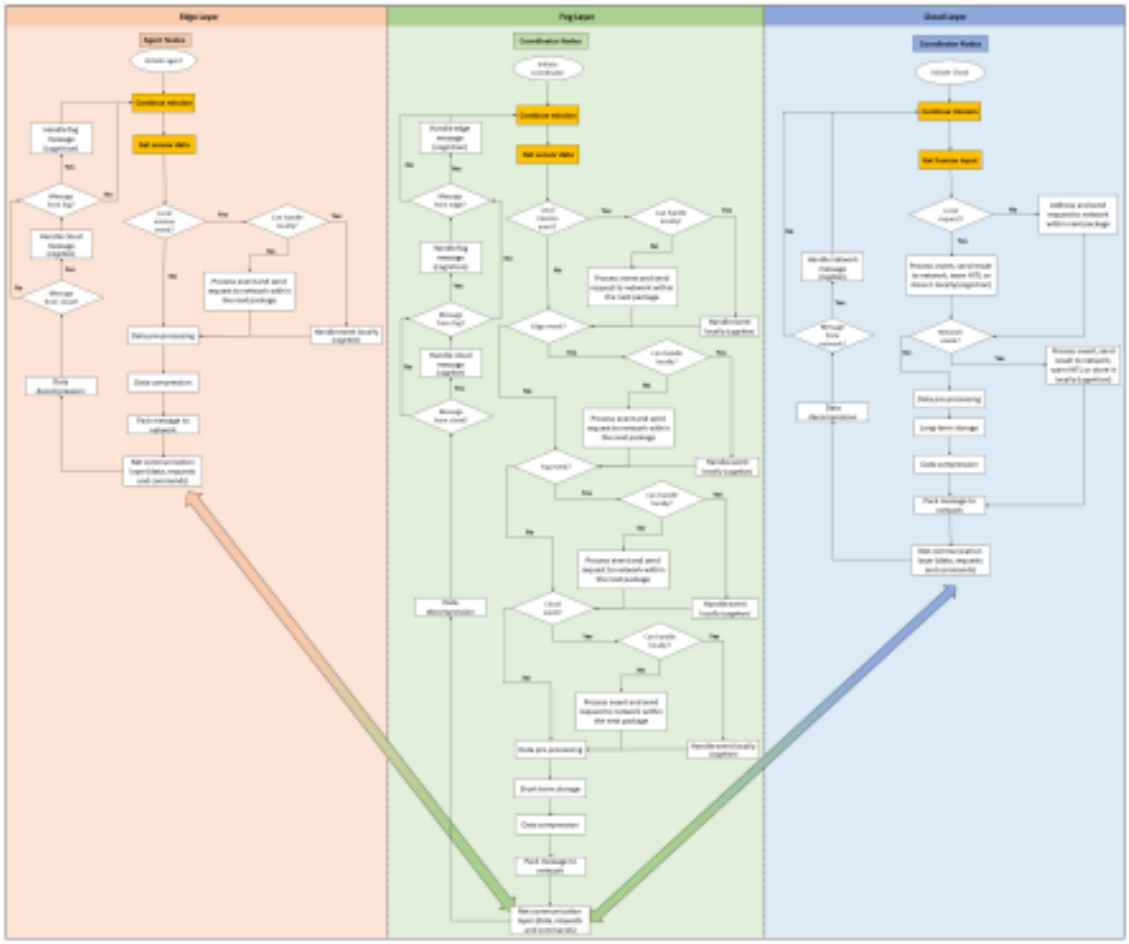


Figure 5 – ARCog-NET complete algorithm flowchart.

3.1.1 Edge nodes design

The edge-level nodes are designed as an end-user mesh of agents that directly execute the operation for which the swarm is designed. For example, collecting images for map generation by photogrammetry or exploring a partially known dynamic environment and sending data to the above layers. The agents, which are the edge nodes in the network structure, should mainly collect data or execute actions for the operation and perform simple data processing (such as segmentation, compression, decompression, or other processing that doesn't need too much computer load), make local decisions (for

example avoiding obstacles in case of collision and returning to home in case of network signal lost or low battery), execute commands from higher level layers (like sending or stop sending some data, switch group, stop mission, etc.) or send requests for it (such as request for change coordinator in case of low signal power, so the coordinator can verify if there is a close one in fog and switching it).

Edge computing nodes are strategically positioned at the network's periphery, closest to the source of data generation, which, in this case, are the individual robots within the swarm. This proximity allows for rapid data processing with minimal latency, enabling online decision-making and action without transmitting vast amounts of data back to a central cloud server. Each node is equipped with processing capabilities to perform some data analysis, run algorithms for local decision-making (cognition), and execute tasks such as obstacle avoidance and navigation. Furthermore, these nodes facilitate localized communication among nearby robots, reducing network congestion and enhancing system scalability. However, for making decisions that are not local (will affect nearby agents, coordinators, or the whole mission), taking actions, executing tasks, and analyzing and processing data with more complex algorithms that demand more computer power or storage data, the agent should send a requirement to its fog coordinators. To achieve this premise, the algorithm represented in the flowchart in Figure 6 was implemented for robots assigned as agents.

This flowchart describes the implemented functioning of agent nodes within an edge computing layer. The process begins with the initiation of an agent, which then continuously seeks to carry out its mission. As part of its operation, the agent node collects data from sensors. These two actions in the flowchart are highlighted in yellow because those are the packages on which developers may work when using ARCog-NET; since it is here, they will develop the control algorithms, sensor integration, and overall behavior of the swarm applications that the framework is conceived to be used as a tool. Once data is obtained, the agent determines if a mission event happened. If the event is local and can be handled by the node itself, the node processes the event locally using its cognitive capabilities.

For ARCog-NET project, mission events in the context of an edge computing layer for agent nodes, particularly within a network of autonomous drones, could represent a wide array of triggers and tasks that the system is designed to handle. These mission events typically require quick, autonomous responses from the agent nodes to ensure

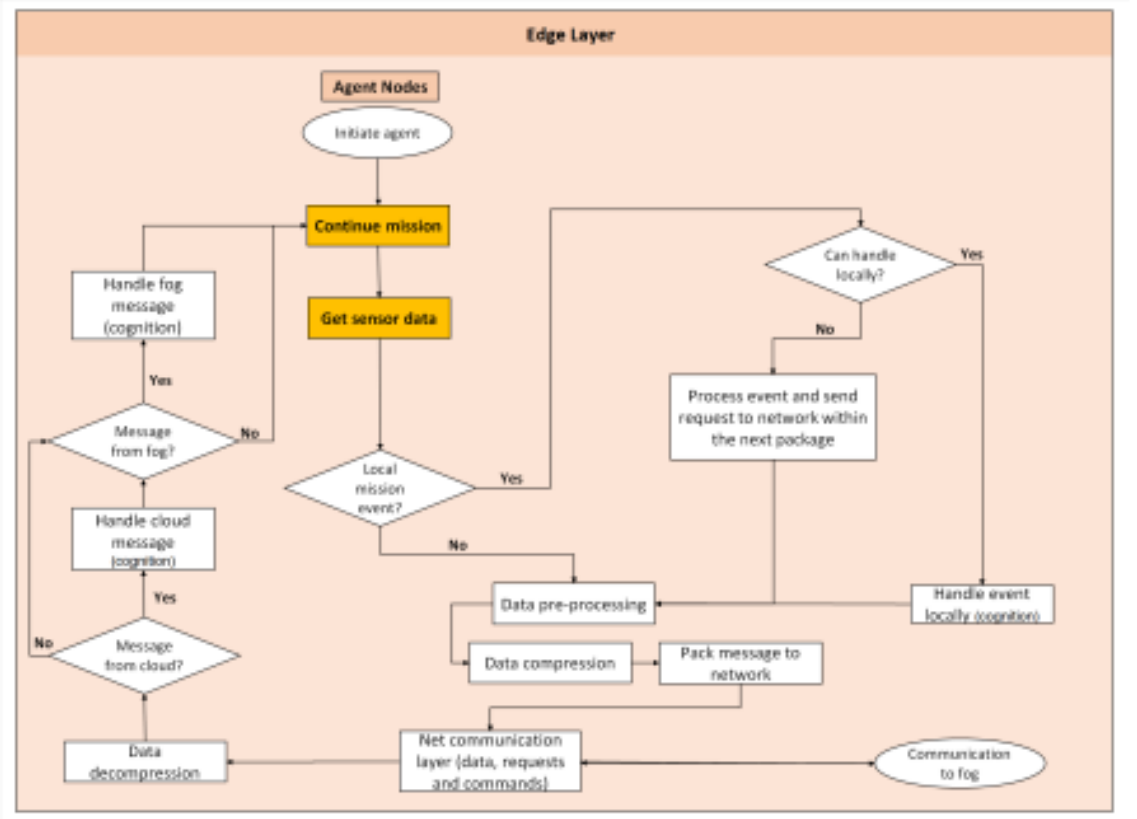


Figure 6 – Simplified edge node algorithm.

the smooth execution of their mission objectives. The flowchart is a visual algorithm representation that guides the agent nodes on responding to these events, whether by processing them locally or escalating them to higher computing layers such as fog or cloud for additional cognition and decision-making.

However, if the node determines that it cannot handle the event by itself, it prepares to communicate this information to the network. Before transmission, the data undergoes pre-processing, which likely includes filtering and reducing the data to the essentials needed for the specific task. This is followed by data compression, which minimizes the size of the data packet, making it easier and quicker to send across the network. The compressed data is then packaged and sent to the net communication layer with the data format presented in Figure 4.

In parallel to handling local events, the agent node is also equipped to handle messages from fog and cloud layers—higher levels of the computing hierarchy that provide broader oversight and can offload processing tasks or provide additional computational resources and intelligence. When a message is received from the fog layer, it is processed with cognition; if the message originates from the cloud layer, it also undergoes cognition-

based processing, indicating that the agent node is using cognitive methods to make decisions based on higher-level input.

If there are no messages from the fog or the cloud, the node continues with its tasks. The final step in the process shows that, after handling local data and interacting with the net communication layer, the agent node communicates with the fog layer, suggesting a two-way interaction where information is received from and sent to the fog layer. Overall, this algorithm outlines a complex data handling system and decision-making system that allows edge nodes to operate semi-autonomously while still being part of a larger, connected network that includes fog and cloud computing resources. Future developers may increase message types within the existing or new workflows. All these assets are addressed by the content of the message, presented in Attachment 6 for the agents.

3.1.2 The Fog layer

In this thesis' context, the fog layer refers to an intermediary computing layer that bridges the edge devices (the UAVs and their sensors) with the cloud. This layer is closer to the network's edge, hence the term "fog", which implies closer to the ground than the "cloud". In this thesis, although the fog layer has a coordination-oriented role (that's why the fog nodes are called coordinators), these robots acting in the network can also be used (but not dedicated) to edge-node tasks depending on the operation they're being deployed for. In summary, the cognitive fog layer is crucial in achieving a harmonious balance between the autonomy of drones and the centralized control provided by the cloud in ARCog-NET. It enhances the swarm's intelligence, responsiveness, and overall efficiency while acting as an intermediary for autonomous control coordination of the edge groups. The fog layer algorithm is presented in Figure 7.

The flowchart outlines the functioning algorithm of robots' processes within the fog layer of a distributed cognitive network, specifically focusing on coordinator nodes. The process begins with initiating a coordinator node, which proceeds to carry out its mission. The coordinator continuously gathers its own sensor data to determine if there's a local mission event. If the event can be addressed locally and requires cognitive processing, it is handled on the spot. Otherwise, it verifies if this event is an edge event coming from

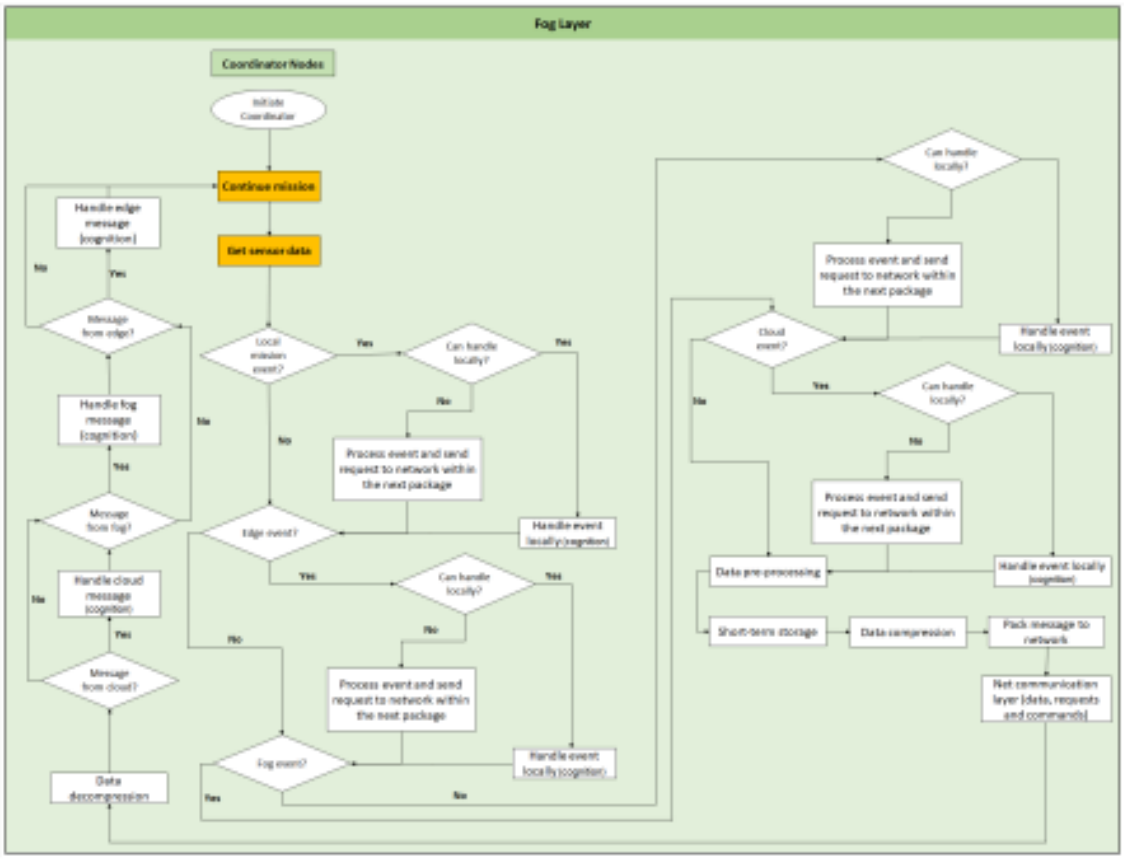


Figure 7 – Simplified fog node algorithm.

one or more agents from the group this fog coordinator is supervising (as a request-type message). If it is an edge event, the coordinator also verifies if it can be handled using its own cognitive processing or if this request must be addressed to the cloud computing service in the next message pack it will send. Continuing the process flow, the node checks for events from other fog coordinators and repeats the same processing to decide whether it can be handled by itself or has to be addressed to the cloud.

In instances where events cannot be handled locally or pertain to the cloud layer, the coordinator node processes the event and readies a request for the network to be sent in the next data package. This step is followed by data preprocessing activities, such as decompression and short-term storage, ensuring the data is primed for efficient transmission. Finally, the data is compressed and packed into a network message, ready to be sent across the network communication layer, which includes data requests (to the cloud or edge) and commands (only to the edge). Should there be an incoming message from the edge, the coordinator node determines whether it pertains to cognition and processes it accordingly. Similarly, it handles cognitive messages originating from

the fog and cloud layers, checking for efficiency and relevance before proceeding. This cyclical and conditional sequence of steps illustrates a dynamic and responsive system capable of making decisions at the fog level, optimizing for network efficiency and swift local processing where possible. The data structure of the ARCog-NET fog is presented in Attachment 6.

3.1.3 Cloud computing, control and supervision

The cloud layer is the central intelligence hub in ARCog-NET, providing deep analytical capabilities and expansive data management. It transcends the proximity-focused processing of edge and fog layers by offering a comprehensive view of the swarm's operations, harmonizing data from dispersed UAVs into actionable insights. The cloud's cognitive prowess enables sophisticated machine learning and AI algorithms to optimize flight patterns, resource allocation, and mission objectives, making it instrumental in processing complex operations requiring intensive computational resources. Such cognitive capabilities are integral for dynamic and adaptive swarm behavior, allowing UAVs to perform advanced tasks like online environmental monitoring and emergency response with enhanced autonomy and precision.

Supplementing the autonomous decision-making process is the pivotal role of the HITL, who acts as a supervisory presence within the cognitive architecture. While the system autonomously handles routine operations, the human supervisor monitors overall mission progress, making critical decisions when exceptional or unforeseen circumstances arise. The HITL ensures that the swarm adheres to overarching strategic objectives and operational guidelines, providing a critical safety net and ethical oversight for the autonomous operations of the swarm. By interfacing directly through the cloud layer, the human supervisor can inject high-level commands, redirect missions in response to global inputs, and guide the collective intelligence of the UAV swarm toward achieving complex, nuanced objectives that require a human touch. Figure 8 presents the algorithm for the cloud layer within ARCog-NET cognitive computing architecture, specifically for coordinating UAVs.

The process initiates with the cloud layer's supervisor node beginning its operation

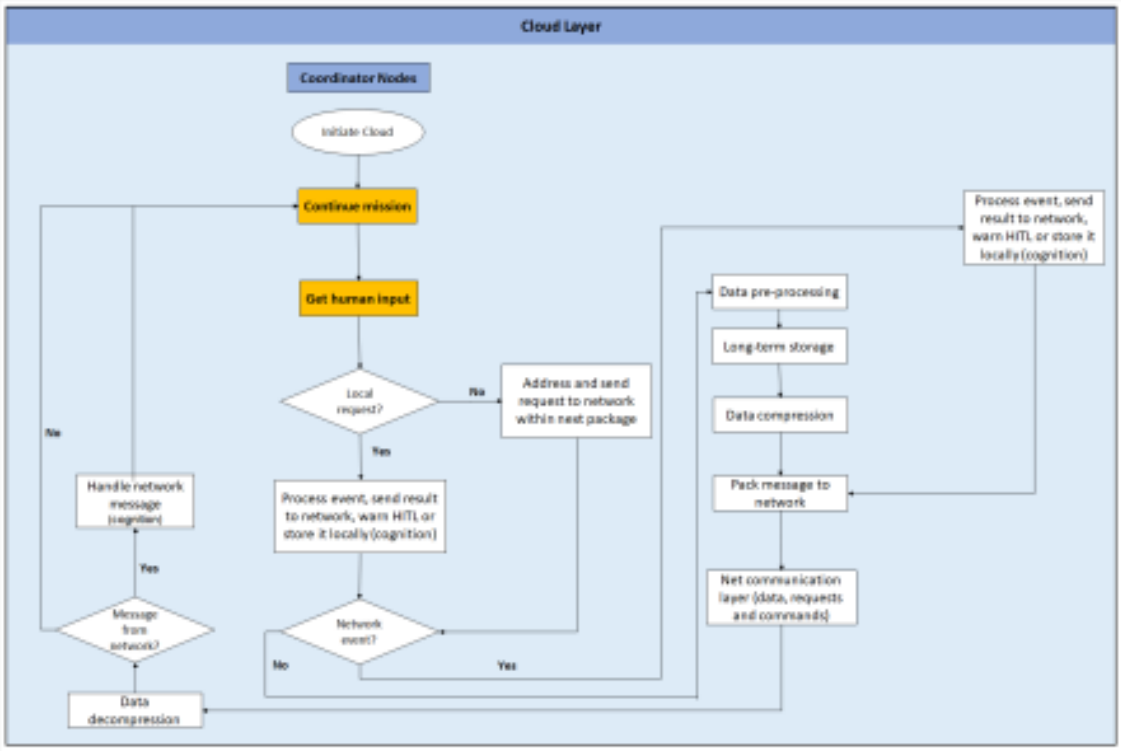


Figure 8 – Simplified cloud layer algorithm.

and continuing an ongoing mission. The first action the coordinator nodes take is to gather human input, which suggests that a level of human interaction or supervision is expected in the system's operation. Upon receiving human input, the system determines if the input constitutes a local request. If it is, the algorithm processes the event, after which the result may either be sent to the network, have its result communicated to a HITL for further instructions, or stored locally, based on a decision-making process incorporating cognitive evaluation. In case the input is not a local request, the system prepares to address it and send the necessary response back through the network in the next data package. It may be processed data for the network to know, a request for network information, or a command to a single UAV or a group of them.

Alongside processing human input, the cloud layer algorithm is also set to manage incoming network messages through cognitive processing, indicating an ability to interpret and respond to messages intelligently. If a message from the network is detected, the system carries out data decompression as needed. The algorithm also accounts for network events, which, if they occur, are processed with the potential for the result to be handled similarly to human input, either communicated to the network, sent to HILT, or stored locally, again indicating cognitive processing capabilities. After processing data for

both human input and network events, the system performs data pre-processing, likely filtering and organizing the data for further use. Subsequent steps include long-term storage, where the processed data can be archived, and data compression, optimizing the data for network transmission. Finally, the processed, stored, and compressed data is packaged and sent to the network communication layer, encompassing outgoing and incoming data, requests, and commands.

This loop demonstrates a complex, cognitive algorithm capable of handling various tasks, from processing simple local requests to managing broader network events, while maintaining a channel for human supervisors to intervene or be informed as necessary. This highlights the cloud layer's role as not just a data processor and storage facility but as a critical decision-maker in the UAV swarm's cognitive architecture. The data structures designed for this matter are presented in Attachment 6 of this thesis.

3.2- Theoretical modeling, cognition and flight algorithms

The input for the cognition in ARCog-NET can be modeled as an optimized task allocation problem, in which the utility factor of each task must be maximized. The weights of each decision will help locate the best solution of an operational event in each node search or create a new solution if it doesn't find one. A task is an action that each UAV should perform as a response to an event (avoiding an obstacle or covering a faulty neighbor, for example), and the weight of the task is the importance of that action for accomplishing mission goals. When an event occurs with a UAV, it employs a systematic approach for continuous learning and adaptation throughout the proposed algorithm, incorporating feedback into its respective knowledge bases and adjusting its decision strategies accordingly when executing the mission. If an optimal solution for allocating the task as a response to an event can not be handled by the UAV, it can send either a network request for a superior layer or a collaboration request to its neighbors from the same layer. This mechanism is called RL and allows the layers to operate independently and cohesively within the network's cognitive architecture. The basic RL process flowchart is presented in Figure 9.

Two specific tasks related to the UAV's navigation run separately in two cognitive

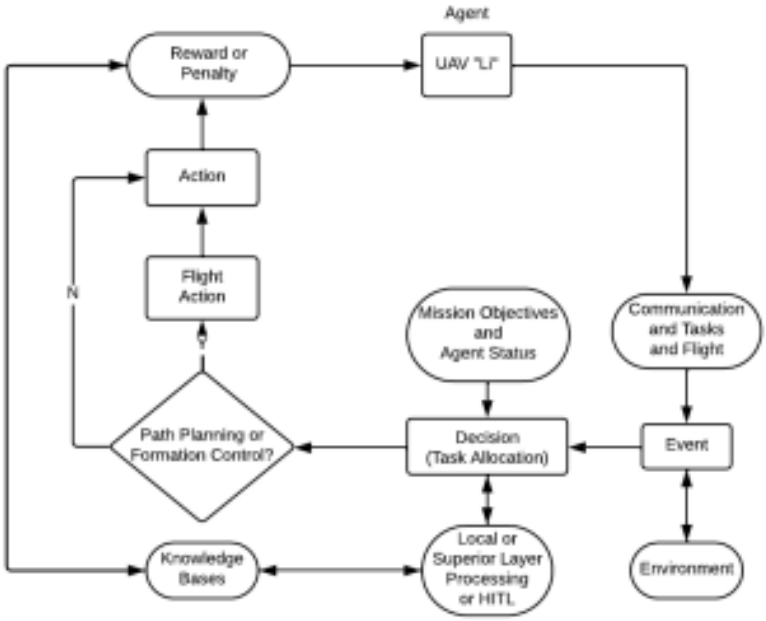


Figure 9 – Cognitive learning process in ARCog-NET.

subroutines. The first is the Path Planning (PP) task, where a UAV will calculate its navigation route or ask superior layers for a solution, learning the best solutions in the process and gaining access through the network to collective databases. The second is the formation control, which will be triggered every time a group must perform close and similar paths or if it is a formation control mission configured by the HITL. Therefore, ARCog-NET has 3 built-in cascade cognitive algorithms, one for general task allocation and the other two for specific cognitive flight sub-routines that may be triggered in navigation or flight tasks, if allocated. Users may configure, change, and improve those algorithms and triggers, facilitating UAV swarm implementation with the proposed architecture. In the next subsections, the mathematical models and algorithms of such features will be demonstrated.

3.2.1 Task Allocation

Tasks may have different natures when accomplishing mission goals. Let $\mathcal{T} = \{T_1, T_2, \ldots, T_R\}$ represent the set of all possible tasks that a UAV can perform, where R is the total number of task types. Each task $T_r \in \mathcal{T}$ is associated with a specific action, such

as:

- T_{nav}: Navigate to a new waypoint.
- T_{avoid}: Avoid an obstacle.
- T_{role}: Change role (e.g., edge agent to fog coordinator).
- T_{conn}: Change connection to a fog coordinator.
- T_{compress}: Adjust data compression level.
- $T_{\text{data_toggle}}$: Start or stop data transmission.
- T_{form}: Enter, leave or change formation.

Users may configure new tasks and subroutines within this task-dealing model. At any given time t, a UAV i in layer L may execute a task T_r or remain idle if no task is allocated. To efficiently allocate tasks, ARCog-NET employs a cognitive utility model. Each UAV evaluates the utility of executing a specific task for itself or another node on an inferior layer (if requested) based on the current mission state, network conditions, and task objectives. In the ARCog-NET model, the utility is understood as a quantitative measure that evaluates the benefit or fitness of a UAV (or node) performing a specific task at a given time. The state of UAV i on layer L (edge or fog) at time t is represented as:

$$S_{L_i}(t) = \left(P^{\kappa}_{L_i}(t), E_{L_i}(t), C^{D}_{L'_j, L_i}(t), \mathcal{T}_{L_i}(t) \right), \tag{1}$$

where $P^{\kappa}_{L_i}(t)$ is the current position of the UAV, $E_{L_i}(t)$ is the remaining energy, $C^{D}_{L'_j, L_i}(t)$ is the current task coverage[1], and $\mathcal{T}_{L_i}(t)$ is the buffer vector of assigned tasks. At each instant t, each UAV (or network node) keeps a weighting factor tensor keeping a multilinear relationship to tree weighting factor matrices that help decide tasks as a response to events, denoted by:

[1]Coverage refers to a quantitative measure that evaluates how effectively a UAV has addressed or completed a set of tasks in ARCog-NET. Path Planning coverage is a specific case which measures the error between a calculated trajectory and the actual flight trajectory and Formation Control coverage measures how well a UAV keeps its position in formation.

$$W_{L_i}(t) = \begin{bmatrix} W_{L_i}^{TA}(t) \\ W_{L_i}^{PP}(t) \\ W_{L_i}^{FC}(t) \end{bmatrix}, \quad W_{L_i}^{D}(t) = \begin{bmatrix} w_{L'_1,L_1}^{D}(t) & \cdots & w_{L'_m,L_1}^{D}(t) \\ \vdots & \ddots & \vdots \\ w_{L'_1,L_n}^{D}(t) & \cdots & w_{L'_m,L_n}^{D}(t) \end{bmatrix}, \quad D \in \{TA, PP, FC\} \tag{2}$$

in which $w_{L'_j,L_i}^{D}(t)$ is the weighting factors for activities D (TA for task allocation, PP for path planning and FC for formation control) of the task planning decision for the i-th robot at layer L commanded by node j at layer L'[2] [3] The matrix $W_{L_i}(t)$ attaches a physical meaning of trust or influence to each collaborating node. Each element $w_{L_i}^{L'_j}(t)$ represents how reliable node L'_j has proven to be, in practice, when suggesting motion for UAV i in layer L. High weights correspond to guidance that previously led to safe flight, efficient coverage, low collision risk, or fast completion of tasks in real space. Lower weights reflect advice that produced inefficient detours, communication delays, or near-collision behavior. The same is valid for coverages. The utility for UAV i to perform task T_r at time t is defined as:

$$U_{i,r}(t) = w_{L'_j,L_i}^{TA}(t) \cdot [\Phi_r^{\text{mission}}(t) + \Phi_r^{\text{network}}(t) + \Phi_r^{\text{energy}}(t)] + w_{L'_j,L_i}^{PP}(t) \cdot w_{L'_j,L_i}^{FC}(t) \cdot \Phi_r^{\text{navigation}}(t), \tag{3}$$

where:

- $w_{L'_j,L_i}^{TA}(t)$: Weighting factor for task allocation. This factor is randomly assigned using a continuous probability distribution for the first task the UAV receives. This is done for simplicity, and as the mission evolves, the weighting factors become more precise due to the continuous learning.
- $w_{L'_j,L_i}^{PP}(t)$: Weighting factor for path planning.
- $w_{L'_j,L_i}^{FC}(t)$: Weighting factor for formation control.
- $\Phi_r^{\text{mission}}(t)$: Utility of the task in achieving mission goals.

[2] Suppose, as an example, a formation with five UAV, being two for coordinators (*Fog*$_0$, *Fog*$_1$), one with two connected agents (*Edge*$_0$, *Edge*$_1$) and another with just one (*Edge*$_2$). The weighting factor $w_{L'_j,L_i}^{D}(t)$ of an activity D of the edge agent number 1 commanded by its coordinator is represented as $w_{Fog_0,Edge_1}^{D}(t)$.

[3] i may be equal to j and L may be equal to L' as well. If there's no connection between L_i and L'_j at time t, then $w_{L'_j,L_i}^{D}(t) = 0$.

- $\Phi_r^{\text{network}}(t)$: Utility of the task in improving network performance (e.g., reducing latency or packet loss).
- $\Phi_r^{\text{energy}}(t)$: Utility of the task in conserving energy resources.
- $\Phi_r^{\text{navigation}}(t)$: Utility of the task in UAV flight.

Each $W_{L_i}(t)$ matrix is directly associated with a historical task coverage matrix $H_{L_i}(t)$ of the same size. These historical task coverage matrices store the coverage of a task D assigned by a UAV j in layer L' to a UAV i in layer L at time t. The representation of $H_{L_i}(t)$ is given by:

$$H_{L_i}(t) = \begin{bmatrix} H_{L_i}^{TA}(t) \\ H_{L_i}^{PP}(t) \\ H_{L_i}^{FC}(t) \end{bmatrix}, \quad H_{L_i}^{D}(t) = \begin{bmatrix} C_{L'_1,L_1}^{D}(t) & \cdots & C_{L'_m,L_1}^{D}(t) \\ \vdots & \ddots & \vdots \\ C_{L'_1,L_n}^{D}(t) & \cdots & C_{L'_m,L_n}^{D}(t) \end{bmatrix}, \quad D \in \{TA, PP, FC\}. \tag{4}$$

The matrix $H_{L_i}(t)$ therefore encodes, for UAV i, which parts of the environment were physically explored by which assisting node L'_j (peer edge node, fog coordinator, or higher layer) at time t. Each entry $C_{L_k}^{L'_j}(t)$ can be interpreted as the measured contribution of node L'_j to the spatial coverage achieved by node L_k, e.g., how much of the currently relevant area was already mapped or cleared of obstacles thanks to shared information rather than a new flight.

Thus, each element of the matrix $H_{L_i}(t)$ represents the coverage outcome of a task computed by an individual j in layer L' for the corresponding individual i in layer L. In cases where $L_i = L'_j$, the decision-making occurs locally. Each element of $H_{L_i}(t)$ is linked to the corresponding weight matrix $W_{L_i}(t)$, where the elements of $W_{L_i}(t)$ quantify the effectiveness of different trajectory coverage options. These weights serve as reference points for querying the knowledge base. The historical weight tensor records for a UAV are represented as:

$$\omega_{L_i} = \{W_{L_i}(t_0), W_{L_i}(t_1), \cdots, W_{L_i}(t_n)\}, \tag{5}$$

and the knowledge base tensor, which stores the historical coverages, is represented as:

$$\kappa_{L_i} = \{H_{L_i}(t_0), H_{L_i}(t_1), \cdots, H_{L_i}(t_n)\}. \tag{6}$$

These tensors are used by the UAV or its coordinator to search for weighting factors previously applied to a similar task that this robot executed. This makes the UAV "consider" whether a task similar to the current one, allocated to it in a past time τ ($t > \tau$), has the same utility in the mission and therefore should be executed, or whether it is not relevant or a priority within its scope of current objectives. In physical terms, this hierarchical access corresponds to real wireless exchange of navigation history, mapped landmarks, and collision experiences between UAVs and coordinators, allowing one agent to benefit from another agent's prior flight without having to re-fly the same region. A UAV selects the task with the highest utility:

$$T_i^{\text{alloc}}(t) = \arg \max_{w_{\text{set}} \in W_{L_i}(t)} U_{i,r}(t), \quad w_{\text{set}} = \{w^{TA}_{L'_j,L_i}(t), w^{FC}_{L'_j,L_i}(t), w^{PP}_{L'_j,L_i}(t)\}. \tag{7}$$

Each w_{set} is associated to a historic task T_r that can be assingned as response to an event, which is also related to $C_{\text{set}} = \{C^{TA}_{L'_j,L_i}(t), C^{FC}_{L'_j,L_i}(t), C^{PP}_{L'_j,L_i}(t)\}$ groups. If multiple tasks have equal utility, the selection is based on predefined tie-breaking rules, such as prioritizing tasks critical to mission continuity or assigning tasks that minimize disruption to the swarm. The Mission Utility component ($\Phi_r^{\text{mission}}(t)$) can be defined by:

$$\Phi_r^{\text{mission}}(t) = \begin{cases} -\|\epsilon_{\text{goal}}(t)\|^2 \cdot C^{TA}_{L'_j,L_i}(t), & \text{if } C^{TA}_{L'_j,L_i}(t) \text{makes the respective task } T_r \\ & \text{contribute to the goal,} \\ 0, & \text{otherwise,} \end{cases} \tag{8}$$

where $\epsilon_{\text{goal}}(t)$ is the error between the UAV's current state coverage and the mission goal, and $C^{TA}_{L'_j,L_i}(t)$ is the task allocation coverage, defined as:

$$C^{TA}_{L'_j,L_i}(t) = \frac{n_{T_{L_i}}}{n_{\text{total}_{L_i}}}, \tag{9}$$

where $n_{T_{L_i}}(t)$ is the number of successfully completed tasks and $n_{\text{total}_{L_i}}(t)$ is the total number of tasks assigned to the UAV until time t, respectively. The Network Utility component ($\Phi_r^{\text{network}}(t)$) can be defined by:

$$\Phi_r^{\text{network}}(t) = -C^{TA}_{L'_j,L_i}(t) \cdot \left(\frac{1}{\lambda_{\text{max}}} \cdot \Delta\lambda_{\text{latency}} + \frac{1}{\varrho_{\text{max}}} \cdot \Delta_{\varrho_{\text{loss}}} \right), \tag{10}$$

where $\Delta\lambda_{\text{latency}}$ and $\Delta\varrho_{\text{loss}}$ are the latency and packet loss instant variation, normalized by their respective maximum tolerable values (λ_{max}, ϱ_{max}) defined by HITL. Notice that if the instant latency or packet loss variation values overcome the maximum tolerable values, the network utility component has high importance. Finally, the Energy Utility component ($\Phi_r^{\text{energy}}(t)$)is defined by:

$$\Phi_r^{\text{energy}}(t) = -C^{TA}_{L'_j,L_i}(t) \cdot \frac{P_{\text{task}}(t)}{P_{\text{max}}}, \tag{11}$$

where $P_{\text{task}}(t)$ is the power consumption associated with task T_r, and P_{max} is the UAV's maximum power capacity. The navigation utility $\Phi_r^{\text{navigation}}(t)$ is a specific utility that is applied only for flight tasks, denoted as:

$$\Phi_r^{\text{navigation}}(t) = \begin{cases} -\|\epsilon_{\text{pos}}(t)\|^2 \cdot C^{PP}_{L'_j,L_i}(t) \cdot C^{FC}_{L'_j,L_i}(t), & \text{if } T_r \text{ navigation coverages is both PP and FC,} \\ -\|\epsilon_{\text{pos}}(t)\|^2 \cdot C^{PP}_{L'_j,L_i}(t), & \text{if } T_r \text{ navigation coverages comprehends only PP,} \\ -\|\epsilon_{\text{pos}}(t)\|^2 \cdot C^{FC}_{L'_j,L_i}(t), & \text{if } T_r \text{ navigation coverages comprehends only FC,} \\ 0, & \text{if } T_r \text{ navigation coverages is null.} \end{cases} \tag{12}$$

being $-\|\epsilon_{\text{pos}}(t)\|$ the position error of the UAV in time t (position of the UAV in relation to the position it should be in trajectory, formation, or both). Notice that all utilities and the general utility for the UAV are calculated as consequences of the UAV's current state.

Tasks may be reallocated during the mission in response to events, such as changes in mission priorities or network conditions. If a UAV fails to execute a task due to resource constraints, the task is reassigned to a neighboring UAV or escalated to a higher network layer (e.g., fog or cloud). The reallocation process minimizes disruption and ensures mission continuity, which is a collaborative process. Collaboration between UAVs is facilitated through shared knowledge bases at the edge, fog, and cloud layers. Each UAV shares its utility values with neighboring nodes to coordinate task allocation. The collaborative utility $\mathcal{U}_r^{\text{collab}}(t)$ for task T_r is computed as:

$$\mathcal{U}_r^{\text{collab}}(t) = \frac{1}{|\mathcal{N}_i|} \sum_{\eta \in \mathcal{N}_i} U_{\eta,r}(t), \tag{13}$$

where $\mathcal{N}_i$ is the set of neighbors for UAV i. The task is allocated to the UAV with the highest collaborative utility. The overall task allocation problem for the swarm can be formulated as an optimization problem to maximize utility:

$$\arg \max_{z[T_{i,r}(t)] \in \mathcal{T}} \sum_{i=1}^{Q} U_{i,r}(t), \tag{14}$$

subject to:

$$\sum_{r=1}^{R} \zeta_{L_i,r} \leq 1, \quad \forall i, \tag{15}$$

where $\zeta_{L_i,r} \in \{0,1\}$ indicates whether task T_r is assigned to UAV i and $z[T_{i,r}(t)]$ is the function that correlates tasks T_r to the respective w_{set} and C_{set} within tensors in historic tensors.

The task allocation process is re-evaluated when a task-related event occurs, such as introducing a new task or a UAV failure. The edge layer recomputes the suitability scores and assigns tasks locally if possible. If no suitable task is found, the request is escalated to the fog layer, which aggregates task requests from multiple edge agents and optimizes (maximizes the utility of the task) global task allocation. The cloud layer handles critical task reallocation and HITL intervention. After a task execution, the success is evaluated and the weighting factors $w^D_{L'_m,L_n}(t)$ in the weight matrices $W_{L_i}(t)$ are updated by reward or penalty, generating a new entrance on the knowledge bases ω_{L_i} where it is executed.

To explicitly define the adaptive update process of the decision weights, the reinforcement learning component in ARCog-NET models the UAV's decision-making as a discrete-time optimization problem in which each agent i in layer L selects an action $a_t^{L_i}$ (e.g., trajectory adjustment or coordination choice) given its current state $s_t^{L_i}$, producing a reward $r_t^{L_i}$. The cumulative objective for each UAV is to maximize the expected discounted return:

$$J^{L_i} = \mathbb{E}\left[\sum_{t=0}^{T} \gamma^t, r_t^{L_i}\right], \tag{16}$$

where $\gamma \in [0,1]$ is the temporal discount factor. The reward function is defined as a weighted sum of mission-relevant performance indicators:

$$r^{L_i}t = \alpha_{C,C^{\text{total}}_{L_i}}(t) + \alpha_{E,E_{L_i}}(t) + \alpha_{R,R_{L_i}}(t) + \alpha_{L,L_{L_i}}(t), \tag{17}$$

where $C^{\text{total}}L_i(t)$ denotes the trajectory coverage defined in Equation (25), $EL_i(t)$ the normalized energy consumption, $R_{L_i}(t)$ the collision-risk indicator derived from inter-agent proximity, and $L_{L_i}(t)$ the decision latency of the node in milliseconds, all normalized in $[0, 1]$. The coefficients α_C, α_E, α_R, and α_L are tunable hyperparameters that balance exploration quality, safety, and resource usage. At every decision step, the weight matrix $W_{L_i}(t)$ is updated according to the temporal-difference rule:

$$w_{L_i}^{L'j}(t+1) = w_{L_i}^{L'j}(t) + \eta \cdot \delta_t^{L_i}, \tag{18}$$

where η is the learning rate and $\delta_t^{L_i}$ is the temporal-difference error computed as:

$$\delta_t^{L_i} = r_t^{L_i} + \gamma \cdot \max\{a' Q^{L_i}(s_{t+1}, a') - Q^{L_i}(s_t, a_t)\}, \tag{19}$$

with $Q^{L_i}(s_t, a_t)$ representing the estimated value of performing action a_t in state s_t. Each decision weight $w_{L_i}^{L'j}$ therefore represents the learned quality of adopting trajectory recommendations from node j at time t, gradually converging as the agent accumulates successful outcomes. The learnt weights are periodically normalized:

$$w_{L_i}^{L'j}(t) \leftarrow \frac{w_{L_i}^{L'j}(t)}{\sum_{k=1}^{N} w_{L_i}^{L'k}(t)}, \tag{20}$$

ensuring that influence among peer or superior nodes remains bounded and interpretable. To ensure reproducibility and isolate the contribution of each cognitive component prior to field trials, a simulation-based ablation is performed using digital UAV models and ARCog-NET topology within a indoor scenario (see the third scenario from Ramos et al. (2025b)), comparing three reduced configurations against the full ARCog-NET: (i) without decision-weight reinforcement (no w-update), where agents follow Equation (25) but do not adapt weights; (ii) without knowledge reuse (frozen κ_{L_i}), forcing purely local decisions and disabling cross-layer sharing; and (iii) without Edge–Fog–Cloud coordination, removing fog intermediaries so all agents communicate directly with the cloud. For each configuration, the ATE, RPE, trajectory coverage, knowledge reuse rate, decision latency, and failure recovery rate are recorded. This protocol quantifies the marginal effect of each component and demonstrates whether reinforcement-based weighting yields measurable gains in accuracy and robustness before real-world experimentation.

The task allocation algorithm is presented in Algorithm 1 and Figure 10 presents

the simplified task allocation process flowchart.

Algorithm 1 – Task Allocation Process in ARCog-NET.

Input: $S_{L_i}(t)$: State of UAV i at time t.
$\mathcal{T}$: Set of tasks $\{T_1, T_2, \ldots, T_R\}$.
$W_{L_i}(t)$: Weight matrix for UAV i at time t.
$H_{L_i}(t)$: Historical coverage matrix for UAV i at time t.
κ_{L_i}: Knowledge base for UAV i.
ω_{L_i}: Historical weight tensor for UAV i.
$\mathcal{L}$: Learning rate
Initialize: $T_i^{\text{alloc}}(t) \leftarrow \emptyset$
$U_v^{\text{collab}}(t) \leftarrow 0$ for all $T_v \in \mathcal{T}$
for *each task* $T_v \in \mathcal{T}$ **do**
  Compute $C^{TA}_{L^t_j, L_i}(t)$
  if $T_v \leftarrow T_{\text{new}}$ **then**
    Compute $C^{PP}_{L^t_j, L_i}(t)$ and $C^{FC}_{L^t_j, L_i}(t)$
  end
  else
    $C^{PP}_{L^t_j, L_i}(t) \leftarrow 0$ and $C^{FC}_{L^t_j, L_i}(t) \leftarrow 0$
  end
  for *Each* W_{L_i} *in* ω_{L_i} **do**
    n $\leftarrow$ index of W_{L_i} in ω_{L_i}
    $H_{L_i} \leftarrow \kappa_{L_i}[n]$
    [a, b, c] $\leftarrow$ index of [max($w^{TA}_{L^t_j, L_i}$), max($w^{PP}_{L^t_j, L_i}$), max($w^{FC}_{L^t_j, L_i}$)] in W_{L_i}
    $[C^{TA}, C^{PP}, C^{FC}] \leftarrow [H^{TA}_{L^t_j, L_i}[a], H^{PP}_{L^t_j, L_i}[b], H^{FC}_{L^t_j, L_i}[c]]$
    Switch Case D:
      Case TA and $C^D_{L^t_j, L_i}(t) \leq C^D$: $C^D_{L^t_j, L_i}(t) \leftarrow C^D$
      Case PP and $C^D_{L^t_j, L_i}(t) \leq C^D$: $C^D_{L^t_j, L_i}(t) \leftarrow C^D$
      Case FC and $C^D_{L^t_j, L_i}(t) \leq C^D$: $C^D_{L^t_j, L_i}(t) \leftarrow C^D$
    end
  end
  Compute $\Phi_v^{\text{mission}}(t)$, $\Phi_v^{\text{network}}(t)$, $\Phi_v^{\text{energy}}(t)$, $\Phi_v^{\text{navigation}}(t)$ using $S_{L_i}(t)$ and $H_{L_i}(t)$
  Compute $U_{i,v}(t) = w^{TA}_{L^t_j, L_i}(t) \cdot [\Phi_v^{\text{mission}}(t) + \Phi_v^{\text{network}}(t) + \Phi_v^{\text{energy}}(t)] + w^{PP}_{L^t_j, L_i}(t) \cdot w^{FC}_{L^t_j, L_i}(t) \cdot \Phi_v^{\text{navigation}}(t)$
  Compute $U_v^{\text{collab}}(t) = \frac{1}{|\mathcal{N}_i|} \sum_{\eta \in \mathcal{N}_i} U_{\eta,v}(t)$
  Update $U_{i,v}(t) \leftarrow U_v^{\text{collab}}(t)$.
  if $U_{i,v}(t) > U_{i,\text{alloc}}(t)$ **then**
    $T_i^{\text{alloc}}(t) \leftarrow T_v$
    $U_{i,\text{alloc}}(t) \leftarrow U_{i,v}(t)$
  end
end
if $T_i^{\text{alloc}}(t) \neq \emptyset$ **then**
  Execute $T_i^{\text{alloc}}(t)$.
  Update $H_{L_i}(t)$ and $W_{L_i}(t)$ based on task execution: $w^{D,\text{new}}_{L_i}(t) = w^D_{L_i}(t) + \mathcal{L} \cdot \Delta C^D_{L^t_j L_i}$
  Append $W_{L_i}(t)$ to ω_{L_i}.
  Append $H_{L_i}(t)$ to κ_{L_i}.
end
else
  Send a collaboration request to neighbors or escalate to a higher layer.
end
Return: $T_i^{\text{alloc}}(t)$
Output: Updated task allocation, UAV state, ω_{L_i} and κ_{L_i}

One of the primary challenges in UAV task allocation is balancing efficiency, energy consumption, and communication overhead, particularly in decentralized swarm architec-

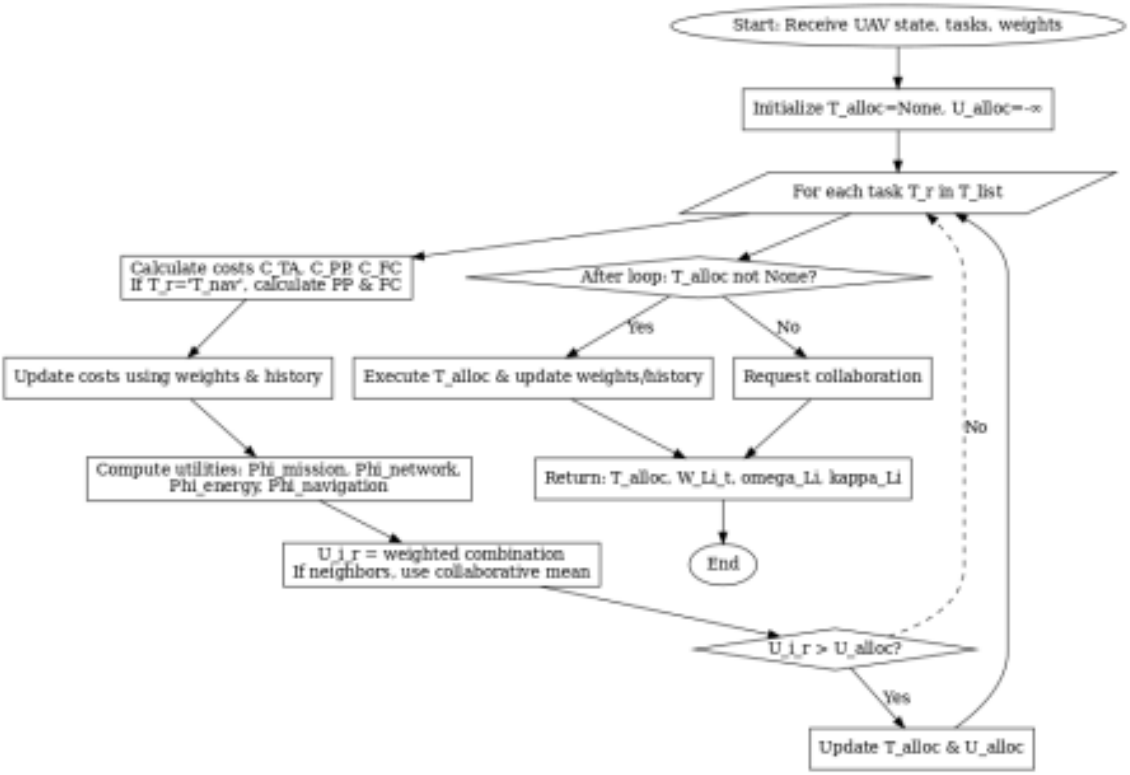


Figure 10 – Task Allocation (TA) algorithm flowchart.

tures (Gupta et al., 2016). Heuristic methods, such as task clustering and auction-based assignments, have demonstrated effectiveness in distributing workloads while minimizing computational complexity (Zhao et al., 2021). In addition, reinforcement learning has been explored for online task assignment, allowing UAVs to adapt to environmental changes and resource constraints autonomously (Liu et al., 2022). The ARCog-NET task allocation framework builds upon these advancements by incorporating cognitive reasoning and adaptive utility-based selection mechanisms. It enables UAVs to reconfigure their tasks dynamically while ensuring mission robustness and efficiency. Therefore, it will have task allocation results compared to those in this literature.

3.2.2 Path Planning

Navigation tasks (T_{nav}) are particularly important for ARCog-NET, as it is the most complex action a UAV node can execute. Whenever a navigation task is prioritized and allocated to a UAV, it enters a specific cognitive sub-process to calculate or recalculate

its trajectory. This approach was chosen for ARCog-NET flight navigation due to its efficiency in exploring search spaces, adaptability to dynamic environments, and suitability for multi-objective optimization in UAV path planning. Integrating cognitive models with path planning makes decision-making more advanced, allowing UAVs to perform complex tasks with minimal human input. This integration enhances mission efficiency and reliability, making it a powerful solution for UAV swarm applications in challenging environments.

As described in (Ramos et al., 2024), it is assumed that UAVs operate within a three-dimensional space $A \in \mathbb{R}^3$. This spatial domain physically represents the real operational volume of the swarm, where each coordinate (x, y, z) corresponds to a measurable position in meters relative to the laboratory environment. This distributed system consists of nodes structured into two operational levels: edge agents $e = \{e_1, e_2, ..., e_m\}$ and fog coordinators $f = \{f_1, f_2, ..., f_n\}$, where e_m and f_n define the maximum number of active units per layer. In physical terms, edge agents correspond to individual UAVs performing perception and navigation tasks, while fog coordinators act as higher-level processors integrating data from nearby UAVs to manage coordination and local map fusion. At deployment (t_i), UAVs, their group assignments and fog coordination roles are configured via Human-In-The-Loop (HITL) supervision at the cloud layer(Ramos et al., 2024, 2025b). This is the initial validation process before takeoff, equivalent to defining initial mission parameters and ensuring communication readiness.

The human supervisor can also pre-configure the initial trajectory points and possible deviations in flight paths due to operational events. Initially, each UAV follows a predefined trajectory (either computed algorithmically or set manually) in ARCog-NET $(t = 0)$, but modifications can occur dynamically at any time $t > 0$ in response to mission requirements. The number of objective trajectory points for the i-th UAV in a given layer $L \in e, f$ at the start of the operation is n^{L_i}. In contrast, the number of objective trajectory points at any moment t is represented as $n_p^{L_i}(t)$. This quantity is determined as follows:

$$n_p^{L_i}(t) = \sum_{\iota=1}^{M} \sum_{\gamma=1}^{N} \sum_{k=1}^{O} P_{L_i}^{\kappa}(x_\iota, y_\gamma, z_k), \tag{21}$$

where $t > 0$ represents a moment when trajectory modification is triggered (or a navigation task T_{nav} is allocated) due to an operational event, handled either locally or via commands from higher network layers. Here, $P_{L_i}^{\kappa}$ denotes the κ-th point of a UAV i in layer L following the trajectory ϱ_{L_i}. The parameters M, N, and O represent the maximum number of

coordinates in the x, y, and z dimensions, respectively, defining the total number of three-dimensional points in a finite set. The indices ι, γ, and k iterate over the spatial coordinates x, y, and z, and $P^{\kappa}_{L_i}(x_\iota, y_\gamma, z_k)$ indicates the presence of a point at the coordinate (x_ι, y_γ, z_k). Physically, this triple summation represents the discrete sampling of the UAV trajectory through the 3D workspace, where each $P^{\kappa}_{L_i}(x_\iota, y_\gamma, z_k)$ corresponds to an attainable spatial waypoint determined by navigation commands and sensor feedback. Equation (21) thus quantifies the total number of navigational points UAVs must traverse to complete their assigned tasks efficiently. This formulation is crucial as it dictates how UAV trajectories are dynamically optimized in response to changing environmental conditions, ensuring efficient coverage, minimal energy consumption, and reduced operational time—all essential for swarm-based autonomous UAV systems.

It is also important to know an incremental function denoted in Equation (22), which represents how much a point κ changes when it is modified by an event:

$$\Delta P^{\kappa}(t - t_i) = \begin{cases} \delta^{\kappa}_{p}, & \text{if } ||P^{\kappa}_{L_i}(t)_{\text{new}}|| - ||P^{\kappa}_{L_i}(t)_{\text{old}}|| \geq 0, \\ -\delta^{\kappa}_{p}, & \text{if } ||P^{\kappa}_{L_i}(t)_{\text{new}}|| - ||P^{\kappa}_{L_i}(t)_{\text{old}}|| < 0, \end{cases} \tag{22}$$

being $\delta^{\kappa}_{p} = ||P^{\kappa}_{L_i}(t)_{\text{new}}|| - ||P^{\kappa}_{L_i}(t)_{\text{old}}||$ at time t, $P^{\kappa}_{L_i}(t)_{\text{new}}$ is the new κ trajectory ϱ_{L_i} point and $P^{\kappa}_{L_i}(t)_{\text{new}}$ is the previous one. For a generic i-th flying robot assigned in a layer L (e or f) in the present framework, the position of a node's j-th "current objective" (doesn't matter the layer) at any given time t is $P^{L_i}_{\text{obj}_j}(t) \in \mathbb{R}^3$. For mathematical generalization, each UAV is assumed to navigate toward its objective at a fixed altitude[4], which may be adjusted dynamically in response to mission events through the path planning algorithm. Equation (22) physically expresses the displacement $\Delta P^{\kappa}(t - t_i)$ of a UAV between two instants, corresponding to how much its real position in 3D space changed after an event. The scalar δ^{κ}_{p} quantifies this spatial correction in meters, and the ℓ_2-norm measures the Euclidean distance traveled. A positive increment indicates forward progression in the mission area, while a negative one represents retreat or obstacle avoidance maneuvers.

The objective's position depends on the assigned navigation task, remaining constant (in applications such as photogrammetry) or changing abruptly due to operational events (such as obstacle detection or trajectory modification commands, for example). The resulting "objective motion" can be described as:

[4] In practice, an altitude range is set in the algorithm startup configuration, and the UAV control just corrects the UAV height if the altitude position is above or below the superior or inferior limits, respectively.

$$\begin{cases} \theta_{\text{obj}_j}^{L_i}(t) = \arctan\left(\frac{P_{\text{obj}_j}^{L_i}[y](t_i) - P_{\text{obj}_j}^{L_i}[y](t)}{P_{\text{obj}_j}^{L_i}[x](t_i) - P_{\text{obj}_j}^{L_i}[x](t)}\right) \\ V_{\text{obj}_j}^{L_i}(t) = \frac{\|P_{\text{obj}_j}^{L_i}(t) - P_{\text{obj}_j}^{L_i}(t_i)\|}{t - t_i} \\ P_{\text{obj}_j}^{L_i}[x](t_{i+1}) = P_{\text{obj}_j}^{L_i}[x](t) + V_{\text{obj}_j}^{L_i}(t) \cdot \cos\theta_{\text{obj}_j}^{L_i}(t) \cdot \Delta t \\ P_{\text{obj}_j}^{L_i}[y](t_{i+1}) = P_{\text{obj}_j}^{L_i}[y](t) + V_{\text{obj}_j}^{L_i}(t) \cdot \sin\theta_{\text{obj}_j}^{L_i}(t) \cdot \Delta t, \end{cases} \tag{23}$$

where $V_{\text{obj}_j}^{L_i}$ denotes the "objective's speed" (or how fast the position of the j-th objective changed considering the UAV as an inertial referential), and $\theta_{\text{obj}_j}^{L_i}$ is the "heading" of this objective (the direction in which this objective changed in relation to the UAV), both at any time t. The UAVs can operate at varying altitudes relative to ground level. At any time t and for any given layer (except the cloud, which remains fixed as it is not a UAV), the position of a UAV is represented as $P_{L_i}(t) \in \mathbb{R}^3$. Equation (23) represents the physical kinematic model governing UAV motion toward a mission objective. The heading $\theta_{\text{obj}_j}^{L_i}(t)$ corresponds to the real orientation angle in radians, derived from the relative displacement between past and current positions. The velocity $V_{\text{obj}_j}^{L_i}(t)$ represents the UAV's true linear speed (m/s), calculated by dividing the traveled distance by elapsed time. The last two expressions describe how these quantities update the UAV's Cartesian coordinates (x, y) over a discrete time step Δt, modeling continuous flight through sequential motion increments.

To simplify the mathematical model, UAVs are assumed to navigate between objectives using vectorial speed control while maintaining a constant altitude, as described by the following equations:

$$\begin{cases} \theta_{L_i}(t) = \arctan\left(\frac{P_{L_i}[y](t_i) - P_{L_i}[y](t)}{P_{L_i}[x](t_i) - P_{L_i}[x](t)}\right) \\ V_{L_i}(t) = \frac{\|P_{L_i}(t) - P_{L_i}(t_i)\|}{t - t_i} \\ P_{L_i}[x](t_{i+1}) = P_{L_i}[x](t) + V_{L_i}(t) \cdot \cos\theta_{L_i}(t) \cdot \Delta t \\ P_{L_i}[y](t_{i+1}) = P_{L_i}[y](t) + V_{L_i}(t) \cdot \sin\theta_{L_i}(t) \cdot \Delta t, \end{cases} \tag{24}$$

in which $V_{L_i}(t)$ and $\theta_{L_i}(t)$ denote the instant velocity and the instant heading of any UAV in a given layer, respectively. It is assumed that $C_{L_i}(t)$ is the trajectory coverage at time t of navigated points P_{L_i} of a UAV in relation to its $P_{\text{obj}_j}^{L_i}$ objective trajectory points. The coverage represents the trajectory traveled by a robot i between two consecutive moments t and $t + \tau$, where τ is a fixed time step. Equation (24) expresses the real planar motion

of a UAV during navigation. The heading $\theta_{L_i}(t)$ gives the instantaneous flight direction, while $V_{L_i}(t)$ represents its measured translational velocity derived from displacement. The coordinate updates for $P_{L_i}[x]$ and $P_{L_i}[y]$ physically describe how the UAV's position changes on the horizontal plane per time step Δt, corresponding to actual odometric movement tracked via onboard sensors or vision-based localization.

As ϱ_{L_i} is the defined trajectory of waypoints performed (or to be performed) by an i-th UAV connected to ARCog-NET at layer L in an observed time t (being t_i the previous time), and $P_\kappa(t)$ is the κ-th point of this trajectory, the total coverage $C_{L_i}^{\text{total}}$ of the trajectory can be described as:

$$\begin{cases} C_{L_i}^{\kappa}(t-t_i) = \Delta P_\kappa(t-t_i) \\ C_{L_i}^{\text{total}} = \frac{1}{n_p^{L_i}} \sum_{k=0}^{n_p^{L_i}} C_{L_i}^{\kappa}(t-t_i) \end{cases}, \tag{25}$$

where $n_p^{L_i}$ is the number of points observed in a time interval. This way, $C_{L_i}^{\text{total}}$ will depend on how efficiently $n_p^{L_i}$ is defined, especially when navigation events happen (which could be a detected obstacle by an edge agent or a command to change trajectory by a superior layer). Physically, $C_{L_i}(t)$ represents the portion of the environment (area or volume in m^2 or m^3) that has actually been visited, scanned, or mapped by UAV i over a given time interval, and therefore corresponds to effective spatial exploration progress. Thus, $C_{L_i}^{\text{total}}$ can be understood as the reward for a navigation trajectory decision, thus:

$$C_{L_j' L_i}^{PP}(t) = C_{L_i}^{\text{total}}. \tag{26}$$

In Equation (25), $C_{L_i}^{\kappa}(t-t_i)$ denotes the incremental contribution of a trajectory segment to the explored region, which is directly related to the physical path increment $\Delta P_\kappa(t-t_i)$ that the UAV actually flew in 3D space. The term $C_{L_i}^{\text{total}}$ is the accumulated fraction of mission waypoints covered by UAV i, normalized by $n_p^{L_i}$, and can be interpreted as a coverage efficiency score: values closer to 1 indicate that most intended locations have been physically reached and inspected.

Selecting a new trajectory in response to an event that results in a navigation task is determined by its expected impact on mission success, as defined by the optimal coverage process in Equation (25). Operationally, this means that higher physical coverage of the environment translates into higher reward during decision-making, incentivizing UAV behaviors that expand explored volume, avoid redundant revisits, and close blind

spots in the map. If a processing layer achieves the best coverage at a given time t, its corresponding weight $w_{L_i}^{L'_j}$ is reinforced. Conversely, suboptimal decisions are penalized, leading to adjustments in the weight tensor ω_{L_i}. This reinforcement learning mechanism continuously refines decision-making, prioritizing more effective solutions as the operation progresses.

When an event happens, by default, the method to calculate new trajectory points in the present work is the path planning algorithm of ARCog-NET, based on Kalami (2021). ARCog-NET integrates a cognitively enhanced spline-path algorithm for trajectory planning, significantly improving both processing time and trajectory accuracy over conventional methods. The UAVs store and utilize previously executed trajectories to construct new ones dynamically, reducing computational overhead when recalculating paths for new mission events. By incorporating reinforcement learning into the path planning framework, ARCog-NET achieves a more adaptive and efficient trajectory optimization strategy. The RL-enhanced path planning continuously adjusts its parameters based on online navigation feedback, improving convergence speed and solution quality.

In summary, when a navigation task resulting event occurs for a UAV i in layer L, it attempts to compute a trajectory adjustment to optimize coverage. The UAV first consults its weight tensor ω_{L_i} to identify whether a previously recorded weight matrix $W_{L_i}(t)$ contains a weight $w_{L_i}^{L'_j}(t)$ that surpasses others in effectiveness. If such a weight is found, it references a prior event processed by node L'_j and retrieves relevant historical decisions from the knowledge base tensor κ_{L_i}. This lookup directs the UAV to past coverage matrices $H_{L_i}(t)$ to inform its current trajectory choice. The implemented path planning algorithm for handling navigation events is detailed in Algorithm 2 and Figure 11 presents a simplified algorithm flowchart.

If the previously recorded decision yields better coverage than the newly computed result, the UAV will adopt the prior solution, update its weight matrix, store it in the knowledge base, and adjust the weight tensor, historical matrix, and learned policies accordingly. Otherwise, the UAV can either continue searching the knowledge base, use the newly computed result, or escalate the decision request to a higher-layer entity. This hierarchical decision process extends up to the cloud level, where, as a last resort, the human supervisor may intervene. Once the decision is executed, the UAV evaluates its real-world effectiveness by verifying coverage outcomes. If the chosen decision improves coverage, it is reinforced; otherwise, it is penalized, ensuring continuous optimization of

Algorithm 2 – Particle Swarm Optimization for ARCog-NET navigation event trajectory path planning.

Input: $T_r = T_{\text{nav}}$: A Path Planning (PP) task is allocated
ϱ_{L_i}: Initial trajectory positions of UAV i
$V^{L_i}_{\text{obj}_j}$: Initial velocities of trajectory points
κ_{L_i}: Current knowledge base and ω_{L_i}: Current decision weights history
t_i: Previous time and t: Current time
C_{min}: Minimum coverage threshold, max_{it}: Maximum number of iterations and min_{err}: Minimum error threshold
while *Error* $> \text{min}_{err}$ *and Iterations* $\leq \text{max}_{it}$ **do**
  for P *in* ϱ_{L_i} **do**
    $C_{L_i}(t - t_i) = \Delta P(t - t_i)$
    if $C_{L_i}(t - t_i) \geq C_{\text{min}}$ **then**
      Trajectory **keep** (P)
    end
    else
      Trajectory **drop** (P)
      New_Point = empty
      for *a in* ω_{L_i} **do**
        for *b in* $W^{PP}_{L_i}$ *(rows)* **do**
          for *c in* $W^{PP}_{L_i}$ *(columns)* **do**
            Get BestFitting($w^{PP}_{L'_j L_i}$)
          end
        end
      end
      $P \leftarrow \kappa_{L_i}[a]\{H_{L_i}[b, c]\}$
    end
  end
  Update Trajectory points:
  for κ *in* $range(length(P))$ **do**
    Get best fitting κ-th P:
    $V^{L_i}{}_{\text{obj}_{j_\kappa}}(t) = \frac{||P^{L_i}_{\text{obj}_{j_\kappa}}[x](t) - P^{L_i}_{\text{obj}_{j_{\kappa-1}}}(t)||}{t - t_i}$
    $\theta^{L_i}_{\text{obj}_{j_\kappa}}(t) = \arctan\left(\frac{P^{L_i}_{\text{obj}_{j_\kappa}}[y](t) - P^{L_i}_{\text{obj}_{j_{\kappa-1}}}[y](t)}{P^{L_i}_{\text{obj}_{j_\kappa}}[x](t) - P^{L_i}_{\text{obj}_{j_{\kappa-1}}}[x](t)}\right)$
    $P^{L_i}_{\text{obj}_{j_\kappa}}[x](t) = P^{L_i}_{\text{obj}_{j_{\kappa-1}}}[x](t_i) + V^{L_i}_{\text{obj}_{j_\kappa}}(t) \cdot \cos\theta^{L_i}_{\text{obj}_{j_\kappa}}(t) \cdot \Delta t$
    $P^{L_i}_{\text{obj}_{j_\kappa}}[y](t) = P^{L_i}_{\text{obj}_{j_{\kappa-1}}}[y](t_i) + V^{L_i}_{\text{obj}_{j_\kappa}}(t) \cdot \sin\theta^{L_i}_{\text{obj}_{j_\kappa}}(t) \cdot \Delta t$
    if $|C_{L_{i_\kappa}}(t) - C_{L_{i_{\kappa-1}}}(t)| \leq$ *Error* **then**
      Error $\leftarrow |C_{L_{i_\kappa}}(t) - C_{L_{i_{\kappa-1}}}(t)|$
    end
    $C^{PP}_{L'_j L_i}(t) \leftarrow C_{L_i\kappa}(t)$
  end
  Iterations $\leftarrow$ Iterations $+1$
end
$t_i \leftarrow t$
Create new $H_{L_i}(t)$ matrix incorporating the new $C^{PP}_{L'_j L_i}(t)$
Incorporate new $H_{L_i}(t)$ to knowledge base κ_{L_i}
Create new $W_{L_i}(t)$ matrix incorporating modified $w^{PP}_{L'_j L_i}(t)$
Incorporate new $W_{L_i}(t)$ to weights history ω_{L_i}
Output: $\varrho_{L_i} = P_{\text{optimized}}$: Optimized trajectory positions
κ_{L_i}: New knowledge base and ω_{L_i}: New decision weights history

the decision-making framework.

At the edge layer, the process begins with identifying an agent's unique tag number,

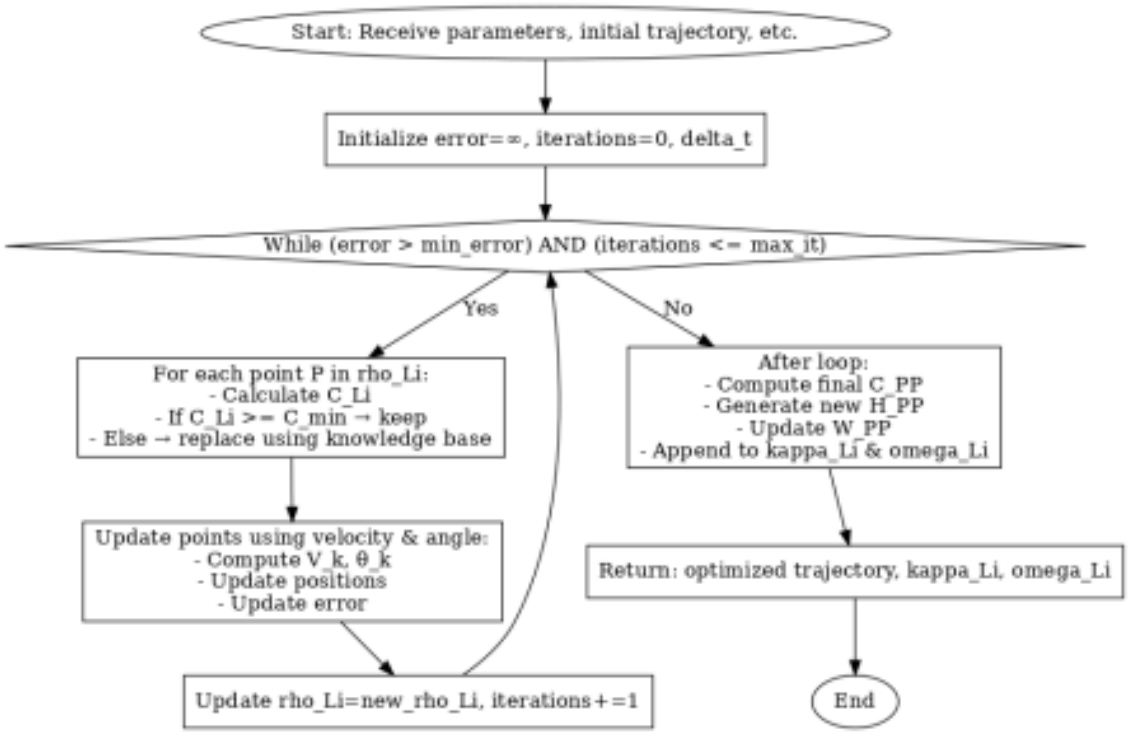


Figure 11 – Path Planning (PP) algorithm flowchart.

followed by retrieving its current position and objectives. These inputs are used to compute new trajectory objectives aimed at optimizing coverage while fulfilling mission goals. The proposed solution is then evaluated against the existing knowledge base. If a superior solution is found, it is added to the solution tensor and stored in the knowledge base. The agent then executes the action and assesses its effectiveness, adjusting the action weight accordingly. If no improved solution is found, but a previously executed action is deemed suitable, the agent may repeat it, update its weight, and store it as a new entry in the knowledge base. If no suitable solution exists, the edge agent escalates the request to the fog layer coordinator for further computation.

In parallel, the fog layer follows a similar process. It checks for incoming requests from edge agents and identifies relevant tags, whether from its own group or other fog coordinators. Based on whether it is handling an edge request or responding to an internal event, it computes new objectives to optimize trajectory coverage. The effectiveness of these objectives is evaluated against their knowledge base, with efficient solutions being reinforced and incorporated into the knowledge base. Upon completing a cognitive processing request or executing an action, the fog layer reassesses the effectiveness of the decision, updating weights and historical records before concluding its cognition

process.

Simultaneously, the cloud layer processes incoming requests and commands. When a layer L tag is detected with a path planning solution request, it collects relevant UAV data and objectives to compute new trajectory solutions to optimize flight coverage. These solutions are assessed against the cloud's extensive knowledge base. If a superior solution is identified, it is integrated into the solution tensor, stored in the knowledge base, and executed. Otherwise, a previously stored solution may be reused, as the cloud maintains a more comprehensive knowledge base. The cloud layer finalizes its decision-making process by evaluating the effectiveness of the executed solution, updating weights, saving the decision across all layers, and transmitting the optimized command back to the requester.

As UAVs must navigate dynamic environments while avoiding collisions, ARCog-NET enables rapid path adaptation in response to environmental changes, ensuring uninterrupted operation with minimal human intervention. In swarm applications, each UAV autonomously determines its trajectory while maintaining formation integrity and avoiding both static and dynamic obstacles. If necessary, UAVs can request trajectory recalculations from superior network layers. This functionality is crucial for missions in complex environments such as urban areas and cluttered terrains, where adaptability is essential. Additionally, ARCog-NET compares newly generated SPLINE trajectories with previously recorded decisions, allowing UAVs to replicate optimal past solutions, further improving efficiency through historical learning.

To evaluate its effectiveness, ARCog-NET path planning is benchmarked against classical PSO (Particle Swarm Optimization) (Kennedy and Eberhart, 1995), A* (Hart et al., 1968), Dijkstra (Dijkstra, 1959), and RRT (LaValle, 1998) in terms of convergence time, success rate, trajectory coverage, computational cost, and scalability. While classic PSO is efficient, it suffers from premature convergence and limited adaptability. ARCog-NET mitigates these issues by leveraging a cognitive navigation database, enhancing online decision-making in dynamic UAV operation. A*, though optimal for static pathfinding, lacks adaptability in dynamic UAV tasks. In contrast, ARCog-NET balances exploration and exploitation, delivering superior performance for complex UAV swarm operations. Dijkstra's algorithm guarantees shortest-path solutions in non-negative weighted graphs, making it ideal for static environments, whereas ARCog-NET achieves comparable results with enhanced adaptability. Finally, RRT, known for efficient high-dimensional space exploration

in path planning, is compared against ARCog-NET to demonstrate its ability to optimize search-space exploitation in dynamic UAV scenarios.

3.2.3 Formation Control

The ARCog-NET framework extends its cognitive decision-making processes for navigation to formation control, ensuring coordinated swarm behavior while maintaining compatibility with the path planning models defined earlier. If UAVs are flying close to each other or performing similar paths, ARCog-NET can switch from path-planning flight to formation control, for example. The formation control opportunity can be triggered by any layer, but only the fog or cloud layer can command it.

In other words, if an edge agent detects that its cluster is flying close to each other and in similar trajectories, it sends a formation control request to superior layers, which will decide if this is an optimal solution or not and respond with formation control commands to its agents or not. If a fog coordinator detects a formation control opportunity within its own cluster, it will send formation control commands to each agent. If a fog coordinator detects a formation control opportunity between its own cluster and another coordinator cluster, a request is sent to the cloud to initiate formation control between the two groups commanded by the cloud. If the cloud layer detects a formation control opportunity at any level, it will send commands to initiate it. And finally, if the HITL commands formation control through the cloud, it will send the commands to the respective cluster, group of clusters, or the whole network.

Once again let the position of the i-th UAV in layer L at time t be denoted as $q_{L_i}(t) = [x_{L_i}(t), y_{L_i}(t), z_{L_i}(t)]^T \in \mathbb{R}^3$, consistent with the trajectory definitions in Equation (24). The desired formation is defined through relative positions between UAVs, where $q^d_{L'_j}(t)$ represents the target position of UAV j of layer L' relative to UAV i of layer L. The formation error for UAV i aggregates discrepancies between actual and desired relative positions across its neighbors $\mathcal{N}_i(t)$:

$$e_{L_i}(t) = \sum_{j \in \mathcal{N}_i(t)} |(q_{L_i}(t) - q_{L'_j}(t)) - (q^d_{L_i}(t) - q^d_{L'_j}(t))|, \tag{27}$$

where $\mathcal{N}_i(t)$ dynamically updates based on communication range or mission requirements. To minimize this error, the control input $u_{L_i}(t)$ integrates proportional feedback, historical decisions from the knowledge base κ_{L_i}, and innovations from path planning (PP):

$$u_{L'_j L_i}(t) = K_p e_{L_i}(t) + \sum_{k \in L'_j} w^{FC}_{kL_i}(t) u^{\text{hist}}_{kL_i}(t) + \delta_{L_i}(t). \tag{28}$$

Here, K_p is a proportional gain matrix optimized via ARCog-NET path planning, $w^{FC}_{L'_j L_i}$ are softmax weights favoring historically effective solutions, and $\delta_{L_i}(t)$ represents PP-driven adjustments. The weights $w^{FC}_{L'_j L_i}$ are calculated using:

$$w^{FC}_{L'_j L_i} = \frac{e^{-\beta \| e_{L_i}(t) - e^k_{L_i}(t) \|}}{\sum_m e^{-\beta \| e_{L_i}(t) - e^m_{L_i}(t) \|}}, \tag{29}$$

where $e^k_{L_i}(t)$ denotes past formation errors stored in κ_{L_i}. The PP innovation term $\delta_{L_i}(t)$ minimizes a cognitive cost function balancing current error reduction and historical consistency. Therefore, $\mathcal{J}_p(t)$ can be described as formation stability, which is the necessity of adjusting UAV positions to keep the designed formation:

$$\mathcal{J}_p(t) = \| e_{L_i}(t) + \Delta q^p_{L_i} \|^2 + C^{FC}_{L'_j L_i} \sum_k w^{FC}_{kL_i} \| \Delta q^p_{L_i} - \Delta q^{k,\text{hist}}_{L_i} \|^2, \tag{30}$$

where $\Delta q^p_{L_i}$ represents candidate formation adjustments, and $C^{FC}_{L'_j L_i}$ is the formation control coverage, representing the adherence to historical solutions. The PP velocity update rule incorporates both local and global best solutions:

$$\Delta q^p_{L_i}(t+1) = \Delta q^p_{L_i}(t) + C^{FC}_{L'_j L_i} \cdot (\Delta q^{p,\text{best}}_{L_i} - \Delta q^p_{L_i}(t)) + C^{FC}_{L'_j L_i} \cdot (\Delta q^{g,\text{best}}_{L_i} - \Delta q^p_{L_i}(t)), \tag{31}$$

with $\Delta q^{g,\text{best}}_{L_i}$ drawn from $H_{L_i}(t)$ in κ_{L_i}. The knowledge base κ_{L_i} is augmented to store formation-specific data, which is the historical formation control coverages $C^{FC}_{L'_j L_i}$ components of the H_{L_i} matrices as described in Equation (4), ensuring compatibility with the trajectory coverage metrics in Equation (25).

During formation-disrupting events (e.g., obstacles, UAV failures), the framework triggers a hierarchical response. At the edge layer, UAVs compute local adjustments via PP and compare them to κ_{L_i}, rewarding effective solutions by updating $w^{FC}_{L'_j L_i}$. Conflicting proposals escalate to the fog layer, which resolves them by optimizing global formation

stability:

$$\max_{\{\Delta q_{L_i}\}} \sum_{i=1}^{N} \mathcal{J}_p(t) \quad \text{subject to} \quad \|q_{L_i}(t) - q_{L'_j}(t)\| \geq d_{\text{safe}}, \tag{32}$$

where d_{safe} enforces collision avoidance. The cloud layer intervenes for critical failures via human-in-the-loop (HITL) commands, updating κ_{L_i} with new formation templates. The existing path planning Algorithm 2 is extended to handle formation events by incorporating $e_{L_i}(t)$ as input and modifying the PP loop to optimize $\mathcal{J}_p(t)$. For example, after line 37, replacing line 38 of Algorithm 2, the following steps (Algorithm 3) are added:

Algorithm 3 – ARCog-NET PP Extension for Formation Control.

Input: $e_{L_i}(t)$: Formation error from Equation (27)
H_{L_i}: Historical formation adjustments from κ_{L_i}
W_{L_i}: Historical weightings adjustments from ω_{L_i}
while $\|e_{L_i}(t)\| > min_{err}$ *and Iterations* $\leq max_{it}$ **do**
    Compute $\Delta q^{p}_{L_i}$ via PP velocity update
    Evaluate $\mathcal{J}_p(t)$ using Equation (30)
    Update $w^{FC}_{L'_j L_i}$ using softmax weights
    if $\mathcal{J}_p(t) > \mathcal{J}_p^{best}$ **then**
        Update $\Delta q^{p,best}_{L_i}$ and $C^{FC}_{L'_j L_i}$
    end
    Iterations $\leftarrow$ Iterations $+1$
end

The simplified algorithm flowchart is presented in Figure 12. This approach ensures seamless integration of formation control into ARCog-NET's cognitive architecture, leveraging existing components like the knowledge base κ_{L_i} and reinforcement learning weights ω_{L_i} while preserving scalability across edge, fog, and cloud layers.

The performance of ARCog-NET formation control is evaluated against state-of-the-art formation control algorithms including Distributed Model Predictive Control (DMPC) (Liu et al., 2020), Deep Reinforcement Learning (DRL)-Based Formation Control (Pu et al., 2023), and Consensus-Based Formation Control with Event-Triggered Communication (Dai et al., 2022). Distributed Model Predictive Control, which optimizes formation trajectories in a decentralized manner, is compared in terms of computational efficiency and scalability for large-scale swarms. Deep Reinforcement Learning approaches, which leverage neural networks for adaptive formation control, are evaluated against ARCog-NET in terms of adaptability to dynamic environments and learning convergence speed. Consensus-based formation Control with event-triggered communication, known for reducing communication

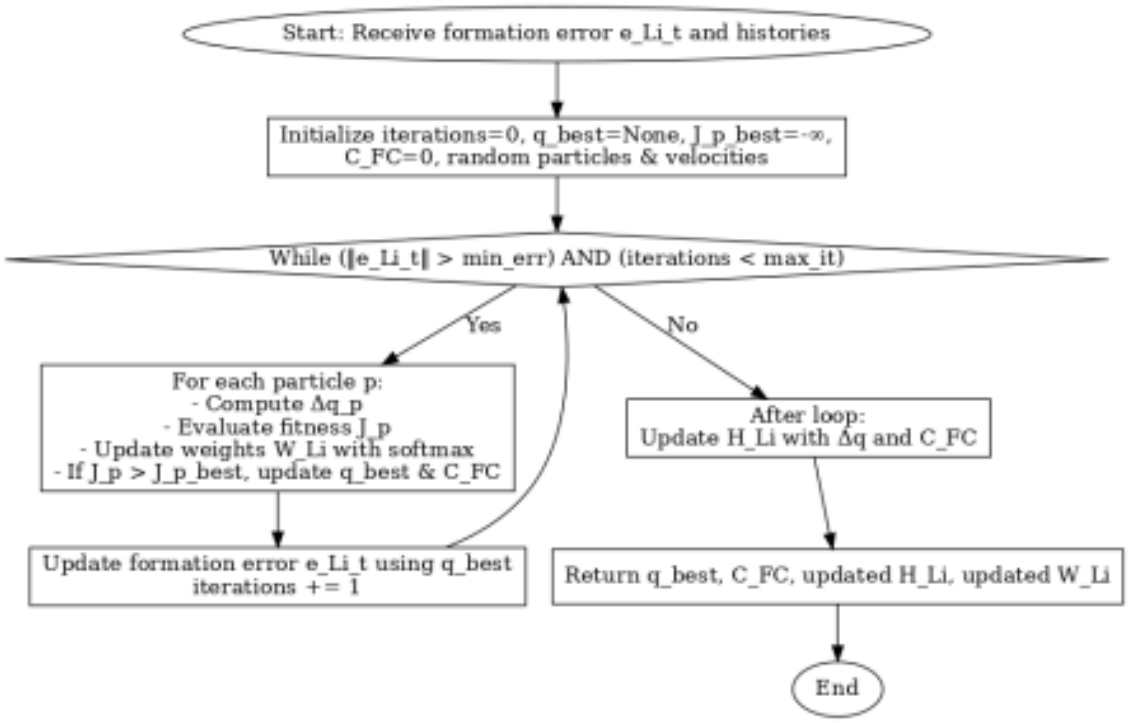


Figure 12 – Formation Control (FC) algorithm flowchart.

overhead, is compared in terms of energy efficiency and robustness to communication delays. ARCog-NET is benchmarked on key metrics such as formation error convergence time, energy consumption, and success rate in obstacle-rich scenarios.

3.2.4 Monocular Vision SLAM

Recaptulating, each UAV i at layer L[5] (Fog or Edge) gather a sequence of monocular camera frames captured from time $t = t_i$ to $t = t_n$, denoted as: ${}^{L_i}I = \{{}^{L_i}I_0, {}^{L_i}I_1, {}^{L_i}I_2, \ldots, {}^{L_i}I_{t_n}\}$. Each term $P^{\kappa}_{L_i}(x_\iota, y_\gamma, z_k)$ indicates a navigation point on trajectory ϱ_{L_i}. In addition to the navigation points, each UAV will carry a local map of the environment, composed of 3D landmarks ${}^{L_i}M = \{{}^{L_i}p_1, {}^{L_i}p_2, {}^{L_i}p_3, \ldots, {}^{L_i}p_t\}$ where each p_t is a 3D point in the environment reconstructed from visual data. The objective of ARCog-NET Monocular SLAM is to cognitively align the traditional navigation task points $n_p^{L_i}(t)$ with the 3D map ${}^{L_i}M$, so the traditional navigation algorithm of ARCog-NET can

[5]Note that when the notation ^{L_i} or ${}^{L'_j}$ appears before the variable symbol, it is related to computer vision. On the other hand, if these notations appear after the variable symbol, they are related to navigation.

both feed the visual odometry and mapping and use mapping information to improve itself in a virtuous cycle.

As UAV flies, the objective is to maximize the joint posterior probability of the camera (or the UAV itself) trajectory $n_p^{L_i}(t)$ and the map ${}^{L_i}M$ given the visual observations ${}^{L_i}I$, denoted $P(n_p^{L_i}(t), {}^{L_i}M \mid {}^{L_i}I)$. According to Bayes' theorem, this posterior can be factorized as:

$$P(n_p^{L_i}(t), {}^{L_i}M \mid {}^{L_i}I) = P({}^{L_i}M \mid n_p^{L_i}(t), {}^{L_i}I) \cdot P(n_p^{L_i}(t) \mid {}^{L_i}I), \tag{33}$$

where $P({}^{L_i}M \mid n_p^{L_i}(t), {}^{L_i}I)$ represents the posterior probability of the local environment map ${}^{L_i}M$ given the estimated UAV trajectory $n_p^{L_i}(t)$ and its image observations ${}^{L_i}I$ and $P(n_p^{L_i}(t) \mid {}^{L_i}I)$ is the posterior probability of the camera trajectory $n_p^{L_i}(t)$ conditioned only on the sequence of image observations ${}^{L_i}I$. Thus, Monocular Visual SLAM in ARCog-NET is naturally decomposed into two interrelated sub-problems: odometry and mapping, which together enable the simultaneous estimation of the agent's motion and the 3D structure of the observed environment.

To gain a clearer understanding of how the local and global 3D map are formed alongside navigation, it is first necessary to model the image formation of each UAV L_i. In the proposed test using the vision system of Tello UAV, spatial relationships are defined with respect to coordinate frames. The world coordinate frame is a fixed, global reference frame used to describe the absolute position and orientation of objects in a 3D environment. Its origin can be arbitrarily chosen; in this work, it is defined as the lower-left corner of the laboratory.

On the other hand, the camera coordinate frame is the local frame centered at the camera's optical center. It represents the position of objects relative to the camera. The transformation between the world and camera of the UAV L_i frames is defined by the camera's extrinsic parameters, represented as $[{}^{L_i}\mathcal{R} \mid {}^{L_i}\mathcal{T}]$, where ${}^{L_i}\mathcal{R} \in \mathbb{R}^{3\times3}$ is the rotation matrix and ${}^{L_i}\mathcal{T} \in \mathbb{R}^{3\times1}$ is the translation vector. This matrix allows conversion between world and camera coordinates, enabling localization, mapping, and visual odometry.

The same way, image coordinate frame is a 2D reference system that represents the position of points on the image plane. It is related to the camera's 3D coordinate frame through the intrinsic camera parameters, also known as the intrinsic matrix of UAV L_i ${}^{L_i}\mathcal{K}$. To project a 3D point from the world onto the image, a two-step transformation

is applied, i.e., first the extrinsic matrix $[^{L_i}\mathcal{R} \mid ^{L_i}\mathcal{T}]$ transforms the point from the world coordinate frame to the camera coordinate frame (3D to 3D) and then the intrinsic matrix $^{L_i}\mathcal{K}$ transforms the point from the camera frame to the image plane (3D to 2D). The complete projection process combines both matrices, enabling the camera to represent 3D world geometry as 2D image data, as presented in Figure 13.

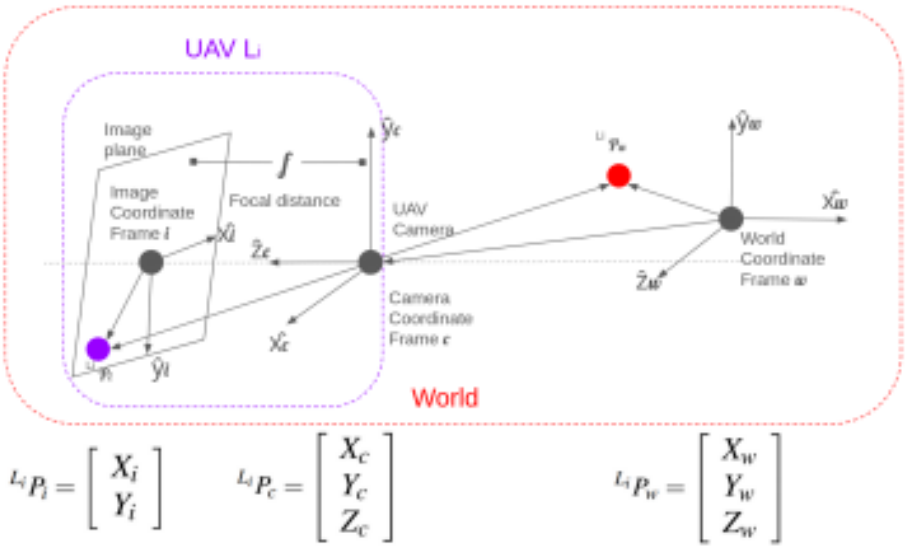


Figure 13 – World 3D point in monocular 2D image.

When a UAV camera captures an image, the 3D coordinates of a point in the world frame are first transformed into the camera coordinate frame using the extrinsic matrix, which includes both rotation and translation components, represented by:

$$^{L_i}P_c = [^{L_i}\mathcal{R} \mid ^{L_i}\mathcal{T}]\,^{L_i}P_w = ^{L_i}\mathcal{R}\,^{L_i}P_w + ^{L_i}\mathcal{T}. \tag{34}$$

Once in the camera frame, the point is projected onto the 2D image plane using the intrinsic matrix $^{L_i}\mathcal{K}$, which encodes the UAV camera's internal parameters $^{L_i}P_i = ^{L_i}\mathcal{K}\,^{L_i}P_c$.

While this suffices to explain how a 3D world point is represented in a local UAV image plane, the reconstruction of the 3D environment as a point cloud and the flight planning through this environment still needs additional steps. To better understand how this is done, it is important to notice that ARCog-NET navigation consists in generating 3D trajectory points for the UAV to fly stochastic exploration, and the trajectory is improved by reinforcement learning as both the subject and the swarm learns about mission objectives and the environment they're operating. Whenever an event is detected during flight, the system will recalculate the trajectories based on self knowledge or collective knowledge

decisions. An event may have various types, from a loss of communication link to an obstacle detection. The trajectory point increment due to event-driven updates is defined in Equation (22).

With the understanding of how navigation points are defined, it is possible to rollback to image mapping to understand how a monocular vision of a Tello UAV was programmed to generate 3D points locally, relating the trajectory points to these. Basically, as UAVs fly they generate local 3D world points in 2D images and share these points through the network, so it's possible to use feature matching[6] techniques on those points to recognize corresponding world coordinates and apply epipolar geometry solutions and triangulation to both recreate 3D geometry and align trajectories. To ARCog-NET, this comes naturally because if a higher decision weight is found at the weight tensor, automatically it is related to the better trajectory coverages in the knowledge base found at that moment, and this also will indicate that the framework already has stored points of that region to serve as base for 3D reconstruction and trajectory refinement.

For instance, if a UAV L_i encounters an obstacle (avoidance flight event) and has to recalculate its trajectory, it can search for a good decision weight in similar task, indicating optimal coverage solution previously registered by UAV L'_j. This also indicates that, in L'_j memory of image frames and points, some features that UAV L_i is "seeing" can be matched, as illustrated in Figure 14.

To extract the exact global P_w point from different images captured by different UAV, the following system must be solved:

$$\begin{cases} {}^{L_i}P_c = {}^{L_i}\mathcal{R}\,{}^{L'_j}P_c + {}^{L_i}\mathcal{T} & (E1) \\ {}^{L_i}P_c^{\top} E\,{}^{L'_j}P_c = 0 & (E2) \\ E = {}^{L_i}\mathcal{R}\,{}^{L_i}\mathcal{T} & (E3) \\ u_{L_i} = {}^{L_i}f_x \cdot \frac{{}^{L_i}X_i}{{}^{L_i}Z_c} + {}^{L_i}O_x & (E4)\,, \\ v_{L_i} = {}^{L_i}f_y \cdot \frac{{}^{L_i}Y_i}{{}^{L_i}Z_c} + {}^{L_i}O_y & (E5) \\ ({}^{L_i}\mathcal{K}^{-1})^{\top} F\,{}^{L'_j}\mathcal{K} = E & (E6) \\ {}^{L_i}P_c^{\top} F\,{}^{L'_j}X_i = 0 & (E7) \end{cases} \tag{35}$$

where $(E1)$ is the 3D to 3D transformation, $(E2)$ is the 3D epipolar constraint, $(E3)$

[6]Feature extraction and matching between image frames is implemented using OpenCV ORB and Brute-Force Matching libraries OpenCV (2025c,b).

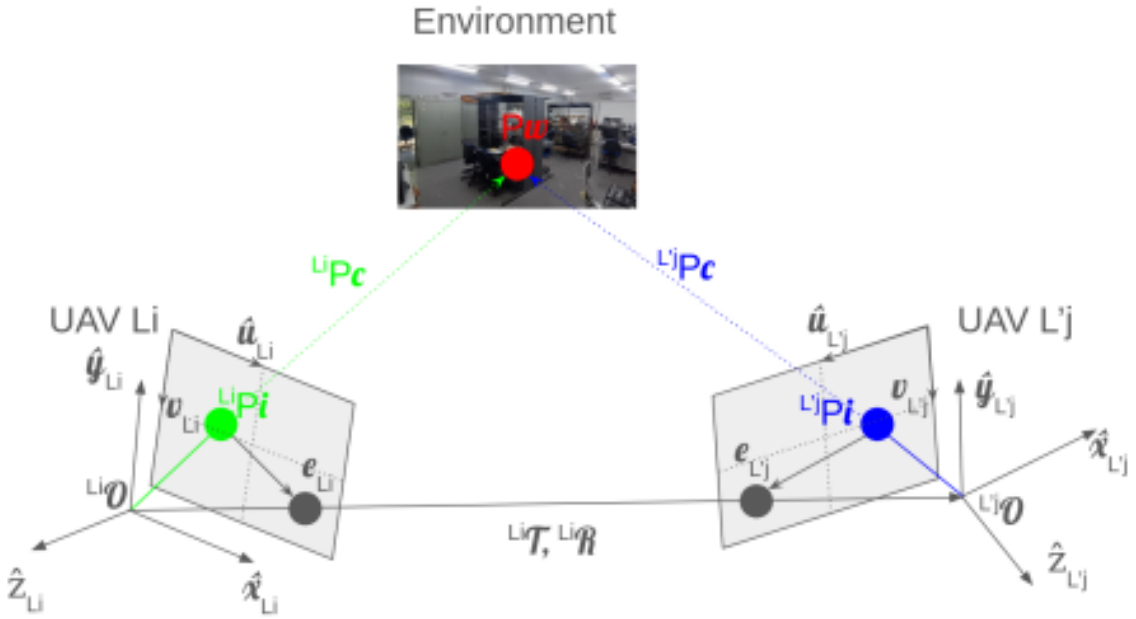


Figure 14 – Epipolar geometry from point matching.

represent the essential matrix E, $(E4)$ and $(E5)$ are equations for perspective projections in x and y directions on the images of UAV L_i, respectively, being ${}^{L_i}f_x$ and ${}^{L_i}f_y$ the focal lengths on same directions. Finally, $(E6)$ represents the relation between the fundamental matrix F and the essential matrix and $(E7)$ is the 2D epipolar constraint. In a subset of n point correspondences (obtained solving equations $E1$ and $E2$), the fundamental matrices can be found solving:

$$\begin{bmatrix} u_{1,L_i}u_{1,L'_j} & v_{1,L'_j}u_{1,L_i} & u_{1,L_i} & u_{1,L'_j}v_{1,L_i} & v_{1,L_i} & u_{1,L'_j} & v_{1,L'_j} & 1 \\ u_{2,L_i}u_{2,L'_j} & v_{2,L'_j}u_{2,L_i} & u_{2,L_i} & u_{2,L'_j}v_{2,L_i} & v_{2,L_i} & u_{2,L'_j} & v_{2,L'_j} & 1 \\ u_{3,L_i}u_{3,L'_j} & v_{3,L'_j}u_{3,L_i} & u_{3,L_i} & u_{3,L'_j}v_{3,L_i} & v_{3,L_i} & u_{3,L'_j} & v_{3,L'_j} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ u_{n,L_i}u_{n,L'_j} & v_{n,L'_j}u_{n,L_i} & u_{n,L_i} & u_{n,L'_j}v_{n,L_i} & v_{n,L_i} & u_{n,L'_j} & v_{n,L'_j} & 1 \end{bmatrix} \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ \vdots \\ F_n \end{bmatrix} = 0. \tag{36}$$

By using OpenCV's libraries to solve the system and find the fundamental matrices OpenCV (2025a), it's possible to find all other parameters through the system of Equations (35) and therefore obtain the UAV (camera) position related to the matching points from image frames history. Knowing the UAV's position and its motion through trajectories, it is possible to estimate the spatial position of the matching points in the local 3D world coordinate frame, as illustrated in Figure 15.

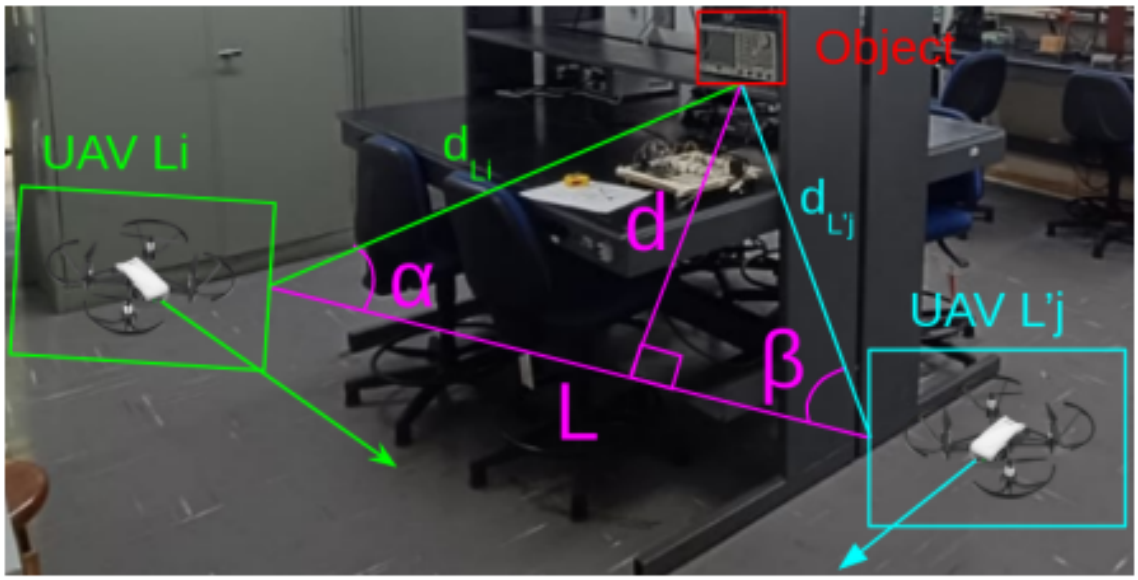


Figure 15 – Triangulation to form the 3D point cloud from matching image points and camera parameters.

Knowing all presented camera calibration parameters and matching points, suppose (u_{L_i}, v_{L_i}) be the coordinates of a point in the image frame of the UAV L_i and $(u_{L'_j}, v_{L'_j})$ the corresponding point in the image frame of the UAV L'_j. The projection matrices for the two camera poses between the initial time t_i and the next captured frame time $t + \tau$ are defined as ${}^{L_i}P_c = {}^{L_i}\mathcal{K}[{}^{L_i}I \mid 0]$, ${}^{L'_j}P_c = {}^{L'_j}\mathcal{K}[{}^{L'_j}\mathcal{R} \mid {}^{L'_j}\mathcal{T}]$. The relationship between a 3D point P_w and its 2D projection P_i in an image is given by the camera projection model $P_i = P_c P_w$. Thus, the projections in both images are ${}^{L_i}P_i = {}^{L_i}P_c^{L_i}P_w$, ${}^{L'_j}P_i = {}^{L'_j}P_c^{L'_j}P_w$. Since the cross product of a vector with itself is zero, and a valid projection must lie along the projected ray, it's possible to derive the following constraints ${}^{L_i}P_i \times ({}^{L_i}P_c^{L_i}P_w) = 0$, ${}^{L'_j}P_i \times ({}^{L'_j}P_c^{L'_j}P_w) = 0$.

These cross-product constraints ensure that the point P_w lies along the correct epipolar lines in both images. Expanding these equations yields a system of linear constraints, typically assembled into a matrix A, such that $AP_w = 0$. The solution to this homogeneous system is found using the Singular Value Decomposition (SVD) of A, which provides the least-squares estimate of the 3D point P_w in homogeneous coordinates.

Notice that this is a local point, as it is formed by a UAV i in layer L. Usually, edge agents will fly more, collecting more points and sending them to their fog coordinators. The fog coordinators usually process the mathematics to group the 3D points of their cluster, generating the cluster 3D point cloud. Finally, the server receives all data from the network and can generate the global 3D reconstruction, feeding back the network with the global

map, which is aligned to the trajectories of each component, allowing the system to fly while mapping the environment.

This combination allows ARCog-NET to improve trajectories as the mission unfolds and prevent collisions or other situations that will menace UAV flight. When events demand trajectory re-planning, ARCog-NET invokes its internal navigation algorithm, which leverages past experience and global mapping to optimize future choices. Previously computed trajectories are revisited, and if suitable, reused, which reduces the computational burden.

To evaluate the adaptive capability of the proposed architecture when exposed to dynamic disturbances, a metric named the Adaptive Response Index (ARI) was developed. This index quantifies how effectively the system reacts to and recovers from events described before. The ARI varies within the interval $[0, 1]$, where values close to one indicate an excellent response characterized by fast reaction, limited degradation, and stable recovery, while values near zero indicate poor adaptability.

For each disturbance event starting at time t_0, the evaluation considers several measurable components that describe the dynamic response of the system. The first component is the detection delay, which represents the time required for the controller to recognize that a disturbance has occurred or that any monitored metric has exceeded its tolerance band. It is defined as:

$$\tau_{\text{det}} = \min\{\, t \geq t_0 : |m_k(t) - \mu_k| > \varepsilon_k \text{ for any } k \,\} - t_0, \tag{37}$$

where $m_k(t)$ is the observed value of metric k, μ_k is its steady-state reference, and ε_k is the acceptable tolerance around that value. Once a disturbance is detected, the magnitude of the degradation is assessed through the maximum normalized deviation across all monitored variables, expressed as:

$$\Delta_{\max} = \max_{k} \max_{t \in [t_0, t_1]} \left(\frac{|m_k(t) - \mu_k|}{\varepsilon_k} \right). \tag{38}$$

After the disturbance, the system begins to stabilize, and the time required for it to return and remain within the nominal performance band is represented by the recovery time, defined as:

$$\tau_{\text{rec}} = \min \left\{ t \geq t_0 : |m_k(u) - \mu_k| \leq \varepsilon_k,\ \forall k,\ \forall u \in [t, t + \theta] \right\} - t_0, \tag{39}$$

where θ corresponds to the minimum period of sustained stability required to consider the

recovery complete. The intensity and duration of the degradation are jointly represented by the degradation area, given by:

$$\mathrm{ID} = \sum_{k} \int_{t_0}^{t_1} \left[\frac{|m_k(t) - \mu_k| - \varepsilon_k}{\varepsilon_k} \right]_+ dt, \tag{40}$$

where $[x]_+ = \max(x, 0)$ denotes the positive part operator. This formulation aggregates both the amplitude and persistence of performance deviations across all monitored quantities. In addition to these behavioral indicators, a reconfiguration cost term is introduced to account for the system's internal effort to adapt during the disturbance. This cost is defined as:

$$C_{\mathrm{rec}} = c_R N_{\mathrm{roles}} + c_E E_{\mathrm{sup}} + c_Q Q_{\mathrm{fog}}, \tag{41}$$

where N_{roles} represents the number of edge/fog role transitions, E_{sup} is the additional energy consumption in watt-hours, and Q_{fog} is the average backlog increase observed in the fog layer during adaptation. The coefficients c_R, c_E, and c_Q assign relative importance to each cost component depending on mission constraints. In practice, the end of observation t_1 can be defined as the end of the experimental window or as a fixed period after the disturbance, typically 60 s, and the stability threshold θ is often between 2 and 5 s. Each component is normalized by its respective reference limit to produce dimensionless quantities, according to:

$$\hat{\tau}_{\mathrm{det}} = \frac{\tau_{\mathrm{det}}}{\tau_{\max}}, \qquad \hat{\tau}_{\mathrm{rec}} = \frac{\tau_{\mathrm{rec}}}{T_{\max}}, \qquad \widehat{\mathrm{ID}} = \frac{\mathrm{ID}}{\mathrm{ID}_{\max}}, \qquad \hat{C}_{\mathrm{rec}} = \frac{C_{\mathrm{rec}}}{C_{\max}}. \tag{42}$$

The overall adaptive response for a given event e is then computed by combining these normalized terms into a single scalar indicator, as shown in:

$$\mathrm{ARI}_e = 1 - \left(w_1 \hat{\tau}_{\mathrm{det}} + w_2 \hat{\tau}_{\mathrm{rec}} + w_3 \widehat{\mathrm{ID}} + w_4 \hat{C}_{\mathrm{rec}} \right), \qquad \mathrm{ARI}_e \in [0, 1], \tag{43}$$

where $w_1 + w_2 + w_3 + w_4 = 1$. The weights w_i define the relative influence of detection speed, recovery duration, degradation severity, and adaptation cost, respectively, according to the priorities of the evaluated mission. A higher ARI indicates a system capable of quickly identifying perturbations, limiting the magnitude of performance losses, and recovering

stability with minimal internal overhead. When multiple disturbance events occur within an experiment, an aggregated value can be obtained by computing the mean index across all events, expressed as:

$$\overline{\mathrm{ARI}} = \frac{1}{E} \sum_{e=1}^{E} \mathrm{ARI}_e, \tag{44}$$

and complementary indicators such as the 5th, 50th, and 95th percentiles (ARI_{p5}, ARI_{p50}, ARI_{p95}) can be used to evaluate consistency across repeated trials. In the experimental setup involving six DJI Ryze Tello UAVs, tolerance bands included a maximum deviation of $0.05\,\mathrm{m}$ for absolute trajectory error (ATE), $40\,\mathrm{ms}$ for network latency, and $0.10\,\mathrm{m}$ for formation error. The selected weights were $w_1 = 0.2$, $w_2 = 0.4$, $w_3 = 0.3$, and $w_4 = 0.1$, emphasizing recovery stability over instantaneous response. This metric provides a systematic means of comparing adaptive performance among different ARCog-NET configurations and quantifying, in a consolidated manner, the system's robustness under real dynamic indoor conditions.

3.3- Simulation framework

Crafting a simulation architecture for ARCog-NET validation aims to significantly enhance the development and testing of its usage in UAV swarm systems. The usage of Robotics Operating System (ROS) (Quigley et al., 2009) and Gazebo (Gazebo, 2021) to provide a flexible framework for writing robot software and performing simulations has become a standard for researchers and developers working on autonomous systems. By incorporating Gazebo, a high-fidelity 3D robotics simulator, this architecture achieves realistic modeling of UAV dynamics and environmental interactions. When these capabilities are augmented with the MAVLink ROS (MAVROS) package (ROS, 2022), a ROS-based library that enables Micro Air Vehicle Link (MAVLink) protocol communication (MAVLINK, 2022) between ROS and the ArduPilot/PX4 platforms (PX4, 2022), the simulation gains the ability to closely replicate real-world UAV behaviors and control mechanisms.

Integrating ROS and Gazebo provides an environment where the physical properties of UAVs and their sensors can be accurately simulated. This allows the emulated

edge layer to process data at the source, the fog layer to manage latency-sensitive computations, and the cloud layer to handle complex analytics and aggregate data storage. The MAVROS package bridges the gap between the simulated UAVs in Gazebo and the actual Flight Controller Unit (FCU) firmware, ensuring that commands, controls, and data flow adhere to the same protocols used in live systems. This congruence is crucial for testing the scalability of UAV swarm operations and their respective data handling across the network layers.

Within this simulated setup, the MAVROS package is invaluable for conducting advanced experiments involving autonomous navigation, collision avoidance, and swarm coordination. It enables the simulation to extend beyond the confines of a virtual environment, allowing for a seamless transition to field trials with real UAVs. Developers can fine-tune the system's responsiveness and efficiency by simulating various network configurations and workloads. Incorporating MAVROS into the architecture also facilitates the exploration of HITL dynamics, providing a comprehensive platform for testing both the technological and human-interactive aspects of UAV swarm management. This multifaceted simulation environment is a critical tool for iterating on complex ARCog-NET network, fostering advancements in UAV swarm technologies.

3.3.1 Simulation architecture

Based on the algorithms running at each node on respective layers presented in previous topics, the data structures formatting between nodes, and the theoretical model of the network's cognition, the data flow within ARCog-NET architecture is one of the most important parameters to define the framework and practical operation of this tool. As previously known, the network is segmented into three kinds of primary nodes: the agent, coordinator, and server, each serving distinct functions and interconnected through data transmission channels. Although ARCog-NET can be used for developing general aerial robot swarm applications, this research presents the usage of the MAVLink as a standard transmission protocol and also for exchanging internal data between sensors and other modules. Due to its versatility for integration with real-life hardware (Ardupilot, 2021a,b), lots of sensor plugin packages, and ease of modeling and integrating new sensors with

MAVROS, this can be considered as a contribution of the architecture presented on this research. For understanding ARCog-NET simulation methodology, it is crucial to know the framework's simplified dataflow represented in Figure 16.

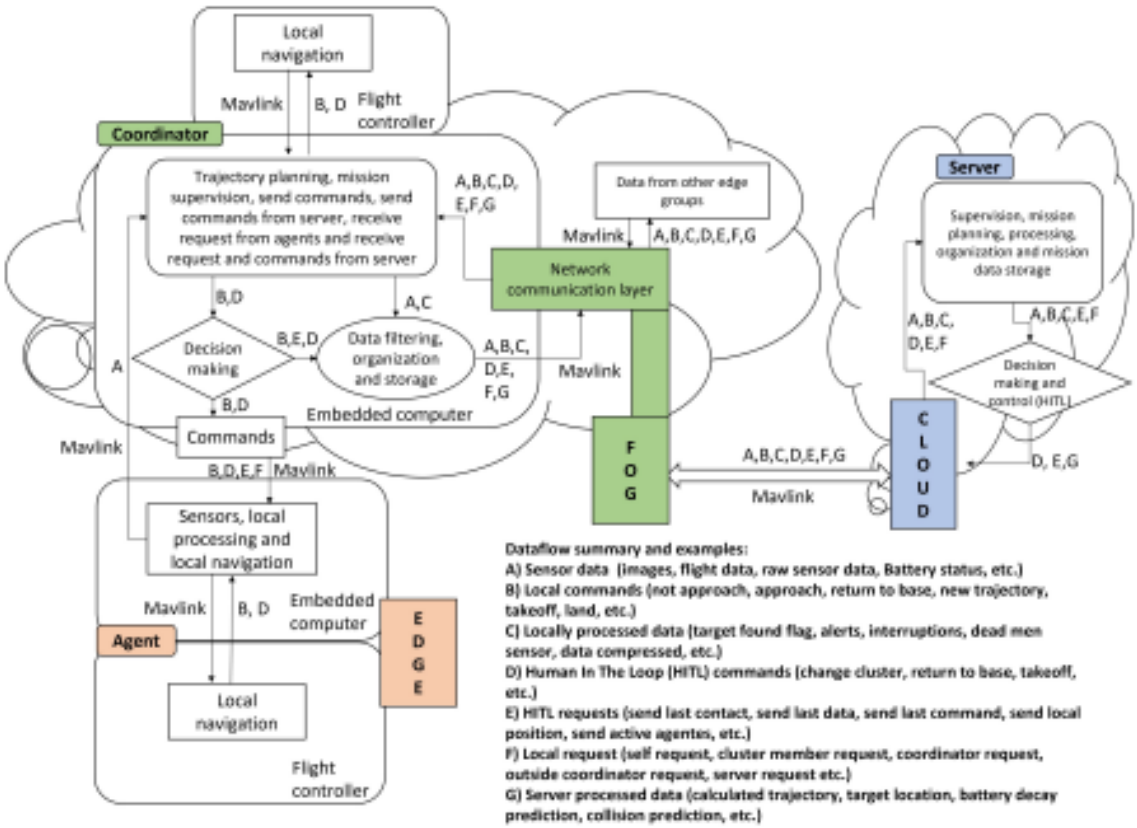


Figure 16 – ARCog-NET simplified dataflow representation.

Data flows seamlessly through this architecture, with the MAVLink protocol enabling communication across the network or between components. This ensures that sensor data, navigation commands, and other critical information are shared efficiently between the Agents, the Coordinators, and the Server. For instance, data generated by the Agents' sensors is sent to the Coordinator, who then decides whether to act on the data locally or send it through the network to the Server for further analysis. The flowchart also outlines various data types and their flow paths. For example, sensor data (type A), local commands (type B), and HITL commands (type E) are channeled through different paths to ensure timely and appropriate responses within the network. The system's design facilitates robust and flexible UAV operations, enabling online responses to local events and integrating human oversight through the cloud for strategic decision-making.

The ARCog-NET is designed to have an FCU dedicated only to flight control,

peripherals, and avionic data gathering, and all the data processing, decision-making, communication and complementary operations that demand computational load may be executed on embedded boards like Raspberry Pi (Pi, 2024) or Galileo (Intel, 2024). These offboard computers will command FCU through serial communication while executing the algorithms described in this section. One of the advantages of this topology is also the use of ROS packages that may run on the offboard controllers with their simulation already tested on Gazebo. The information may be exchanged between nodes and layers through free-to-air modulated signals using radio transmitter/receiver pairs. The cloud layer must have improved communication power, as it doesn't have hardware component constraints (since it is expected that it is a ground-based control unit) and may be farther from the cloud and edge UAV devices. Figure 17 simplifies the basic hardware setup that this research assumes the robots use.

In the simulation, Agents (edge) represent individual UAVs equipped with sensors and embedded computers. They handle local processing tasks like navigation and sensor data analysis. The processed data can include images, flight data, and sensor outputs, which are then sent to the Coordinator (fog) for further decision-making or directly to the Server for storage and higher-level analysis. At the core of this architecture, the Coordinator is responsible for trajectory planning and mission supervision. It sends and receives commands to and from the Server (cloud) while also handling requests from Agents. This node processes data such as sensor inputs and navigation commands, making decisions that are then communicated to both the Agents and the Server for cohesive mission execution. The Server, operating within the cloud layer, oversees the entire mission, providing additional processing power for data organization and long-term storage. It also plays a crucial role in decision-making, particularly with HITL commands, where a human operator can intervene and control the swarm's actions, ensuring mission flexibility and adaptability to changing conditions. Finally, Figure 18 presents the basic simulated components within each layer.

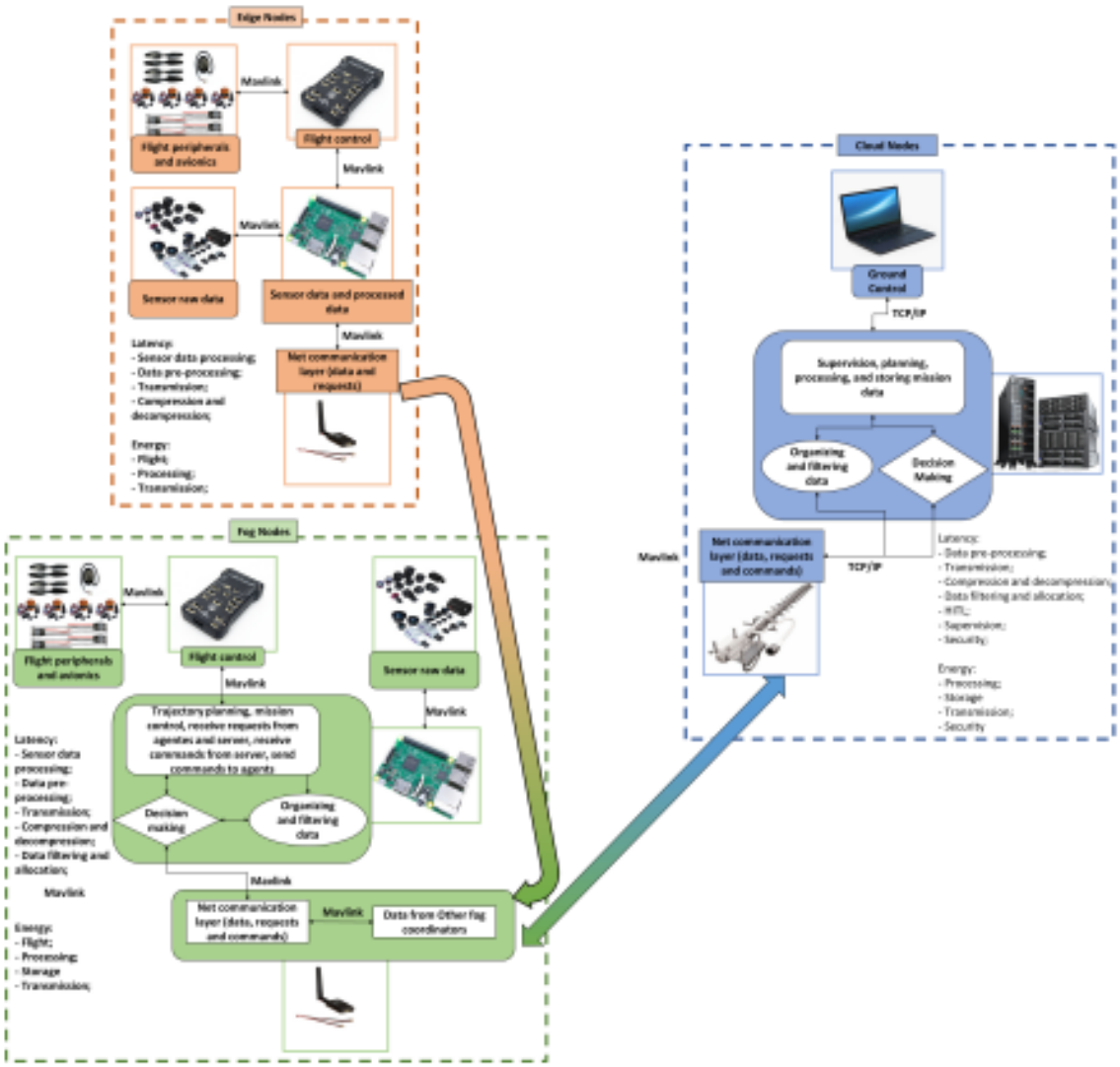


Figure 17 – Standard hardware representation of ARCog-NET.

3.3.2 Multi-UAV simulation with ROS and Gazebo

The idea adopted for developing the UAVs simulation was to launch separated agents with the PX4 FCU and to develop custom nodes representing offboard controllers. These offboard nodes will communicate to their respective FCU through MAVLink protocol (in this case, MAVROS) to receive data from sensors, alarms, status, position, velocity, and any other variable that is important for the operational controller of each robot. In real life, this offboard controller may be a companion computer like a Raspberry Pi attached to the UAV, and the communication between the UAV and the companion computer may happen through serial communication via USB or the MAVLink telemetry radio (one attached to the

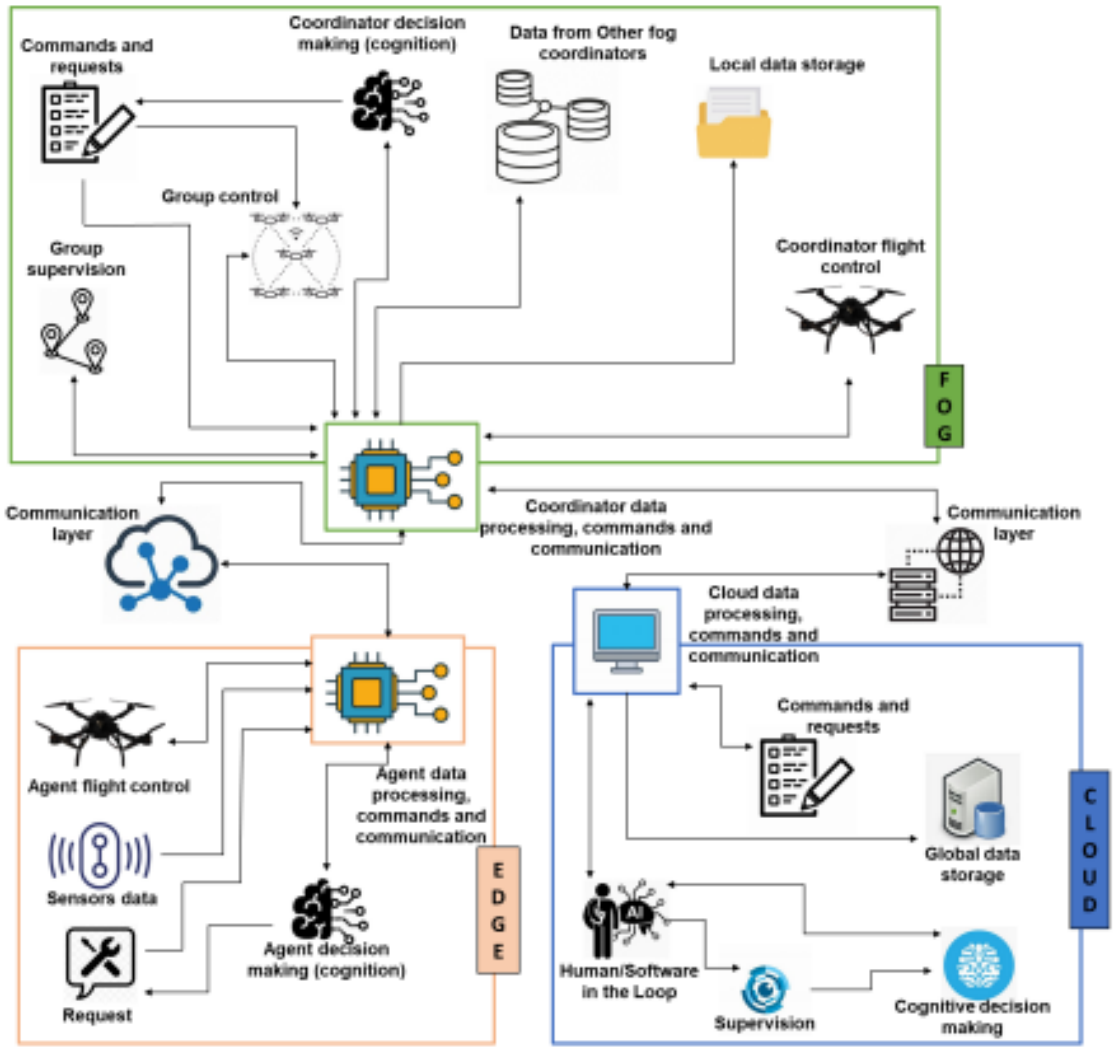


Figure 18 – ARCog-NET basic components.

FCU and one attached to the embedded computer). In the offboard controller, packages for controlling its respective UAV may be developed, but also for communicating with the simulated network and for using a different sensor that can't be embedded in the UAV's FCU. For example, this sensor could be an RGB camera, which in simulation is a Gazebo external plugin streaming the image "seen" by each UAV through a pipeline using the Python GStreamer library and OpenCV, transformed to a camera node that communicates to each offboard node. The active ROS nodes diagram for a 2 UAV group (one edge agent and one fog coordinator) developed under ARCog-NET is presented in Figure 19, and in Attachment 6 there is a representation of ROS nodes in a simulation with a cloud server, 8 edge agents divided into 2 groups and 2 fog coordinators for each group.

Besides the sensor data, an example of data processing is the image compression autoencoder model used for image compression and decompression in simulation. It is

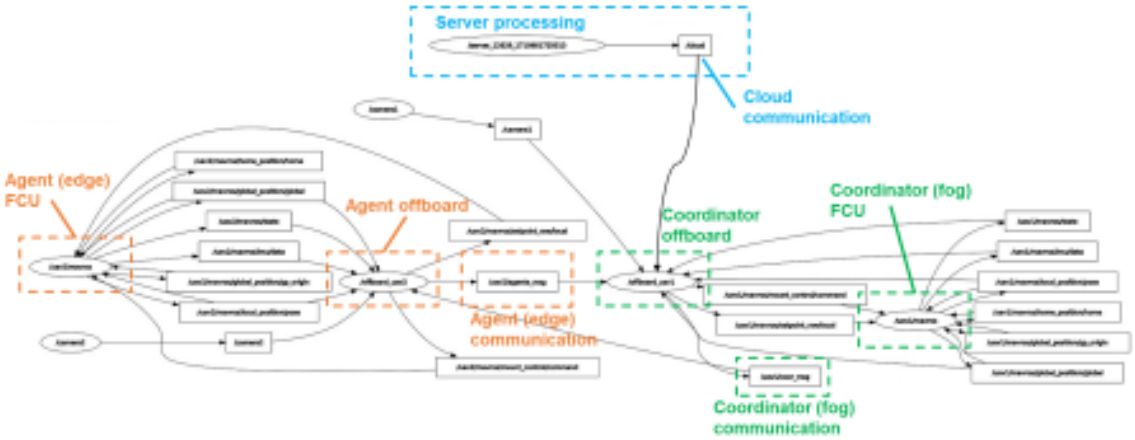


Figure 19 – Active ROS nodes and topics for 2 UAV and a serve in simulation.

also integrated into the offboard nodes, taking the camera's image from the camera nodes and transmitting it to the network. In real life, data communication may be addressed with radio transceivers, WiFi, ZigBee, or other wireless communication technology associated with the companion computer. The simulated UAV was a Typhoon H480 model, and the simulation was performed with groups of 1 to 10 vehicles with Graphic User Interface (GUI), 10 times each. However, without running the Gazebo GUI, simulations were performed from 1 to 1000 vehicles connected to ARCog-NET in order to validate scalability. Figure 20 presents the simulated UAVs in the Gazebo environment.



Figure 20 – Swarm of 10 Typhoon H480 in simulation.

The next step for creating the simulation in ROS and Gazebo for testing the ARCog-NET packages and behavior was to create generic missions to receive applications that could be assigned to a UAV group. Four missions were designed for that matter, being the first scenario a representation of a vertical wind turbine field, where the UAVs should take

off from a land base and fly through wind turbines for predictive maintenance surveillance using coverage path planning algorithms. The simulation environment developed for this first application is presented in Figure 21, and here the major concern is the autonomous navigation and interference between communications. The second simulation background represents a situation of an offshore oil exploration environment in which the UAVs will follow programmed paths for environmental surveillance (search for oil spills, for example). The UAV swarm should take off from the oil production facility's helideck and send its camera images to the control station. First, to simulate the environment in which the mission will take place, an oil rig and a shuttle tanker vessel were designed with Blender 3D modeling software, Gazebo packages for designing the sky, clouds, sea, and ODE plugins for wind and sea parameters, the world on which the simulation took place was created. Figure 22 presents the results.

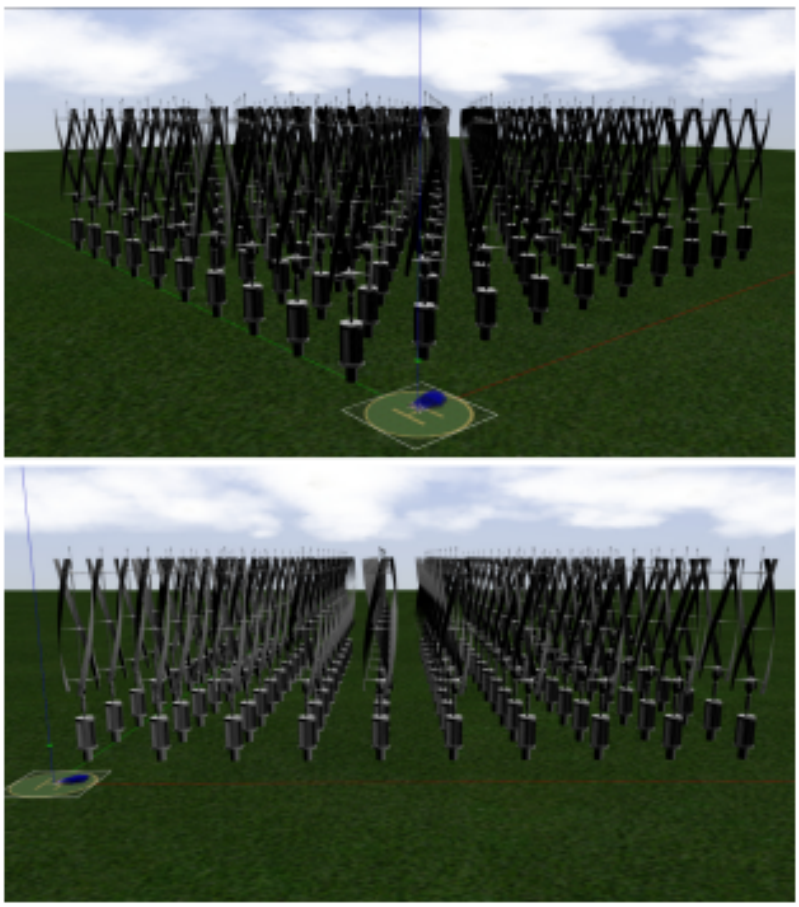


Figure 21 – World created in Gazebo for simulating autonomous predictive maintenance on a vertical wind field mission.

The third application consists of using ARCog-NET robots as internet routers for a communication failure emergency in a neighborhood scenario. The group should take

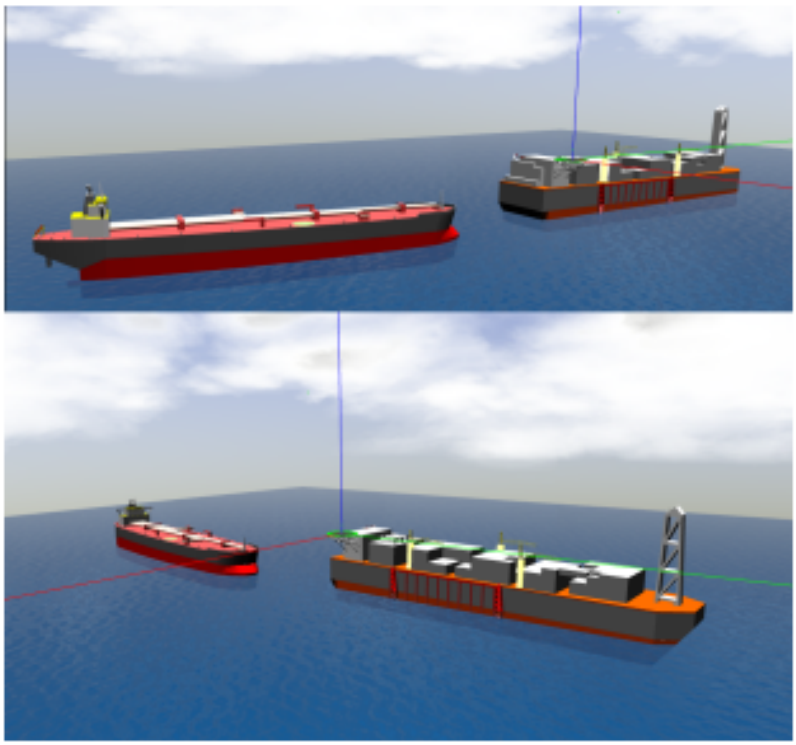


Figure 22 – World created in Gazebo for simulating the image transmission system on an offshore mission.

off from a land base and position itself to provide maximum signal coverage in a city environment given aleatory demand generated by an algorithm. Concerns about this mission are mainly the energy of each UAV for keeping position and transmitting signals, and the decision should be mostly based on the robots' positioning and how to switch each robot at the right time for charging while keeping the internet signals available for the city. This third simulation sandbox is presented in Figure 23. Finally, the last simulation represents an industrial plant undergoing an emergency situation (a gas leakage, for example) to evaluate the indoor behavior of a group of UAVs using

This integration of ROS and Gazebo simulation platforms is pivotal for testing and validating ARCog-NET in different scenarios, providing essential environments to emulate complex interactions within controlled settings. These simulation tools are instrumental in modeling the behavior of UAV swarms and robotic systems, allowing researchers and engineers to meticulously analyze the performance of distributed computing tasks across edge, fog, and cloud layers. With these realistic simulation scenarios, the purpose is to facilitate the examination of data flow, latency, throughput, and other critical network metrics (which will be addressed in the section ahead) in a virtual yet plausible context. This preemptive scrutiny enables the identification and rectification of potential issues, the

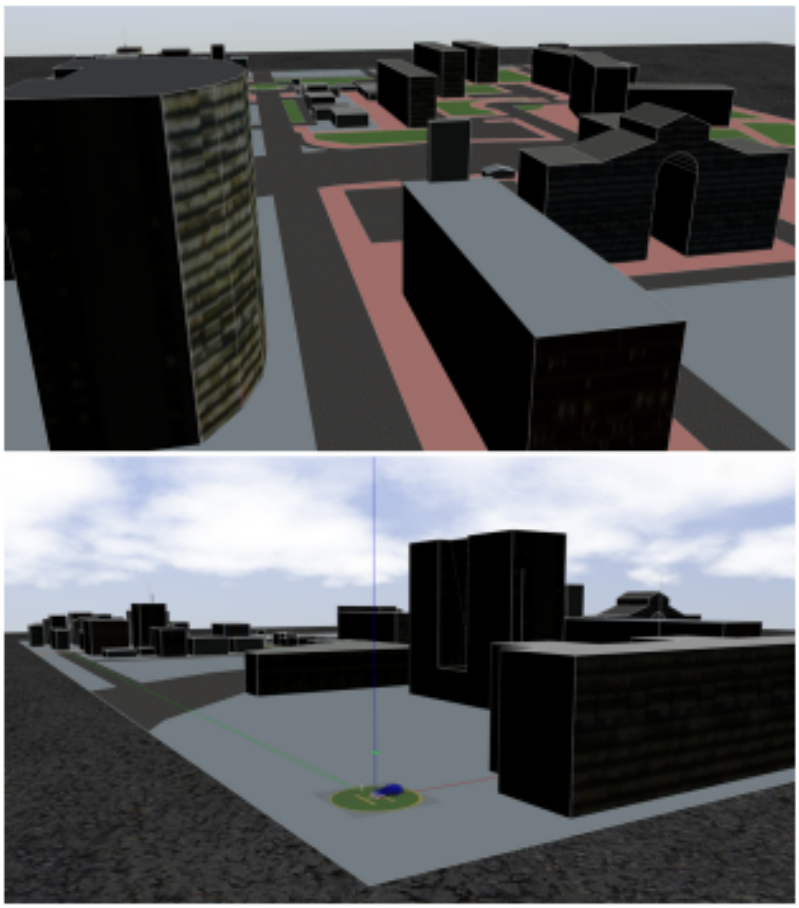


Figure 23 – World created in Gazebo for simulating using the UAV swarm as WiFi routers for communication emergencies.

optimization of network parameters, and the prediction of system behavior under various operational conditions without the risks or costs associated with real-world deployment. Consequently, ROS and Gazebo are invaluable assets in advancing ARCog-NET, contributing significantly to the development of robust, efficient, and scalable solutions in the field of autonomous systems.

All development of ARCog-NET custom packages was made with Python 3.8.10 language, using mainly the libraries tensorflow, numpy, scipy, scikit learn, opencv, rospy and MAVROS. The robot packages used ROS Noetic and the simulations ran in the Gazebo 11 environment and plugins. The main machine used for programming and simulation was a Lenovo IdeaPad Gaming 3i, with the following specs: 4,4GHz Intel Core i5 processor, 16 GB DDR4 3200MHz RAM, NVIDIA GeForce RTX 3050 4GB video board, 512GB SSD storage and Ubuntu 20.04.6 LTS operating system. The simulated UAVs are all Yuneec Typhoon H480 hexacopters with PixHawk Px4 FCU, Raspberry 5 as offboard embedded computer and 4G telemetry and communication modules, with the

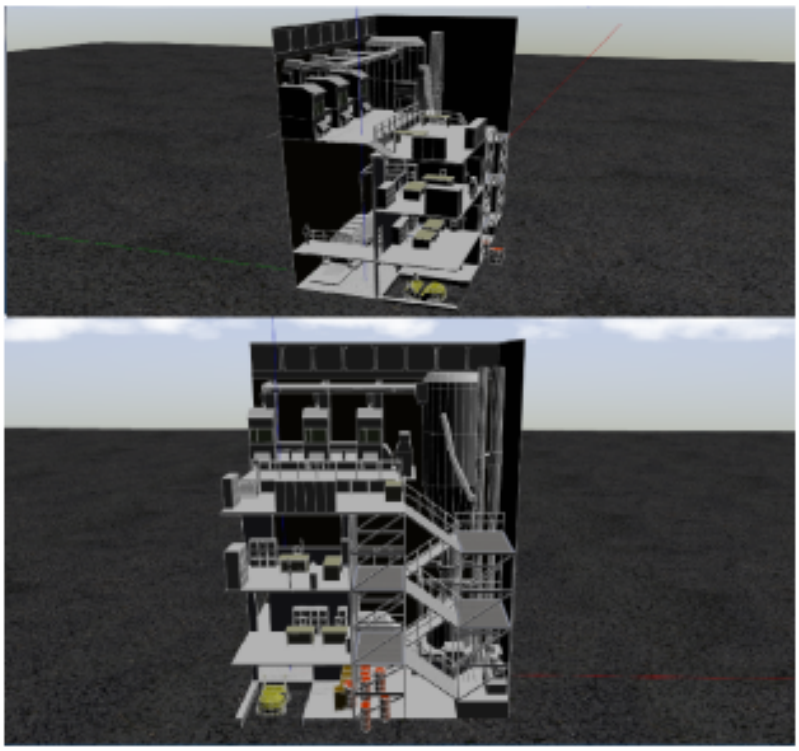


Figure 24 – World created in Gazebo for simulating using the UAV swarm in search and rescue indoor missions.

specs presented in Table 5. The Table 4 shows the simplified hardware power consumption considered in the simulations. The performance of each mission was evaluated alongside the network parameters described in next topics.

Table 4 – Power Consumption of Different Hardware Components.

Hardware	Power
Embedded Computer	27 W
UAV	120 Wh
Radios	1 W
Mobile Communication Modules and Antenna	100 W
Cloud	65 W

3.4- Laboratory SLAM test

To validate the model, an experiment was performed, and it consisted of using 6 DJI Ryze Tello UAVs connected in the ARCog-NET framework to map the Control and

Table 5 – Communication and Processing Power Specifications.

Device	Pixhawk PX4 FCU	Raspberry Pi 5	4G Telemetry
Processor	STM32F427, Cortex M4, 168 MHz	Broadcom BCM2711, Quad core Cortex-A72 (ARM v8), 1.5GHz	N/A
Memory	2 MB Flash, 256 KB RAM	4 GB, 8 GB LPDDR4-3200 SDRAM	N/A
Comm. Interfaces	UART, SPI, I2C, CAN, USB	Gigabit Ethernet, WiFi 5 (802.11ac), Bluetooth 5.0, BLE	LTE, HSPA+
Data Rates	N/A	Ethernet: up to 1 Gbps, WiFi: up to 866.7 Mbps	LTE: up to 150 Mbps (Download), up to 50 Mbps (Upload)
Frequency Bands	N/A	2.4 GHz, 5.0 GHz bands	Multiple bands depending on carrier and region
Additional Features	Advanced sensor integration for navigation, Multiple safety features including triple redundant IMU	Extensive GPIO and peripheral support, 4K video support	High mobility support, designed for IoT applications

Automation Laboratory (LACEA) of the Federal Center for Technological Education Celso Suckow da Fonseca (CEFET/RJ). The environment is composed of 5 environments, being the experimentation area, the workstations area and 3 personal rooms. The UAV's initial information was only the (x, y) coordinate of the deploy point of each UAV in relation to the lowest left corner of the laboratory as the origin of the world coordinate frame. To navigate and map the laboratory, the swarm used only the cognitive architecture for navigation and communication provided by ARCog-NET and the new implementation of the image SLAM, updating position based on Equation (24), aligning trajectory points to world point cloud P_w and updating it on time, velocity and heading angle with Equation (23).

The Tellos were connected to a NAT (Network Address Translator), each using a dedicated Wi-Fi card, which directed each UAV's UDP communication to a respective offboard ROS node running on a laptop connected to the NAT. Using the Tello Standard Development Kit (SDK) (Ryze Robotics, 2025) to read basic data from the UAVs, which were captured by the ROS offboard nodes and processed within ARCog-NET, sending back flight commands to the UAVs also through the SDK. The ROS offboard nodes communicate with each other within the edge-fog-cloud structure of ARCog-NET through MAVLINK (MAVLINK, 2022) protocol over MAVROS (ROS, 2022) package. The 6 offboards are divided into 2 groups of 3 Tellos. Each group had one fog coordinator and two edge

agents. During the mission, the edge and fog roles switched between offboards (or Tello) depending on the UAV status and mission tasks. Only the coordinator (fog) communicates with the server (cloud), also running as a single ROS node. The experiment setup is presented in Figure 25.

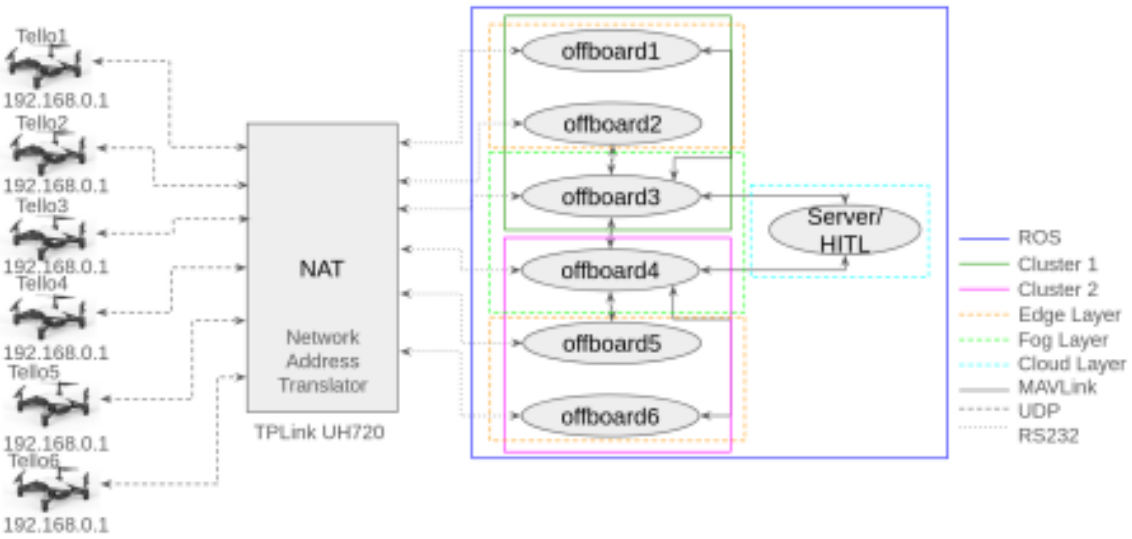


Figure 25 – Laboratory test setup with 6 DJI Ryze Tello connected to ARCog-NET.

To evaluate the performance of the ARCog-NET framework integrated with Monocular Visual SLAM in a multi-UAV indoor mapping task, several quantitative and qualitative metrics were employed. SLAM performance was assessed through Absolute Trajectory Error (ATE), Relative Pose Error (RPE), and reprojection error, ensuring trajectory consistency and map fidelity. The quality of the 3D reconstruction was also evaluated in terms of completeness and point cloud density. For the cognitive and networked aspects, metrics included trajectory coverage score, decision weight convergence, knowledge reuse rate, and replanning frequency, reflecting the system's learning and adaptability. Network performance was analyzed through decision latency across Edge-Fog-Cloud layers, bandwidth usage, and computational load distribution. Additionally, robustness was measured by the rate of failure recovery and the frequency of UAV role-switching between edge and fog nodes. These metrics collectively enabled a comprehensive assessment of the system's mapping accuracy, decision intelligence, and communication efficiency. The next session will explore the obtained results and discussions originating from the laboratory tests.

3.5- Network evaluation

Evaluating the performance of an EFC network, especially one designed to orchestrate the UAV swarms, necessitates a comprehensive set of metrics that can accurately reflect the system's efficiency, reliability, and responsiveness. When assessing such a network, one must consider a variety of evaluation metrics. Latency, for example, is a critical metric that measures the time a command or data packet takes to traverse from one point in the network to another, directly impacting the timeliness of the UAVs responses to control commands or environmental stimuli. Throughput, another essential metric, gauges the volume of data successfully processed over the network within a given time frame, indicative of the network's capacity to handle dense information streams. Moreover, the reliability metric assesses the network's ability to maintain consistent performance under varying conditions, while scalability evaluates how well the network can adapt to increasing numbers of UAVs without degradation in performance. Other key metrics include energy efficiency, which is particularly crucial for battery-dependent UAVs, and fault tolerance, the network's ability to sustain operations despite node or communication failures. Each of these metrics provides valuable insight into different aspects of the network's overall performance, and together they form a multidimensional framework for evaluating the efficacy of an edge-fog-cloud architecture in supporting advanced UAV swarm operations.

3.5.1 Latency

Latency within an edge-fog-cloud network encompasses several forms of delay that data packets incur throughout their journey. Propagation delay, represented as T_{p}, is the time required for a signal to traverse the distance between sender and receiver. It is a function of the distance $S_{L'_j,L_i}$, the speed of signal propagation s, and λ_{L_i} is the number of communication links between the two nodes, given by the equation:

$$T_p = \frac{S_{L'_j,L_i}}{s \cdot \lambda_{L'_j,L_i}}. \tag{45}$$

Transmission delay denoted as T_t, pertains to the time needed to push all the bits of a message into the network. It can be calculated using:

$$T_t = \frac{M_L}{R_{L'_j,L_i}}, \tag{46}$$

where M_L is the message length in bits, and $R_{L'_j,L_i}$ is the transmission rate of the link. Processing delay, or T_{proc}, is the time network devices take to process a packet and determine its forwarding path. Though often considered constant, it varies with computational load and routing complexity. Queuing delay, T_q, the time a packet spends in the queue, is notably variable, dependent on network traffic levels, and is a challenge to model due to its fluctuating nature. In UAV swarm systems, sensor-to-edge delay is critical, marking the latency from when sensors collect data to when it is initially processed at the edge layer. This extends to edge-to-fog and fog-to-cloud latencies, which include the compounded effects of processing and transmission as data progresses through the network layers. The cumulative network latency, T_{total}, is the sum of these individual latencies, expressed as:

$$T_{total} = T_p + T_t + T_{proc} + T_q. \tag{47}$$

Realistic modeling of network conditions requires incorporating the stochastic nature of queuing and processing delays, reflecting the complexities of network traffic and node performance. For ARCog-NET validation, this will be accounted for within the simulation given the time a data starts its processing and the time it ends for calculating the delay, and the time a message enters the queue and the time it leaves it for processing or transmission queue delay calculation as well.

3.5.2 Throughput

Modeling throughput in an edge-fog-cloud network, which is critical for understanding the network's capability to handle data over a specific period, involves considering the data processed and transmitted across different layers of the architecture. Throughput, typically measured in bits per second (bps), can be influenced by several factors, including

network infrastructure, the computational capacity of nodes, and the efficiency of data routing protocols. The ARCog-NET throughput model of a layer j is based on the model described in Pirmagomedov et al. (2022) for 5G network throughput optimization (as the proposed model can be seen as a kind of mobile communication technology), and is calculated using the following equation:

$$Tpt[\text{bps}] = \sum_{j=j_1}^{j_n} \nu_{\text{layers}}^{(j_n)} Q_m^{(j_n)} f^{(j_n)} R_{\text{max}} \cdot \frac{N_{\text{w}}^{\text{bw}(j_n)} \cdot 12}{T_s} \cdot (1 - \text{OH}^{(j_n)}), \tag{48}$$

where the parameters j_n is the number of aggregated component carriers signals (CC) in a band or band combination (which will depend on the chosen wireless technology), $\nu_{\text{layers}}^{(j_n)}$ represents the maximum number of aggregated layers trying to send the message, $Q_m^{(j)}$ represents the maximum modulation order (a integer number that depend on the wireless technology used), $f^{(j)}$ is the scaling factor (used for medium and high mobility and can take values 0.4, 0.75, 0.8, and 1), R_{max} is the maximum rate (which depends on the type of coding, therefore the employed wireless technology and $N_{\text{w}}^{\text{bw}(j_n,\mu)}$ being the number of allocated frequency windows per bandwidth (which will also depend on the transmission technology), T_s is the symbol duration and $\text{OH}^{(j_n)}$ is the overhead for control channels (it depends if it is a uplink or a downlink).

3.5.3 Quality of service and packet loss

Packet loss in an edge-fog-cloud network, particularly when applied to UAV swarm systems, can be attributed to several factors. Congestion is a primary cause, where network traffic exceeds capacity, leading to buffers overfilling and packets being discarded. In wireless network segments, interference from other devices, physical obstructions, or even atmospheric conditions can corrupt packets or prevent them from reaching their destination. Hardware issues are another common culprit, with malfunctioning routers, switches, or network interfaces interrupting the flow of packets. This can be due to aging hardware, defects, or environmental factors like temperature extremes affecting the hardware's operation.

Software problems, such as bugs or configuration errors, can also result in packets

being lost. These might occur within the routing protocols themselves or be due to improper setup by network administrators. Additionally, security features like firewalls or intrusion detection systems might inadvertently block or drop legitimate packets if they are misconfigured. Transitions between the edge, fog, and cloud layers involve complex routing and data handoff processes. If these transitions are not managed seamlessly, packets may be lost during the process, especially if the network's topology is dynamic or if the system lacks adequate handoff protocols.

In the specific context of UAVs, mobility-induced packet loss is also significant. The UAVs' movement can lead to variable signal quality and necessitate frequent handoffs between network cells, increasing the opportunity for packet loss. Moreover, the Quality of Service (QoS) settings within the network, designed to prioritize certain types of traffic, can cause lower-priority packets to be dropped when the network is congested. Overall, mitigating packet loss in such a complex network environment involves implementing robust error handling and correction protocols, designing the network for redundancy, and carefully managing network traffic to prevent congestion.

To estimate packet loss in ARCog-NET a small adaptation of the model proposed in Wu et al. (2023) will be used. The present network architecture can be considered a non-stationary network due to its dynamic nature, which is characterized by changes in their topology, configuration, and operating conditions over time (that can result from mobility, varying network loads, adaptive behaviors, and the fluctuating availability of resources), the proposed evaluation methodology works well for the simulation. The packet loss $Pkg_{L'_j,L_i}$ between information exchanged between member i of layer L and the member j of layer L' can be described as:

$$Pkg_{L'_j,L_i}(t) = 1 - \frac{r_{L'_j,L_i}((t-1)\Delta t, (t-1)\Delta t + \Delta t_W)}{s_{L'_j,L_i}((t-1)\Delta t, (t-1)\Delta t + \Delta t_W)}, \tag{49}$$

where t is the moment the information is sent, and $s_{L'_j,L_i}((t-1)\Delta t, (t-1)\Delta t + \Delta t_W)$ indicates the number of packets sent by node L_i to L'_j within the time period $(t-1)\Delta t, (t-1)\Delta t + \Delta t_W)$. The number of these packets received by L'_j is $r_{L'_j,L_i}((t-1)\Delta t, (t-1)\Delta t + \Delta t_W)$, being Δt_W the length of time observation window and Δt is the sliding step of the observed time window. If the package loss between components is too high, the network may change data routing through performance and QoS evaluation.

3.5.4 Energy consumption and efficiency

Modeling energy consumption in an edge-fog-cloud network, especially one incorporating UAV swarm systems, requires considering various energy-consuming components and activities. The model can be broadly divided into several categories, each associated with different network operation aspects. The energy consumed by UAVs for propulsion, denoted as $E_{\text{propulsion}}$, is a significant factor. This encompasses the energy for taking off, hovering, navigating, and landing, modeled as:

$$E_{\text{propulsion}} = E_{\text{takeoff}} + E_{\text{hover}} \times t_{\text{hover}} + E_{\text{navigate}} \times t_{\text{navigate}} + E_{\text{land}}. \tag{50}$$

Here, E_{takeoff} and E_{land} represent the energy for takeoff and landing, E_{hover} and E_{navigate} are the per unit time energy consumption for hovering and navigating, with t_{hover} and t_{navigate} indicating the respective duration. The energy used for data processing, $E_{\text{processing}}$, includes energy expenditures of onboard computers and sensors on UAVs, as well as edge and fog nodes. It can be estimated by:

$$E_{\text{processing}} = P_{\text{processing}} \times t_{\text{processing}}, \tag{51}$$

where $P_{\text{processing}}$ is the power consumption rate of processing units, and $t_{\text{processing}}$ is the processing time. Energy consumption for data transmission, $E_{\text{transmission}}$, accounts for the energy to send and receive data among UAVs and between the UAVs and network layers:

$$E_{\text{transmission}} = P_{\text{transmit}} \times t_{\text{transmit}} + P_{\text{receive}} \times t_{\text{receive}}, \tag{52}$$

with P_{transmit} and P_{receive} being the power consumption rates for transmission and reception, and t_{transmit} and t_{receive} the respective durations. Including additional energy costs, such as system maintenance ($E_{\text{maintenance}}$), the total energy consumption model becomes:

$$E_{total} = E_{\text{propulsion}} + E_{\text{processing}} + E_{\text{transmission}} + E_{\text{maintenance}}. \tag{53}$$

This model offers a comprehensive framework for understanding and optimizing energy usage across the network.

3.5.5 Data compression

In the proposed architecture, particularly for governing UAV swarm systems, managing data stream and processing, data compression emerges as a critical technology. It enhances the efficiency of data transmission across the network by reducing the bandwidth required to transmit information from UAVs to edge nodes, and ultimately to the cloud. By compacting data, compression mitigates the strain on network resources, which is essential for maintaining system performance as the volume of data generated by the swarm's sensors and instruments grows. This is especially important given the limited bandwidth available for wireless communication, which UAVs heavily rely on. Furthermore, data compression plays a significant role in storage optimization, a crucial consideration given the potentially vast amounts of data generated by each UAV in the swarm. Efficient storage not only conserves space but also speeds up data retrieval and analysis, leading to faster decision-making processes. In essence, data compression serves as a pivotal tool in the network's arsenal, enabling it to maintain high throughput and low latency, ensuring that the swarm can operate seamlessly and respond rapidly to commands and environmental stimuli.

To test a high volume of data being sent and processed in the network, RGB images were chosen. Compressing this data is an essential item to reduce data volume between links without losing too much information. Autoencoders are a type of neural network that tries to rebuild the input data at its output, but with an initial stage of reducing the input data to essential features and then building the output from this reduced set of features, and this reduced set of features can be used as compressed data to be stream in ARCog-NET. The first stage of the network is called the encoder, and the second one is called the decoder. The encoder outputs are the reduced set of features used to reconstruct the original input data. Autoencoders use unsupervised training to learn how to effectively reduce the information (code) and then represent the input again at its output (decode), with some loss of information (compression). For the model construction and training, it was used the TensorFlow Keras library. Figure 26 presents its structure.

The proposed autoencoder's input and output layers have $96 \times 96 \times 3$ shapes, representing an RGB input image with 96×96 pixels. The encoder structure consists of 4 groups of 4 convolutional 2D layers interspersed and stacked with 4 batch normalization

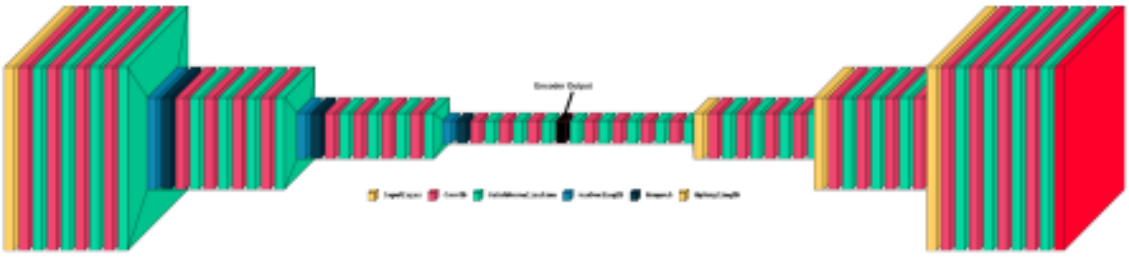


Figure 26 – Autoencoder Structure.

layers, totaling eight layers per group. The convolutional 2D layers create a convolution kernel that is convoluted with the layer input to produce a tensor of outputs. Those kernels represent the information extracted from the input data. The batch normalization layers apply a transformation that maintains the mean output close to 0 and the output standard deviation close to 1. That's why must be a batch normalization layer between two consecutive convolutions (the input data also must be normalized before compression). Between each group of layers, there is a maximum pooling 2D layer, which is what, in fact, downsamples the number of features extracted from each group, followed by a dropout layer to help prevent overfitting during training. The encoded input at the end of the encoder stage has dimension $12 \times 12 \times 32$.

The decoder structure is also composed of 4 groups of 4 convolutional 2D layers interspersed and stacked with 4 batch normalization layers. Still, instead of the maximum pooling 2D layers followed by the dropout layer, there is an upsampling 2D layer for reconstructing the data to its original format. The decoder output has shape $96 \times 96 \times 3$, the same as the input, which also represents an RGB image of 96×96 pixels. In other words, the decoder tries to reproduce the input image data with a reduced set of features after the encoder process. In order to train the model, it was used the STL-10 dataset (Coates et al., 2011), which is composed of 10,000 RGB generic images of 96×96 pixels, with 6,000 being used for training and 4,000 used for the test. The training was performed in 100 epochs, with a batch size of 10, and Adam optimizer with a learning rate of 0.0005. The loss and validation loss criteria chosen were the minimum squared error and the accuracy criteria performed for the evaluation.

In order to validate the model, it was studied the loss and validation loss curves, as well as the accuracy and validation accuracy curves. It also evaluated the peak-to-signal noise ratio (PSNR) variation for 1 to 10 consecutive compressions, presented in Equation (54).

$$PSNR = 10 \log_{10} \left(\frac{{MAX_I}^2}{MSE} \right), \tag{54}$$

where:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} (I_{(i,j)} - K_{(i,j)})^2, \tag{55}$$

being MAX_I is the maximum possible pixel value of the image, and MSE is the minimum squared error between the pixels (i, j) of images I (original) and K (compressed), of shape m$\times$n. The structural similarity index (SSIM, given in Equation (56)) and the multi-scale structural similarity (MS-SSIM, presented in Equation (57)) were evaluated as well.

$$SSIM = \frac{[2\mu_x\mu_y + (k_1L)^2][2\sigma_{xy} + (k_2L)^2]}{[{\mu_x}^2 + {\mu_y}^2 + (k_1L)^2][{\sigma_x}^2 + {\sigma_y}^2 + (k_2L)^2]}, \tag{56}$$

$$MS - SSIM = \frac{1}{M} \sum_{j=0}^{M} SSIM(x_j, y_j). \tag{57}$$

The SSIM measures the similarity between two images, where μ_x and μ_y are the average values of x and y respectively, ${\sigma_x}^2$ and ${\sigma_y}^2$ are the variances of x and y respectively, σ_{xy} is the covariance between x and y, and finally $(k_1L)^2$ and $(k_2L)^2$ are two stabilization constants, where in this analysis $k_1 = 0.01$, $k_2 = 0.03$ and $L = 2^n - 1$ where n is the number of bits per pixel. MS-SSIM is nothing else but the average SSIM for a set of images. All training and evaluation were executed from an Intel NUC 38i3BEK, using Google Collab computational resources to speed up the training.

3.5.6 Reliability and scalability

Network reliability is a critical measure of a system's operational performance over time, reflecting its ability to function continuously without failure. Central to this concept are the metrics of Mean Time Between Failures (MTBF) and Mean Time To Repair (MTTR). MTBF is an indicator of the average time elapsed between inherent failures of a system during operation. It provides insight into the durability and stability of network components under standard usage conditions. A higher MTBF corresponds to a more reliable network,

as it suggests a lower frequency of failures over a given period. In this research, a failure represents an absence of a communication link that should exist between a sender and a receiver. The network oversight should provide a new link between the sender and the network. Conversely, MTTR measures the average time required to repair a failed network component, reflecting the network's maintainability and the efficiency of its recovery processes. An optimal network design strives for a high MTBF and a low MTTR, ensuring both minimal failures and rapid recovery in the event of a malfunction, which together contribute to the overall robustness and dependability of the network. In the present work, this time is related to the time taken for the network to identify this failure and provide a new routing for the link. The availability of a layer is calculated by:

$$Av_{L_i} = 1 - \frac{1}{\sum_{i_1}^{i_n} Tdt_{L_i}}, \tag{58}$$

where Av_{L_i} is the availability of a layer i and Tdt_{L_j} is the total downtime of this layer, which is the sum of total time that a communication link is offline.

Regarding network scalability, it is worth mentioning that it is a vital attribute that determines a network's capability to manage growth and handle an increasing number of concurrent users without performance degradation. Scalability encompasses the network's ability to expand its capacity and functionalities efficiently as demand rises, which includes adding more nodes, such as UAVs in a swarm, or enhancing the bandwidth to accommodate more data flow. The measure of scalability is not just about handling a growing quantity of data or number of devices; it's also about the network's agility in integrating new technologies and adapting its architecture to meet evolving requirements. A truly scalable network maintains its performance levels and service quality even as the number of concurrent users surges, ensuring that each user's experience remains unaffected by the changes and expansions in the network infrastructure.

This aspect of scalability is crucial for future-proofing the network, enabling it to support not just the current demands but also to be prepared for future increases in network traffic and user numbers. In practice, network scalability is often evaluated through modeling and simulation, which can help predict how the network will behave as it grows. Benchmarks and stress tests can also provide insights into the network's scalability by simulating high-traffic conditions and observing the network's response to such stressors. To check ARCog-NET scalability, it will be shown in results how the other mentioned

parameters behave as the number or UAVs increase in network, and also how the network will behave depending on the layer that the new nodes are added.

4- Results and Discussion

The implemented ARCog-NET architecture is available through a GitHub repository (Ramos, 2025a). In the next subsections, the results obtained in simulations will be presented and discussed, analyzing ARCog-NET functionalities and effectiveness compared to other stand-alone solutions in the literature.

4.1- Environment 1: Vertical turbines wind energy field surveillance

Altogether, 10 simulations were performed using 1 to 10 robots using Gazebo GUI, totaling 100 graphical simulations for this application, and another 1000 simulations without graphics, from 1 to 1000 vehicles. As presented before, the goal of this simulation is to check the capability of ARCog-NET framework to plan and execute the task of flight between a known mesh of relatively close obstacles while avoiding collisions between robots and these objects, reducing communication interference, latency and package loss between layers and allow a good level of data throughput, because the robots are being used to inspect the wind turbines structures for predictive maintenance. Figure 27 shows an example of trajectories planned and executed by a group of 10 UAVs, and this simulation video is presented in Ramos (2024a). In this Figure, the server (cloud layer) is initially configured to request the structure to use some of those robots as fog nodes (which can vary from 1 to 10 robots), which are also edge group coordinators. The number of the other edge agents may vary throughout the mission, in a way that a fog coordinator may have up to 9 agents connected to it.

It is possible to notice how the initial calculated trajectories graph, which was planned using the Particle Swarm Optimization (PSO) algorithm presented, shows that on certain occasions, some of the UAV would collide with an obstacle (a vertical wind turbine, represented in the graph as black points). Alternatively, on the navigated trajectories graph, it is possible to notice the evasive actions performed by the UAVs while accomplishing the mission, also calculated with the PSO decision-making loop presented in Methodology.

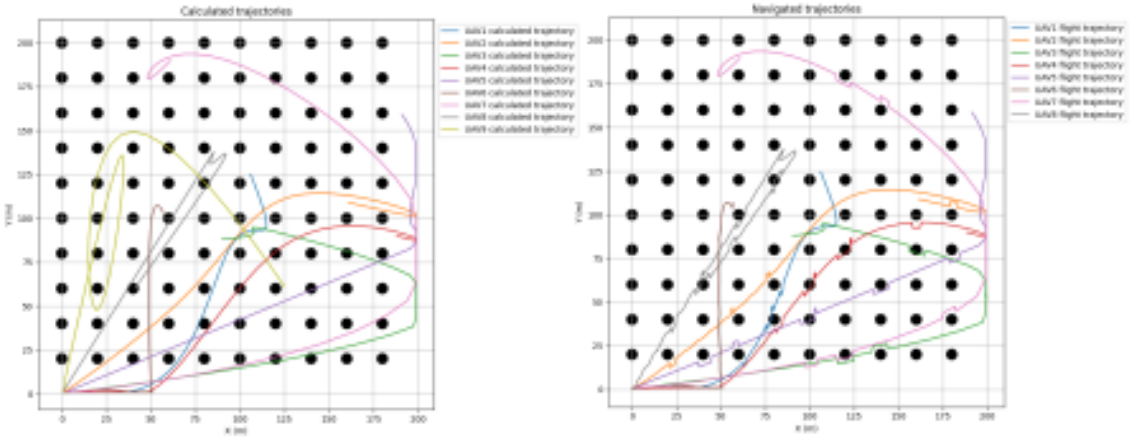


Figure 27 – UAVs flying between wind turbines.

The obstacle avoidance is essentially the decision to fly around the obstacle, bypassing it "above" or "below" on the xy-plane. As exposed in Section 3, this obstacle avoidance event decision can be made in each layer by a robot or in its superior layer by request. Figure 28 shows the number of decisions made at each layer according to the number of UAVs on each simulation.

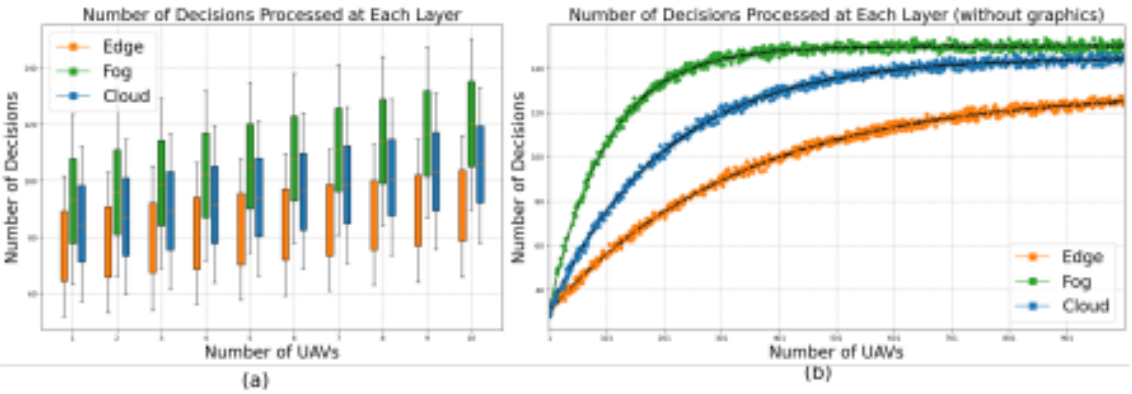


Figure 28 – Number of decisions processed on each ARCog-NET layer in the wind field simulation.

When observing the number of processed decisions, it's possible to notice in (a) graphic of Figure 28 that, when a single UAV is connected to the architecture, the structure becomes only edge-cloud or only fog-cloud (depending on the cloud's configuration, but in practice, these two topologies have the same capabilities). With more than 1 robot, the structure will necessarily have an edge node and a fog node alongside the cloud layer. That being said, the graph suggests variability in the number of decisions processed at different layers, likely showing how increasing the number of UAVs impacts the decision load at each layer. It is expected that most simple decisions will be made in the edge

layer without the need to send processing requests to the layers above. At 2 UAVs, it is possible to see an interesting behavior of ARCog-NET, as one robot is an edge node and the other is a fog (and they can change roles as the mission unfolds), the median number of processed decisions are very close in these two layers for this number of UAVs on the mission, with the edge agent having more processing than the fog coordinator.

However, as the number of UAV increases on ARCog-NET in this environment densely populated with obstacles, more decisions must be processed. The graph shows that fog layer takes the lead in making decisions, an expected result for this kind of scenario, as fog acts like an intermediate layer, sometimes processing the decisions for the edge group and sometimes redirecting requests to the cloud layer. In the same way, the cloud can process more decisions and perform more complex computing, while the edge is most dedicated to flying and gathering data, processing fewer decisions than the other two. In (b) graphic of the same figure, it's possible to notice how the fog layer also saturates first when compared to the other two, but has more capacity to process more decisions in this scenario, showing that ARCog-NET is functioning well in distributing computing power through the network. The Figure 29 presents a Pareto graph containing the average number of each decision type taken during simulations

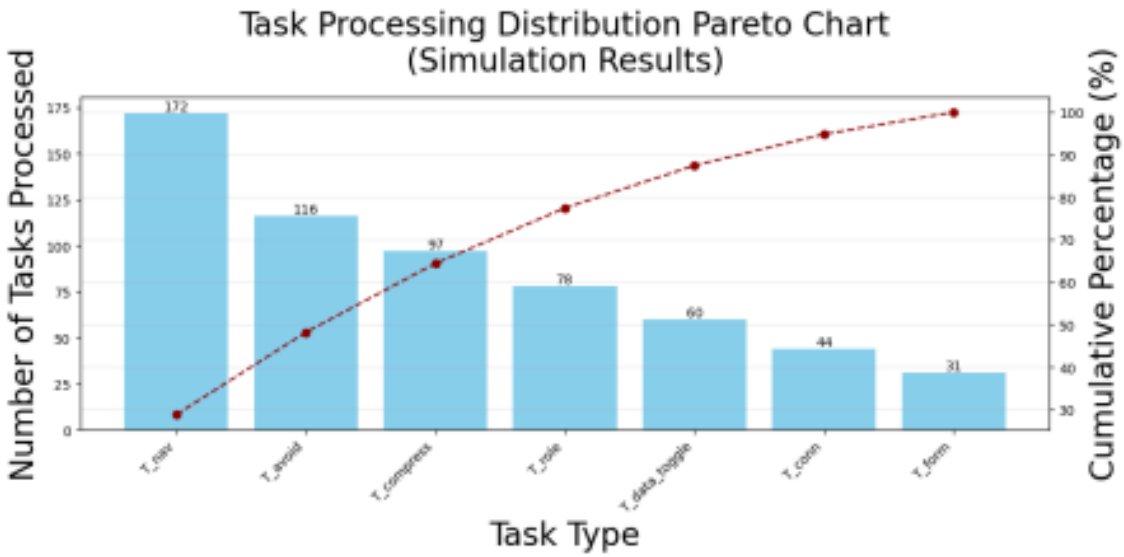


Figure 29 – Pareto Graph of the average number of tasks by type in the wind farm simulation

The largest amount of decisions processed while accomplishing missions in this environment are related to navigation, with T_{nav} and T_{avoid} representing almost 50% of the average decisions. This shows how the navigation-related tasks were important while

groups flew amidst fixed obstacles close to each other. Decisions had to be made in order to efficiently navigate in this environment, avoid collisions with objects or between components (other UAV) while gathering image data of the structures in order to inspect them. Data compression and data toggle (stop or start sending data) were also heavily processed in order to improve network bandwidth, improving availability to UAV, which were finding interesting objects and discarding UAV data that were only sending images of normal structures. The role changes were also used to keep the network connection as the UAV scattered. Finally, Formation Control was not used so much, as the idea for this inspection was to spread the nodes, mostly edges, through the wind farm field to guarantee more inspection coverage.

Overall, the results in Figures 28 and 29 are consistent, showing that the fog layer processed the largest share of decisions during the wind farm inspection. This behavior reflects the intermediate role of the fog layer, which not only handles decisions locally for its connected edge agents but also forwards more complex requests to the cloud when necessary. The predominance of navigation-related tasks (T_{nav} and T_{avoid}), which account for nearly half of all decisions, explains why the fog layer exhibited the highest processing load: navigation events occur frequently in environments with dense and closely spaced obstacles, requiring continuous trajectory updates and obstacle avoidance. While the edge layer primarily executed decisions necessary for immediate flight control and data collection, the fog layer aggregated and optimized these decisions, leveraging its higher computational capacity. The cloud layer, in turn, processed fewer decisions than fog but more complex commands, benefiting from global knowledge to support reinforcement learning and high-level coordination. This, in fact, is the role of fog: allow cloud to work on heavy computation while selecting what should upscale or not, dealing with requests in the intermediate layer, and only sending critical ones to the cloud. This distribution demonstrates that ARCog-NET effectively balances decision-making across its layers, with the fog layer acting as the primary processing hub in scenarios dominated by navigation-intensive tasks. Regarding benchmarking ARCog-NET against cited literature, Table 6 shows the observed results.

The framework achieves the highest efficiency (91%) among the compared methods, surpassing even reinforcement learning approaches. Although its energy consumption (96 J/task) is slightly higher than bio-inspired and reinforcement learning methods, this is offset by the lowest communication overhead (7%), demonstrating efficient network usage.

Table 6 – Task Allocation Comparison in Wind Turbine Inspection Scenario.

Metric	Market-Based	Heuristic Methods	Bio-Inspired Methods	Reinforcement Learning	ARCog-NET
Efficiency (%)	72	83	86	89	91
Energy Consumption (J/task)	125	105	98	92	96
Communication Overhead (%)	18	11	9	8	7
Adaptability (ms)	68	78	81	86	88
Scalability (number of UAVs)	480	1100	880	820	950

In terms of adaptability, ARCog-NET (88 ms) outperforms all alternatives, responding quickly to environmental changes. While its scalability (950 UAVs) is lower than heuristic methods (1100 UAVs), it remains high and suitable for complex scenarios. These results confirm ARCog-NET's viability, as it combines high efficiency, adaptability, and low communication cost, making it a robust solution for real-world applications such as wind turbine inspection and other critical missions. As most of the processed decisions are related to navigation tasks, to assess the efficiency of different algorithms in a static environment, the wind turbine inspection scenario was simulated. This context allows evaluating how each method handles obstacle-rich but predictable conditions. The performance of ARCog-NET was compared with classical PSO, A*, Dijkstra, and RRT in terms of convergence, coverage, computational cost, adaptability, and scalability, as shown in Table 7.

Table 7 – Path Planning Comparison in Wind Turbine Inspection Scenario.

Metric	Classical PSO	A*	Dijkstra	RRT	ARCog-NET
Convergence Time (s)	14	9	11	17	10
Trajectory Coverage (%)	82	89	91	86	94
Computational Cost (ms)	160	210	185	130	135
Adaptability (ms)	70	65	60	75	85
Scalability (number of UAVs)	400	250	350	550	900

In this static scenario, ARCog-NET achieved the highest trajectory coverage (94%) and superior adaptability (85 ms), outperforming traditional algorithms in maintaining precision and flexibility. Although RRT had the lowest computational cost, ARCog-NET balanced cost (135 ms) with high scalability (900 UAVs), making it more suitable for large-scale operations. A* offered the fastest convergence (9 s) but lacked adaptability, while Dijkstra provided good coverage but lower scalability. Overall, ARCog-NET demonstrated

the best trade-off between accuracy, adaptability, and scalability for static inspection missions. Before presenting the formation control results for the wind turbine inspection scenario, it is important to reinforce the context. The main idea in this scenario was for the components to scatter through the map, improving the chance of finding issues on turbine structures. In this sense, fewer Formation Control tasks were triggered as the UAV shouldn't fly in formations close to each other, a result confirmed in Figure 29. However, fewer FC tasks were triggered; what the UAV executed within simulations was compared with Distributed Model Predictive Control (DMPC), DRL-based methods, and Consensus-based control. The evaluation considered metrics such as formation error, energy consumption, adaptability, communication overhead, and robustness to delays. The results are summarized in Table 8.

Table 8 – Formation Control Comparison in Wind Turbine Inspection Scenario.

Metric	**DMPC**	**DRL-Based**	**Consensus -Based**	**ARCog-NET**
Formation Error (m)	0.14	0.19	0.09	0.11
Energy Consumption (J/formation)	52	58	42	68
Adaptability (ms)	82	87	76	80
Communication Overhead (%)	13	14	9	9
Robustness to Delays (%)	86	81	91	89

ARCog-NET achieved a low formation error (0.11 m), close to the consensus-based method (0.09 m) and better than DMPC and DRL. Its robustness to delays (89%) was high, almost matching the consensus approach (91%). However, ARCog-NET consumed more energy (68 J/formation) than the other methods due to its cognitive processing overhead. Adaptability (80 ms) remained competitive, slightly lower than DRL but higher than the consensus approach. Overall, ARCog-NET balanced precision and robustness well, at the cost of increased energy consumption. The average performance of different communication types in ARCog-NET for this static scenario is affected by the number of UAV connected to the network, the processed decisions and data being streamed, which can be assessed by analyzing the relationship between swarm size, latency, and energy consumption. Figure 30 illustrates this comparison.

The results show that edge-edge communication consistently offers the lowest latency, even with increasing swarm size, due to its localized nature. However, energy

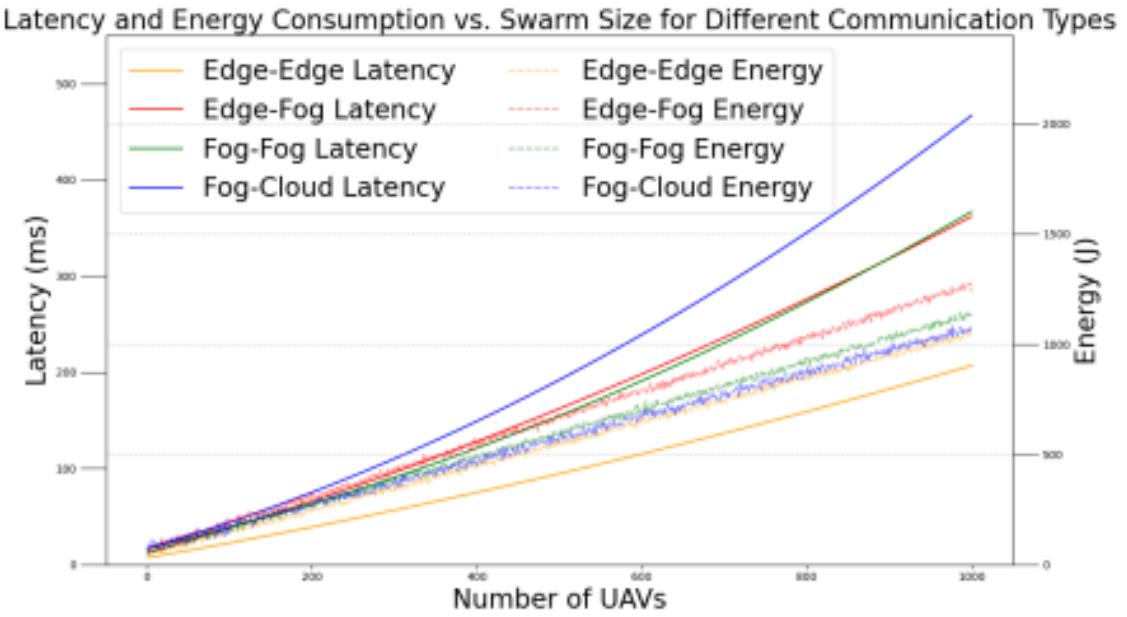


Figure 30 – Average ARCog-NET latency and power consumption by number of UAVs at each time in the wind farm simulation.

consumption grows steadily as more UAVs participate. Edge-fog and fog-fog communications display slightly higher latency, reflecting the intermediate coordination overhead, but maintain moderate energy consumption. fog-cloud communication, while enabling global optimization, incurs the highest latency and energy cost, particularly as the swarm scales. This trade-off indicates that ARCog-NET benefits from balancing edge-local decisions with selective use of fog and cloud layers, minimizing latency while preserving adaptability and energy efficiency in large-scale operations.

As the latency and energy consumption chart (Figure 30) reveals, ARCog-NET maintains consistently low latency while operating within acceptable energy limits. This balance is critical for online UAV coordination, ensuring responsiveness without compromising mission endurance. Compared to traditional methods, ARCog-NET demonstrates improved stability in latency fluctuations, which enhances overall system reliability in static environments. The throughput chart (Figure 31) complements this analysis by showing how ARCog-NET manages data transmission under increasing network demands. The results indicate efficient bandwidth usage with minimal degradation in throughput, confirming the effectiveness of its cognitive data management and compression mechanisms. Together, these results validate ARCog-NET's ability to maintain high communication performance and energy efficiency, even under network constraints, reinforcing its suitability for large-scale UAV swarm operations.

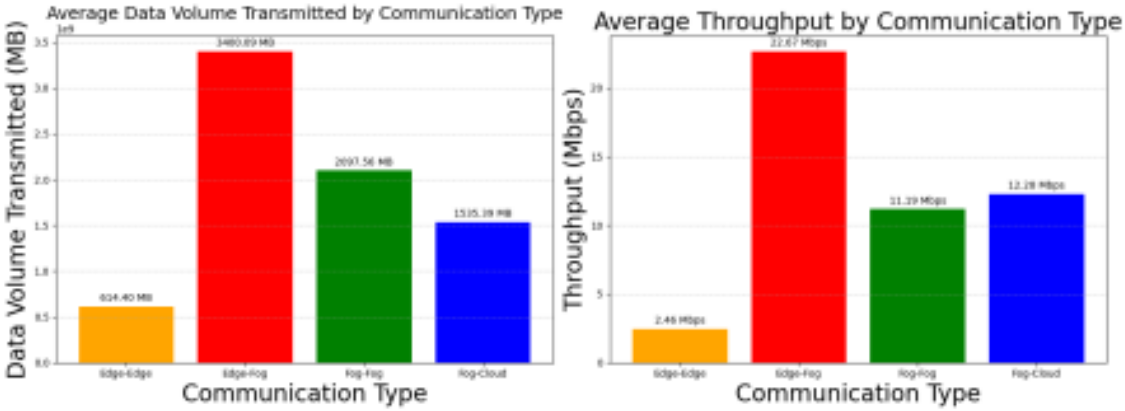


Figure 31 – Average ARCog-NET transmitted data and throughput in wind farm simulation.

In terms of data volume, edge-fog communication stands out as the most demanding, transmitting approximately 3.4 GB. This high volume is consistent with its role as an intermediate layer handling both local processing and coordination. fog-fog and fog-cloud communications follow, with around 2.1 GB and 1.5 GB transmitted, respectively. These values suggest that while fog nodes still handle substantial data, cloud communication is relatively optimized, likely due to data compression or selective transmission strategies. Edge-edge communication, by contrast, involves the least data (about 614 MB), reflecting its localized interactions with limited data sharing.

Throughput results reinforce this interpretation. Edge-fog achieves the highest throughput (22.67 Mbps), indicating that it not only transmits the most data but does so efficiently. fog-cloud and fog-fog communications achieve moderate throughput (12.28 Mbps and 11.19 Mbps, respectively), aligning with their roles in managing larger-scale but less frequent data exchanges. Edge-edge communication shows the lowest throughput (2.46 Mbps), which correlates with its minimal data volume and its primary function of handling small, online exchanges between individual UAVs.

Overall, the analysis reveals that edge-fog communication is both the most data-intensive and the most capable in terms of throughput, supporting its critical role in coordinating swarm activities. fog-cloud strikes a balance between volume and speed, facilitating higher-level processing without overwhelming network resources. Edge-edge remains lightweight, ensuring low latency for direct UAV interactions. These findings emphasize that ARCog-NET effectively leverages the strengths of each communication layer, with data volume and throughput scaling according to their hierarchical roles.

Another important aspect of analyzing is data compression when discussing the

network performance associated with the streamed information. For this particular mission, each UAV image is important as the user (supervisor) HITL may identify corrosion or other kinds of damage at the wind turbines being inspected. As presented in Section 3, there is a built-in data compressor for images in ARCog-NET, which is an autoencoder that allows successive compression and decompression stages in a way that the data can be streamed without any information loss (raw data) or greater losses that the end-user may allow (users set which compression level is more interesting to themselves, and this asset is reconfigurable during the mission). The compression will also affect latency, throughput, and packet loss because the more the data is compressed, the fewer transmission parameters are demanded. Still, more processing loads are required (for data compression and decompression). The details of the implemented autoencoder, training, and evaluation results can be found at Ramos et al. (2023), developed by the authors of the present study. A compression result example is presented in Figure 32, which shows an image captured from one of the robots during the wind field simulation detailing corrosion found at one of the wind turbines' rotary shafts and how this image is affected by successive compression with the trained autoencoder.

It is possible to notice that it's still possible to identify damage to the structure even with five successive compressions. However, at 10 compression levels, the picture becomes unrecognizable, affecting the mission objective. Also, this example shows big and easy-to-identify damage, so smaller ones may also be affected by less compression. The model compressing and decompressing computation is fast, so the byte volume transmission reduction (with the cost of losing data) can be achieved by compressing the image several times before streaming it. The number of compressions for each UAV transmitting image could be defined in the offboard controller nodes, which is the group's collective decision on which agent should detail more or less of the data they're sending. In this simulation case, the number of compressions is a decision made by the cloud supervisor HITL. In order to validate the autoencoder's performance, images from the UAVs were used during the simulation, being compressed from 1 to 10 times, and the Peak signal-to-noise ratio (PSNR), Structural Similarity Index (SSIM), and Multi-Scale SSIM (MS-SSIM) were computed. Figure 33 presents the variation results for each indicator.

The analysis of packet loss and availability by communication type, illustrated in Figure 34, complements the evaluation of ARCog-NET's communication performance in the wind turbine inspection scenario. This figure shows how different communication

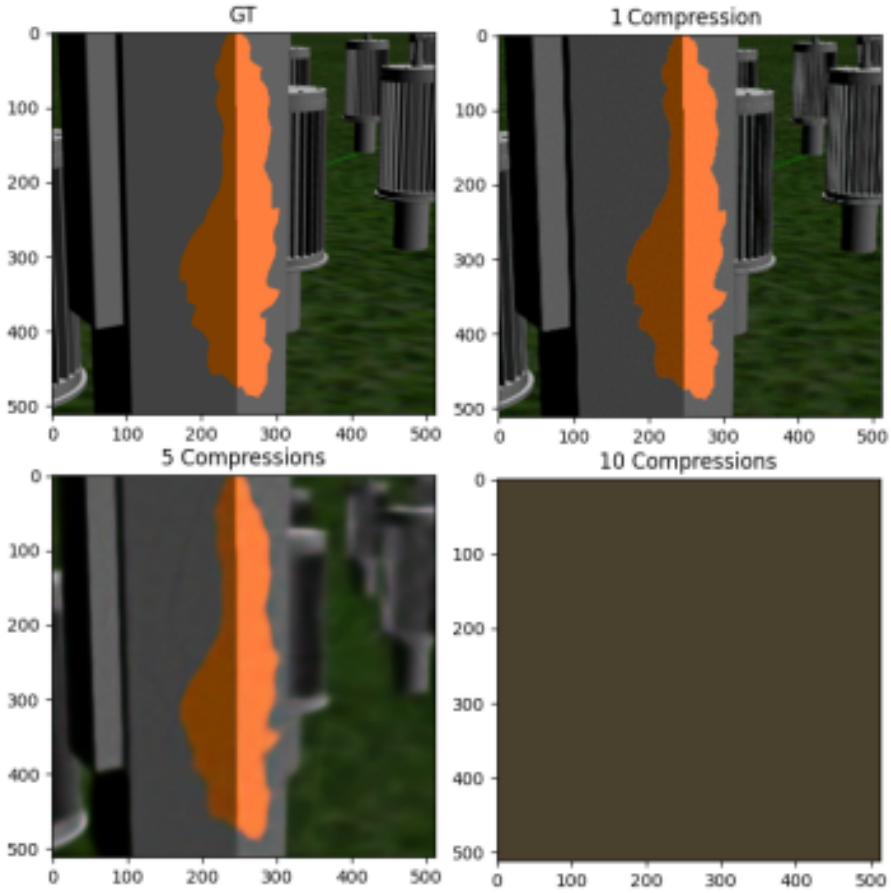


Figure 32 – Compression example of a picture of corrosion found at a wind turbine shaft.

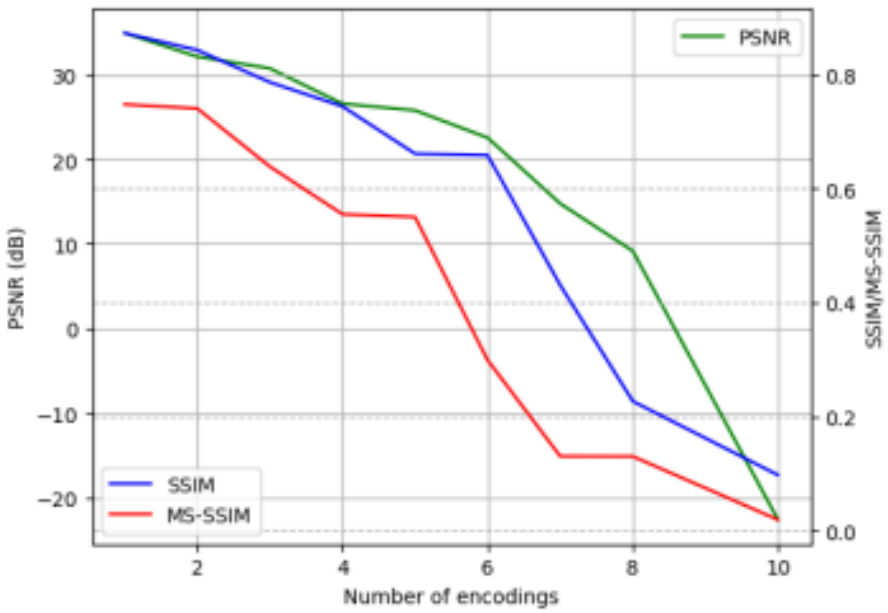


Figure 33 – PSNR, SSIM, and MS-SSIM variation for interlinked compression in wind field simulation.

layers manage reliability under varying network conditions.

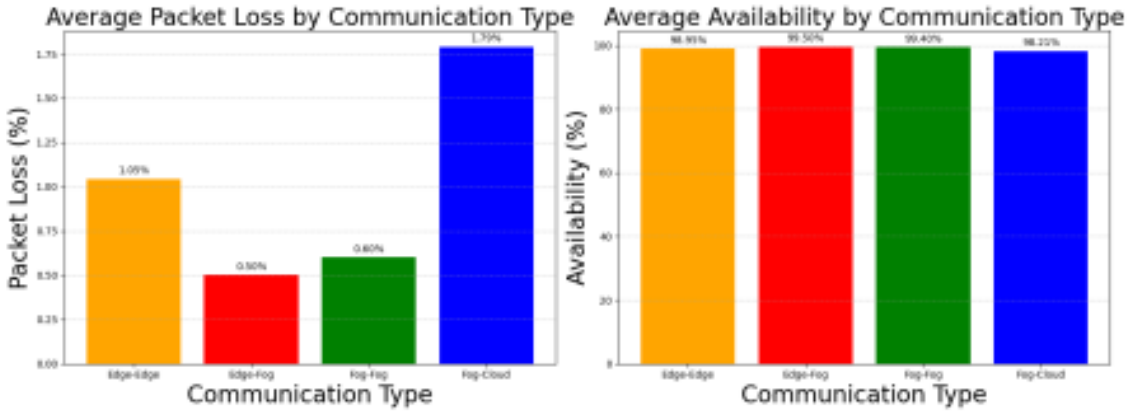


Figure 34 – Average ARCog-NET packet loss and throughput in wind farm simulation.

The results reveal that edge-fog communication achieves the lowest packet loss (0.50%) and the highest availability (99.50%), confirming its effectiveness as the main coordinator for intermediate processing. This aligns with previous findings where edge-fog also demonstrated the highest throughput, making it the most reliable communication channel for ensuring continuous mission execution. fog-fog communication follows closely with slightly higher packet loss (0.60%) and nearly identical availability (99.40%), reinforcing its role as a stable link for inter-coordinator exchanges.

Edge-edge communication presents a higher packet loss rate (1.05%) than fog-fog and edge-fog, with availability slightly lower (98.95%). This is expected since direct exchanges between edge nodes are more susceptible to local interference and rely on simpler transmission mechanisms. Nevertheless, its reliability remains acceptable, supporting its use for online interactions that require low latency. fog-cloud communication shows the highest packet loss (1.79%) and the lowest availability (98.21%). This behavior reflects the greater distance and complexity involved in maintaining stable cloud connections, especially when network load increases. Despite this drawback, cloud communication is used selectively for critical tasks requiring high computational power, as discussed earlier.

These findings reinforce the design choice of ARCog-NET to delegate most decisions and data processing to the fog layer while minimizing cloud interactions and limiting edge-to-edge transmissions to essential exchanges. By balancing packet loss and availability across layers, ARCog-NET ensures resilient and efficient communication even in obstacle-dense environments, thereby sustaining overall system performance and mission reliability.

4.2- Environment 2: Offshore environmental monitoring

This mission aims to evaluate ARCog-NET performance when applied in an outdoor obstacle-free unknown environment by simulating an offshore ocean water surveillance mission scenario. The robots and the network were configured to take off from a helicopter deck placed on an oil production facility and scan the water, searching for oil spills. This time, the strategy programmed in the PSO loop was to deploy one UAV at a time to fly "zigzag" trajectories in order to cover a large area, avoiding colliding with each other. When operating with a single vehicle, it is configured as an edge or fog node (there is no difference in this case) and flies alone, building its knowledge base of trajectories and finding oil spills along the way. If the vehicle doesn't find anything, the loop (local or cloud) will increase the search area. For more than one UAV, as one edge vehicle starts scanning the ocean, it sends the collected data to its fog coordinator, counting spills found and sending other agents to other points, increasing the search area if necessary. Also, the cloud layer will act again as a supervisor layer, commanding the whole system and processing flight decisions. A video simulation example using 10 UAVs with 3D graphics is presented in Ramos (2024b), and Figure 35 presents the flight behavior of this video.

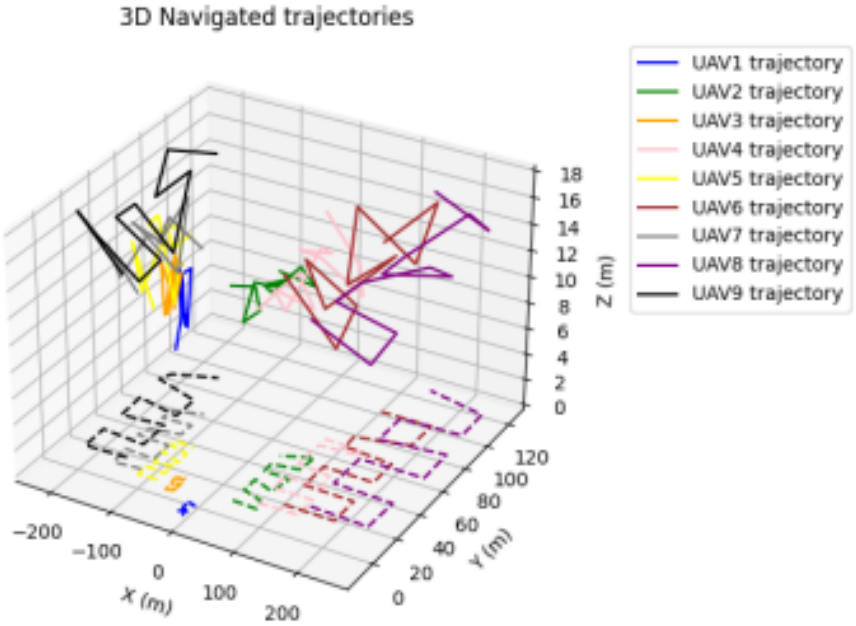


Figure 35 – Group of 10 UAVs flying to search for oil spills in an offshore simulation.

In the video, the group of 10 UAVs (8 edge agents separated into two groups, with

one fog coordinator each) found all oil spills randomly placed over the water surface (26 spots) in simulation, taking 4 minutes and 32 seconds. By navigating the routes presented in Figure 35, the swarm covered an area of $48,000$ m^2. Compared to state-of-the-art methods, the results of ARCog-NET for this scenario are presented in Table 9.

Table 9 – Path Planning Comparison in Offshore Scenario.

Metric	Classical PSO	A*	Dijkstra	RRT	ARCog-NET
Convergence Time (s)	12	8	10	15	9
Trajectory Coverage (%)	85	90	92	88	91
Computational Cost (ms)	150	200	180	120	130
Adaptability (ms)	75	70	65	80	78
Scalability (number of UAVs)	500	300	400	600	1000

For this scenario, ARCog-NET demonstrates competitive performance in path planning, balancing efficiency, adaptability, and scalability. It achieves a convergence time of 9 seconds, faster than classical PSO (12 s) and RRT (15 s) but slightly slower than A* (8 s). This highlights ARCog-NET's ability to optimize path planning in dynamic environments while maintaining computational efficiency. In trajectory coverage, ARCog-NET achieves 91%, outperforming classical PSO (85%) and Rapidly Exploring Random Trees (RRT) (88%) but slightly trailing Dijkstra (92%). This indicates robust path optimization, though Dijkstra remains optimal for static environments.

ARCog-NET's computational cost is 130 ms, lower than A* (200 ms) and Dijkstra (180 ms), but slightly higher than RRT (120 ms). This reflects a trade-off between computational efficiency and adaptability. In adaptability, ARCog-NET scores 78 ms, outperforming classical PSO (75 ms) and Dijkstra (65 ms) but slightly below RRT (80 ms). This underscores its ability to handle dynamic scenarios effectively. Finally, ARCog-NET supports 800 UAVs, surpassing classical PSO (500 UAVs), A* (300 UAVs), and Dijkstra (400 UAVs), though RRT scales to 600 UAVs. This scalability makes ARCog-NET suitable for large-scale deployments.

Overall, ARCog-NET provides a balanced solution for path planning, excelling in scalability and adaptability while maintaining competitive efficiency and computational performance. It is suitable for this dynamic, large-scale UAV operation in an open space, obstacle-free scenario. The formation control asset was triggered very few times, but enough to measure and compare results as presented in Table 10.

Table 10 – Formation Control Comparison in offshore scenario.

Metric	DMPC	DRL-Based	Consensus -Based	ARCog-NET
Formation Error (m)	0.15	0.20	0.10	0.12
Energy Consumption (J/formation)	50	60	40	45
Adaptability (ms)	80	85	75	78
Communication Overhead (%)	12	15	8	9
Robustness to Delays (%)	85	80	90	88

The proposed architecture demonstrates strong performance in formation control, balancing precision, energy efficiency, and adaptability. It achieves a formation error of 0.12 m, outperforming Distributed Model Predictive Control (DMPC) (0.15 m) and Deep Reinforcement Learning (DRL)-based methods (0.20 m) but slightly trailing consensus-based approaches (0.10 m). In energy consumption, ARCog-NET uses 45 J/formation, more efficient than DMPC (50 J) and DRL-based methods (60 J) but slightly higher than consensus-based approaches (40 J). ARCog-NET's adaptability is 78ms, better than consensus-based methods (75 ms) but slightly below DRL-based approaches (85 ms). Its communication overhead is 9%, lower than DMPC (12%) and DRL-based methods (15%), but slightly higher than consensus-based approaches (8%). Finally, ARCog-NET demonstrates 88% robustness to delays, outperforming DMPC (85%) and DRL-based methods (80%) but slightly below consensus-based approaches (90%).

Being an open environment free of obstacles, it is a relatively simple situation. In this case, the ARCog-NET structure is not so demanding to process complex flight decisions; the robots are not submitted to high interference rates, as the signals have free space to propagate, less data to pack, compress, and stream. The result is a more fluid overall performance of the architecture. To exemplify this observed behavior, the first step is to analyze each layer of the computer load of decisions (task allocation as response to events) processed in Figure 36 based on the presented RL cognitive algorithm.

The average type of decisions is compiled and presented in the Pareto graph of Figure 37.

The architecture behavior for this scenario reveals that less cloud computation was required, as the edge and fog robots could deal with simpler decisions locally or within their layer pairs. An interesting result is the saturation behavior of the network, in

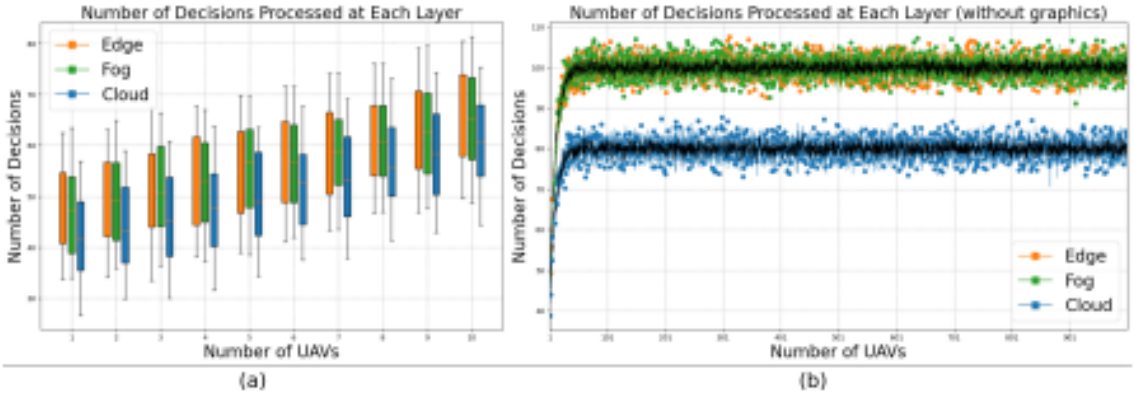


Figure 36 – Number of decisions processed on each ARCog-NET layer in the offshore simulation.

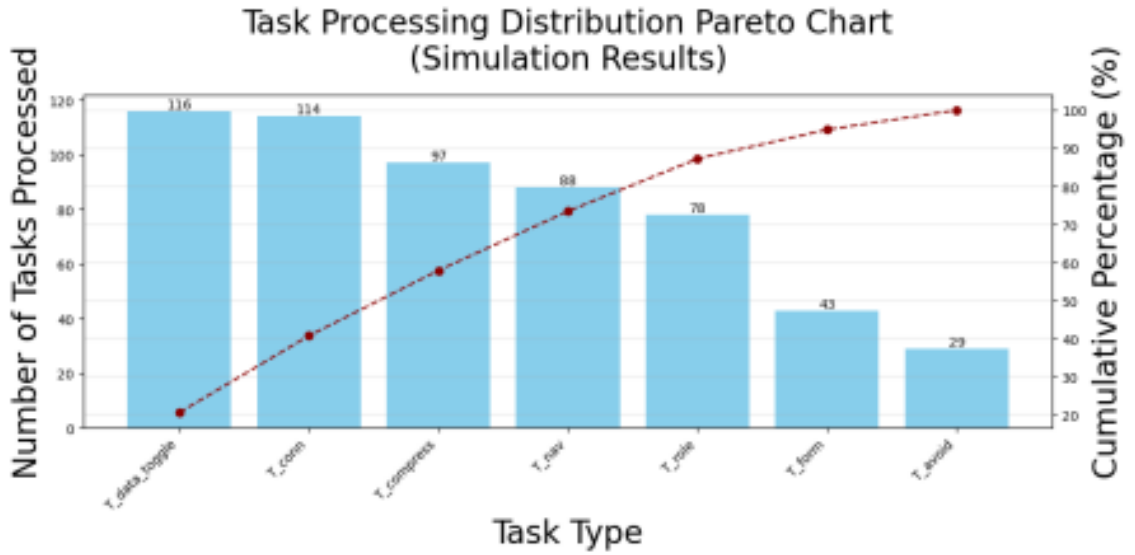


Figure 37 – Pareto Graph of the average number of tasks by type in the offshore simulation.

which the average number of decisions processed on each layer doesn't increase after a count of approximately 50 nodes connected. This has an impact on network scalability for this scenario, as edge, fog, and cloud layers seem to reach their peak of decisions throughout the whole simulation time after 50 robots are connected, indicating that there is no need for using more UAVs to accomplish mission goals with ARCog-NET in this case. The Pareto graph also demonstrated that the architecture had to deal with fewer flight decisions, with communications and data handling being the most relevant factors (probably because of the large area the UAVs were scanning). Comparing task allocation with other methodologies presented the following results (Table 11):

In this scenario, ARCog-NET achieves 88% efficiency, closely matching reinforcement learning (90%) and bio-inspired methods (88%), while outperforming market-based

Table 11 – Task Allocation Comparison in offshore scenario.

Metric	**Market-Based**	**Heuristic** Methods	**Bio-Inspired** Methods	**Reinforcement** Learning	**ARCog-NET**
Efficiency (%)	75	85	88	90	88
Energy Consumption (J/task)	120	100	95	90	95
Communication Overhead (%)	15	10	9	8	7
Adaptability (ms)	70	80	82	85	82
Scalability (number of UAVs)	500	1200	900	800	1000

(75%) and heuristic approaches (85%). It minimizes energy consumption to 95 J/task, comparable to bio-inspired methods (95 J/task) and better than market-based methods (120 J/task). ARCog-NET excels in communication overhead at 7%, outperforming all other methods, including bio-inspired (9%) and reinforcement learning (8%). Its adaptability is 82ms, slightly lower than reinforcement learning (85 ms) but better than market-based (70 ms). ARCog-NET supported 1000 UAVs (although it's not useful to scale so much in this scenario, as demonstrated earlier), surpassing bio-inspired (900 UAVs) and reinforcement learning (800 UAVs), making it highly scalable for large deployments. Overall, ARCog-NET balances efficiency, energy use, and scalability, making it a robust solution for UAV swarm task allocation. With the saturation of decision processing, latency and energy become scalability concerns and are represented in Figure 38.

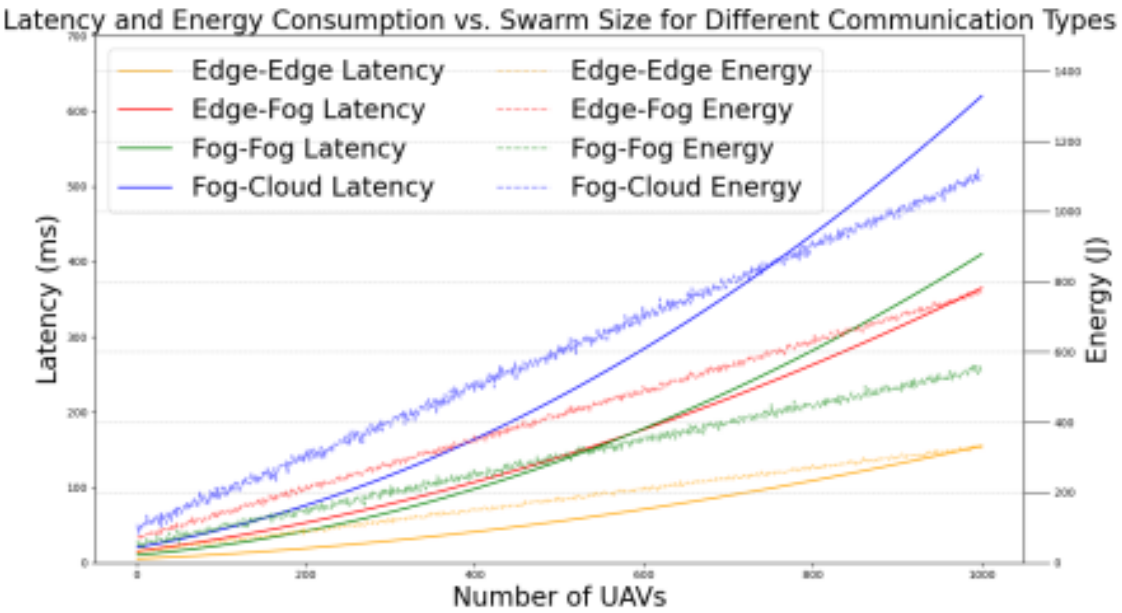


Figure 38 – Average ARCog-NET latency by number of UAVs at each time in the offshore simulation.

As expected, both energy and latency will increase as more robots are added to the architecture. The edge energy and latency are smaller than other communication pipelines within ARCog-NET, as edge communication is usually less heavy, with agents closer to each other. Latency and energy will also be a result of how the architecture will process the task allocation. With this open scenario, there's not much path planning changing (for example, for avoiding an obstacle) or even formation controls being triggered. ARCog-NET can also reconfigure its structure during the mission (edge agents changing groups, becoming coordinators, cloud demanding HITL decision, size of groups, etc.), which may impact energy and latency. Edge-to-fog and fog-to-fog communications have similar latency outputs, but edge-fog demands more energy with the latency gain payback.

In other words, for this scenario, investing in edge-to-fog groups will result in less latency than operating only with a fog computing strategy. The cloud layer presents the greater latency and energy due to its distance from UAVs and radio power. Overall, increasing UAV numbers in this scenario doesn't result in better mission goal accomplishment as the decisions processed will saturate after 50 robots, and latency and power consumption keep increasing as more robots are added.

Streamed data, the major task allocator in this scenario, was basically images (the larger messages) and positions, with some requests and commands eventually (that don't take up much bandwidth). When running some tests, different results were observed in spill detection behavior as the images didn't have many edges and textures (it was just plain water texture most of the time). Mathematically, using the ARCog-NET pre-built data compression autoencoder model (Ramos et al., 2023) in this simulation doesn't result in great PSNR, SSIM, and MS-SSIM when comparing raw captured images to compressed images. The results are very similar to the ones obtained in wind field simulation, as presented in Figure 39.

In practice, however, the cloud server supervisor HITL will struggle more with the compressed images when using the successive compression method to reduce data, as presented in Figure 40. In other words, as the image has little difference at each frame and poor texture, compressing it is unnecessary and hampers mission objectives so that this compression asset of ARCog-NET can easily be put aside in this kind of mission where the network operates with a good performance.

Even not using too much compression in this scenario simulations, the volume of data traffic within the network is relatively low, as detailed in Figure 41. The main factor for

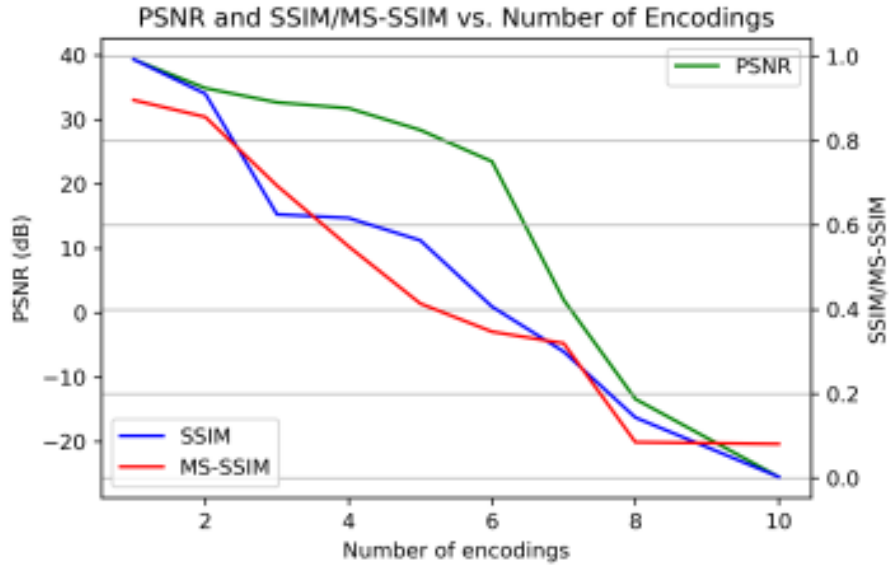


Figure 39 – PSNR, SSIM, and MS-SSIM variation for interlinked compression in offshore simulation.

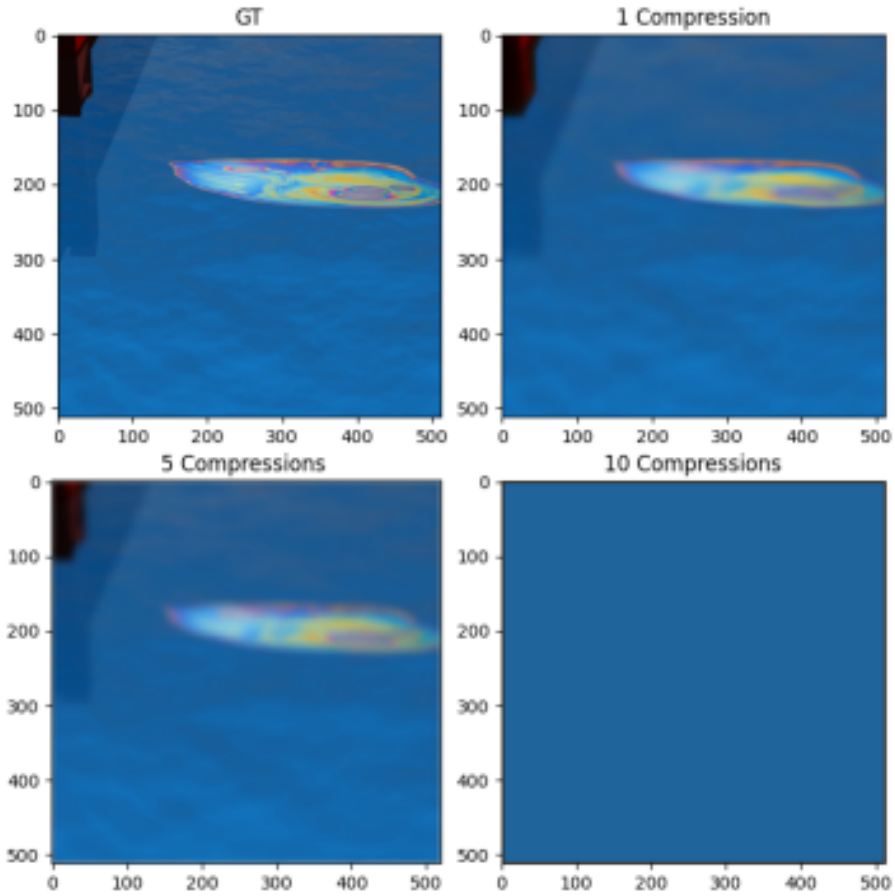


Figure 40 – Compression example of a picture of an oil spill found at the water surface.

this is that, in open space, the edge and fog agents don't exchange so much navigation data between themselves or the cloud, and besides, this type of message has smaller sizes. Also, the UAVs (mostly edge agents) demanded fewer requests for superior layers (fog or cloud) to cognitive reinforcement learning navigation decisions, dealing with events locally, which decreased streamed messages for group control, as shown in Figure 36.

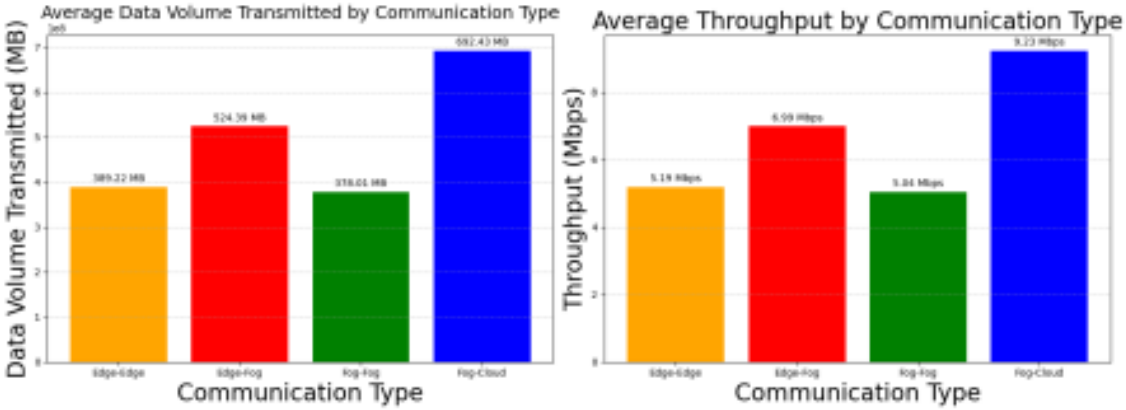


Figure 41 – Average ARCog-NET transmitted data and throughput in offshore simulation.

And finally, the packet loss and network availability results are presented in Figure 42.

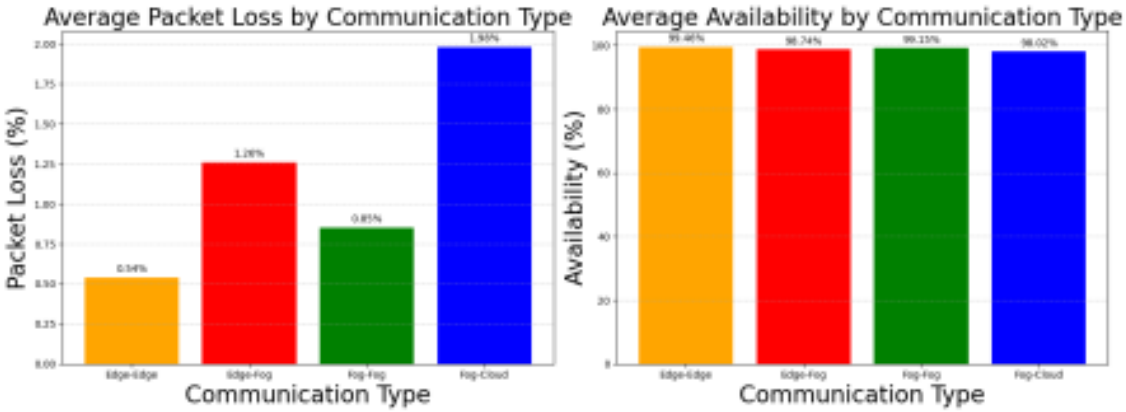


Figure 42 – Average ARCog-NET packet loss and throughput in offshore simulation.

The results demonstrate a clear trade-off between data volume, throughput, packet loss, and availability across the different communication types in ARCog-NET. Edge-to-edge communication exhibits the lowest data volume and throughput, which aligns with its lower latency and energy consumption, as shown in the latency and energy graphs. This makes edge-to-edge communication ideal for localized, low-latency tasks such as formation control and obstacle avoidance, where online coordination is critical. However,

its limited throughput and higher packet loss (compared to edge-to-fog) suggest it is less suitable for data-intensive tasks. The edge-fog communication shows moderate data volume and throughput, balancing energy efficiency and latency, making it suitable for tasks requiring both computational offloading and parallel responsiveness, such as role changes and data compression.

Fog-to-fog communication demonstrates higher data volume and throughput, with slightly increased latency and energy consumption compared to edge-to-edge and edge-fog. This makes it effective for tasks requiring significant data exchange between UAVs, such as collaborative path planning or shared mission updates. However, its higher packet loss and lower availability compared to edge-to-edge highlight potential reliability challenges in dynamic environments. fog-to-cloud communication exhibits the highest data volume and throughput, supporting large-scale data aggregation and processing, such as mission analytics or global task reallocation. However, its significantly higher latency and energy consumption, as shown in the latency and energy graphs, make it less suitable for time-sensitive or energy-constrained operation.

In this open, obstacle-free scenario covering a large area, ARCog-NET effectively leverages the hierarchical architecture to distribute tasks based on their requirements. The edge-to-edge communication is prioritized for online, low-latency tasks, while fog-to-cloud communication handles data-intensive, non-time-critical tasks. The intermediate edge-fog and fog-to-fog layers provide a balance, enabling efficient resource utilization and scalability. The results highlight ARCog-NET's adaptability in managing diverse task requirements, ensuring optimal performance across latency, energy, data volume, and reliability metrics. However, the higher packet loss and lower availability in fog-to-fog and fog-to-cloud communications suggest the need for robust error correction and redundancy mechanisms to enhance reliability in large-scale deployments. Overall, ARCog-NET demonstrates strong potential for open-environment applications, offering a scalable and efficient framework for UAV swarm coordination and task allocation.

4.3- Environment 3: Provide mobile phone coverage to a neighborhood

In this scenario, only graphic simulations with groups of 10 UAVs were executed, being 10 simulations in total, and another 1000 simulations from 1 to 1000 vehicles without a graphic interface to test ARCog-NET scalability. The graphic simulation represented a city region with a critical mobile communication problem, and the UAV swarm was deployed to act as communication towers to allow client connection while the main system is restored. This scenario represents an outdoor partially known scenario, as the ARCog-NET needs to position the UAV groups strategically to provide service to clients and not collide between themselves or other city structures like buildings or trees. No connection aspects were studied in this section, as only ARCog-NET behavior is the subject of this research. The focus is on how the UAVs cover the client demand areas and how quickly they process events of repositioning and flying to new points. Figure 43 shows the points occupied by each UAV and the cluster coverage of these points throughout the simulation time in 2D.

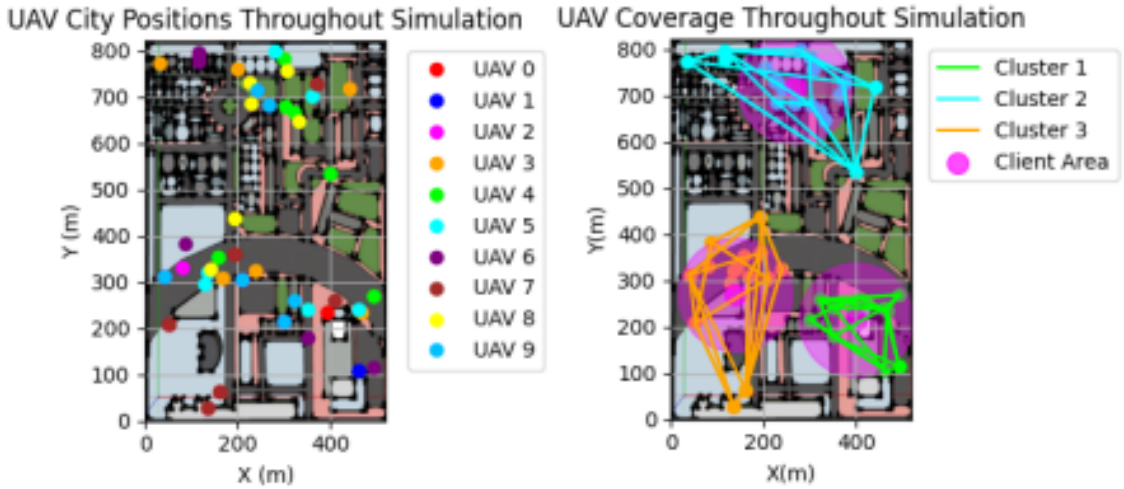


Figure 43 – 2D points occupied by each UAV in city simulation and cluster coverage.

In the same way, Figure 44 shows the points occupied by each UAV and the cluster coverage of these points throughout the simulation time in 3D. The client concentration hotspots are represented within circular areas that are randomly placed in the neighborhood and change over time throughout the simulation. The UAVs then adapt their positions to guarantee continuous operation. In this simulation, it's supposed that each

UAV carries portable radio antennas to provide service to clients, and this equipment has no interference with the ARCog-NET hardware (it is only powered by the UAVs) and also is not used by the framework. A video with a simulation example of this scenario is presented in Ramos (2024c).

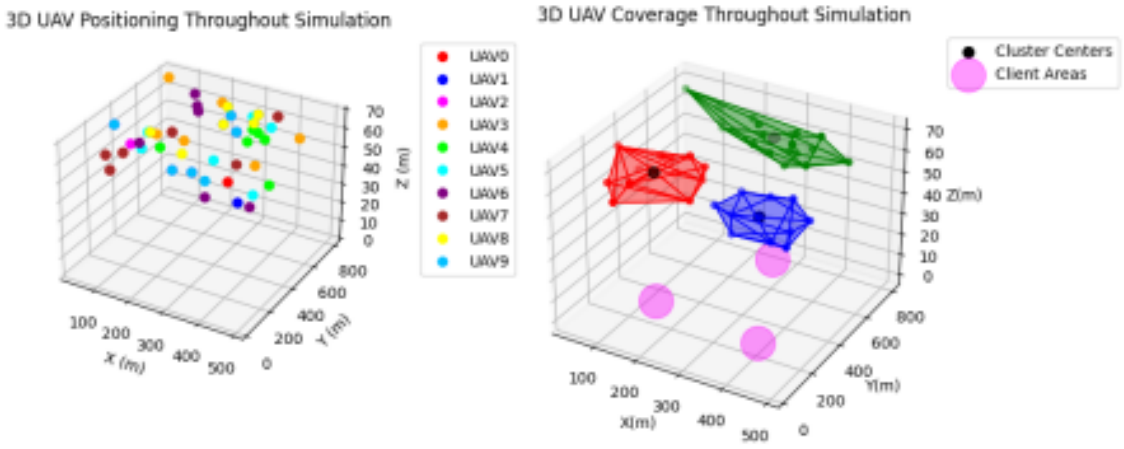


Figure 44 – 3D points occupied by each UAV in city simulation and cluster coverage.

The UAV group deploys from a fixed base and positions itself, one by one, at a point in the sky. The group's position is coordinated by fog or cloud nodes, but the flight trajectories to change points can be processed by any layer (edge, fog, or cloud). When a client concentration demand shifts within the marked areas, an event will be created by a group serving that area, and the UAVs will change their positions. The client demand variation is perceived by fog or cloud layers through information passed by the edge nodes. Figures 45 and 46 represent variations in X and Y coordinate setpoints, respectively, and how each UAV responds to that change.

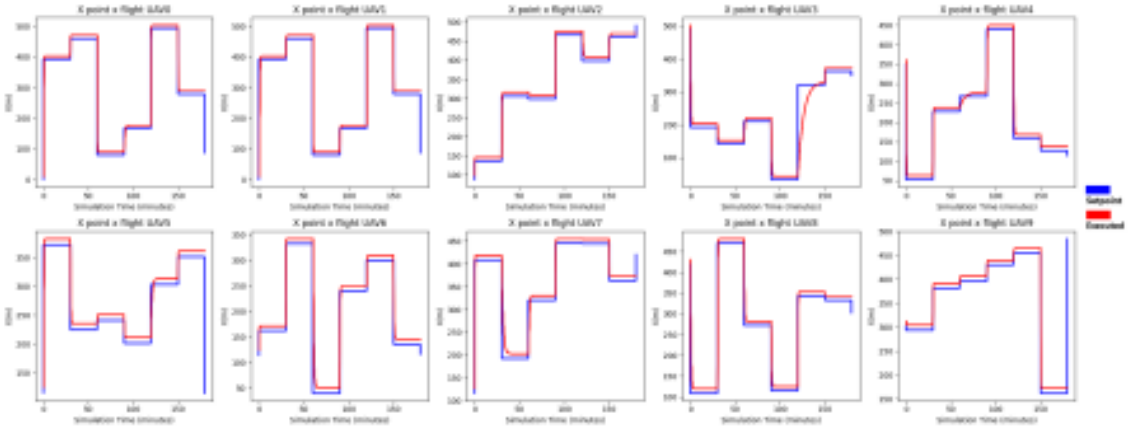


Figure 45 – Response of each UAV to X coordinate setpoint variation in city simulation.

A UAV can change its fog coordinator over time, belonging to another cluster for a period and then shifting it. This decision is based on how close the UAV gets to a fog coordinator. An edge agent can become a fog coordinator and vice-versa, which is a cloud-layer decision to change the network's structure on demand. However, the number of fog clusters is always 3 in all executed simulations with graphics enabled.

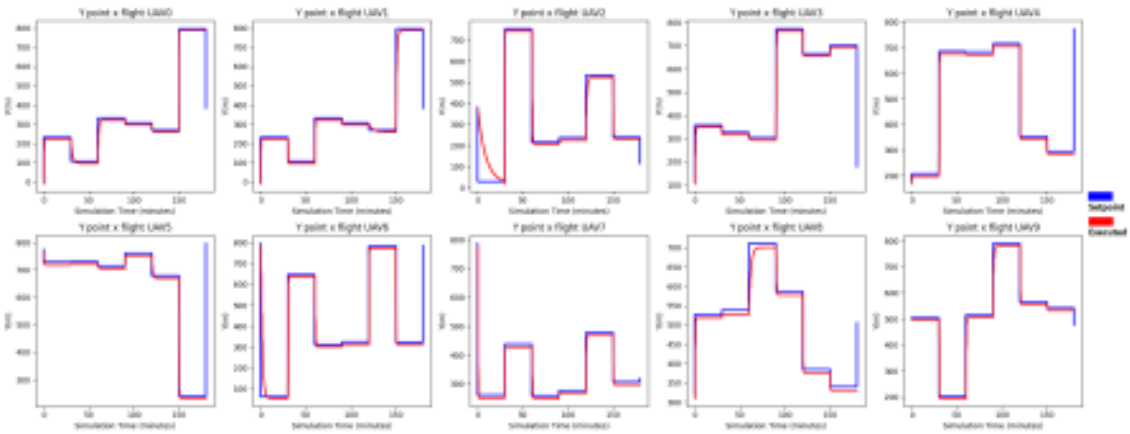


Figure 46 – Response of each UAV to Y coordinate setpoint variation in city simulation.

Table 12 represents the Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) for each executed setpoint, respectively. Given the coverage plot, the response to setpoint shifts, and the error table, it's possible to notice that the ARCog-NET could provide a quick response to environment changes and keep clients covered in this simulation scenario.

Table 12 – Comparison of MAE and RMSE for X and Y coordinates for each UAV.

UAV TAG	X MAE	X RMSE	Y MAE	Y RMSE
0	0.10	0.15	0.05	0.10
1	0.12	0.16	0.06	0.11
2	0.14	0.18	0.07	0.12
3	0.13	0.17	0.06	0.11
4	0.11	0.16	0.05	0.10
5	0.10	0.15	0.05	0.10
6	0.12	0.16	0.06	0.11
7	0.13	0.17	0.07	0.12
8	0.11	0.16	0.05	0.10
9	0.10	0.15	0.05	0.10

With small positioning errors, quick response time, and a client area coverage perception, the swarm behavior presented good performance for such kind of application. In terms of network architecture behavior, fewer decisions were processed on average, as the UAVs weren't flying continuously but statically positioned at a point and only flying

to another point to attend to clients eventually. As the cloud layer has information on all data traffic of clients measured from edge agents through fog coordinators, this layer is responsible for most of the repositioning decisions at the beginning of the mission, and fog cluster control is the main decision-maker as the cognitive knowledge evolves. These results can be noted in Figure 47.

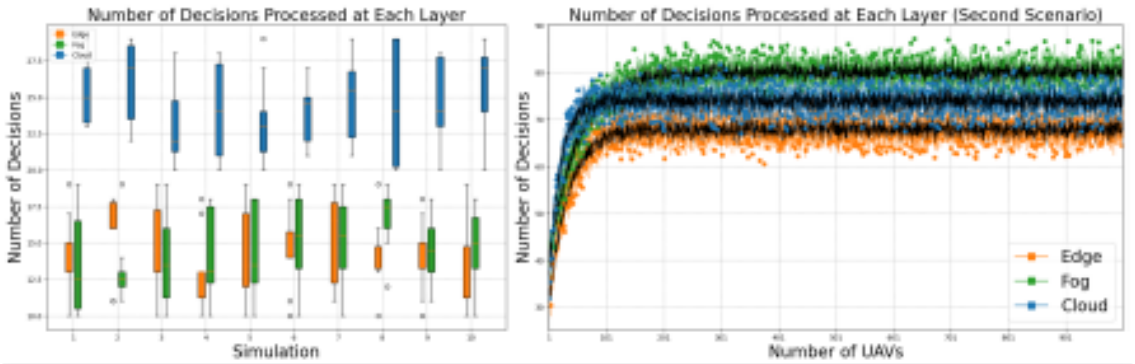


Figure 47 – Number of decisions processed on each ARCog-NET layer in the city simulation.

In this scenario, even fewer decisions have to be processed. ARCog-NET presented the saturation behavior one more time as more network nodes were connected to the architecture. As the coverage fault is identified by the cloud server, most of the decisions are initially made there. As the connected nodes increase, the cognitive task allocation makes it easier for fog nodes to perceive the communication fault and reallocate their edge group without messaging the cloud server or HITL. Another interesting asset of Figures 45 and 46 and this decision result is the X and Y coordinates, revealing that probably the formation control cognitive sub-processing was triggered more times, which is indeed confirmed by Figure 48.

In fact, note that formation control is the third most allocated task type. This makes sense for the scenario as, once again, ARCog-NET is dealing with a wide open environment with obstacles apart from each other. As client demand is geographically and randomly set by the algorithm and changes over time, the network will trigger a formation to better answer connectivity calls, and groups will tend to displace themselves as a whole instead of calculating separate routes, making formation control a more logical decision than path planning. This scenario almost doesn't use image compression, as the sent images are probably initial data transmission decisions taken by the network transient while it still learns how to behave to better serve its configured mission purpose of covering

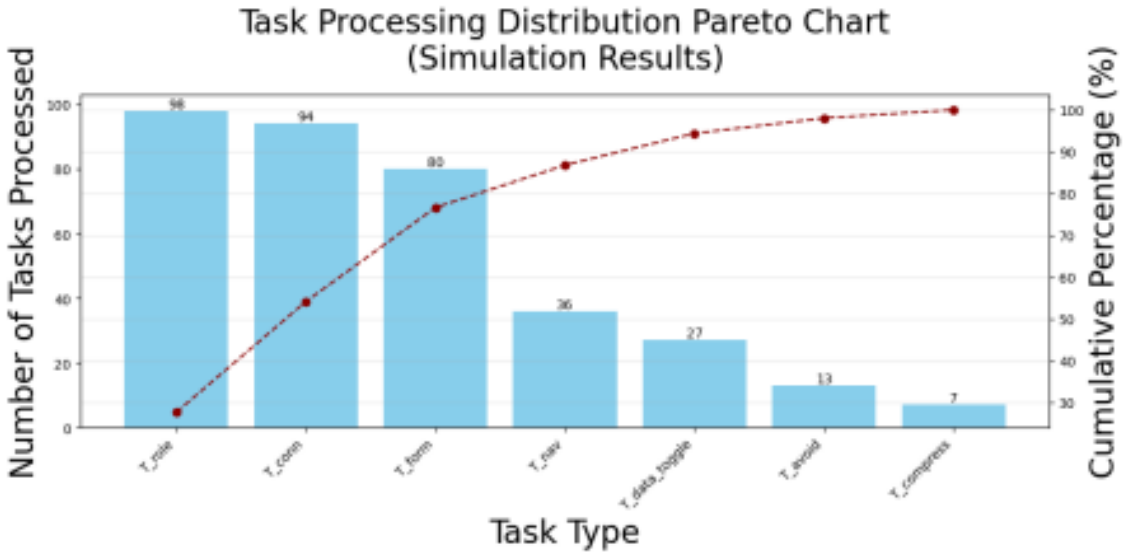


Figure 48 – Pareto Graph of the average number of tasks by type in the city simulation.

a deficient mobile communication network. The task allocation overall efficiency in this scenario was also evaluated and presented in Table 13.

Table 13 – Task Allocation Comparison in city scenario.

Metric	**Market-Based**	**Heuristic** Methods	**Bio-Inspired** Methods	**Reinforcement** Learning	**ARCog-NET**
Efficiency (%)	70	80	85	88	87
Energy Consumption (J/task)	130	110	100	95	159
Communication Overhead (%)	18	12	10	9	8
Adaptability (ms)	65	75	80	82	81
Scalability (number of UAVs)	400	900	850	750	1000

The task allocation comparison table highlights ARCog-NET's competitive performance in urban scenarios. While ARCog-NET achieves 87% efficiency, closely matching reinforcement learning (88%) and bio-inspired methods (85%), it outperforms market-based (70%) and heuristic approaches (80%). However, ARCog-NET's energy consumption (159 J/task) is significantly higher than all other methods, including reinforcement learning (95 J/task) and bio-inspired methods (100 J/task), indicating a trade-off between efficiency and energy use. ARCog-NET excels in communication overhead (8%), outperforming all other methods, which is crucial for scalability in dense urban environments. Its adaptability (81 ms) is slightly below reinforcement learning (82 ms) but better than market-based (65 ms) and heuristic methods (75 ms). Finally, ARCog-NET supports 1000 UAVs,

outperforming all other methods, including bio-inspired (850 UAVs) and reinforcement learning (750 UAVs), demonstrating superior scalability for large-scale urban deployments. Overall, ARCog-NET provides a robust solution for task allocation in urban scenarios with mission requirements constantly changing (in this case, the traffic demand), balancing efficiency, communication efficiency, and scalability, though its higher energy consumption warrants further optimization.

Table 14 – Path Planning Comparison in city scenario.

Metric	Classical PSO	A*	Dijkstra	RRT	ARCog-NET
Convergence Time (s)	14	9	11	17	11
Trajectory Coverage (%)	83	88	90	85	89
Computational Cost (ms)	160	210	190	130	140
Adaptability (ms)	72	68	63	78	75
Scalability (number of UAVs)	480	290	390	570	1000

Table 15 – Formation Control Comparison in city scenario.

Metric	DMPC	DRL-Based	Consensus -Based	ARCog-NET
Formation Error (m)	0.13	0.18	0.08	0.10
Energy Consumption (J/formation)	48	58	38	74
Adaptability (ms)	82	87	77	80
Communication Overhead (%)	10	13	7	8
Robustness to Delays (%)	87	82	91	90

The path planning and formation control results are presented in Tables 14 and 15, respectively. Table 14 shows that ARCog-NET maintains a competitive balance between efficiency and adaptability, outperforming traditional methods in scalability (1000 UAVs) while remaining close to Dijkstra in trajectory coverage (89% vs. 90%). However, its convergence time (12 s) is slightly slower than A* (10 s) and Dijkstra (12 s), though still faster than RRT (18 s). The computational cost (140 ms) remains moderate, offering better performance than A* (210 ms) and Dijkstra (190 ms), but slightly higher than RRT (130 ms). Adaptability (75 ms) is improved over Dijkstra (63 ms) and A* (68 ms), though still behind RRT (78 ms), indicating that while ARCog-NET offers robust decision-making, additional refinements may enhance online responsiveness.

Table 15 indicates notable improvements in ARCog-NET's ability to maintain formations with a low formation error (0.10 m), outperforming DMPC (0.13 m) and DRL-based control (0.18 m), while closely approaching consensus-based methods (0.08 m). However, energy consumption (74 J/formation) is significantly higher than other methods, suggesting that while ARCog-NET ensures high precision in formation keeping, it does so at a higher energy cost. Despite this, adaptability (80 ms) remains close to DRL-based methods (87 ms) and better than consensus-based control (77 ms), indicating improved flexibility in varying demand environments. The communication overhead (8%) is efficiently minimized, second only to consensus-based methods (7%), while robustness to delays (90%) remains high, comparable to consensus-based methods (91%) and significantly stronger than DRL-based approaches (82%). This suggests that ARCog-NET provides strong formation control capabilities with minor trade-offs in energy efficiency, making it a promising solution for real-world multi-UAV operation with mission objectives constantly changing.

Less decisions (task allocation) and the lack of image data transmission, therefore compression processing time, also resulted in less latency and better connection scalability results. There is an ARCog-NET improvement associated with energy trade-off identified in simulations for this scenario, which is in line with the results presented in Figure 49. It is possible to verify that latency has low values and the increasing rate of latency as more robots are added to simulations is below what was seen in the offshore scenario, with a more linear latency increase, always keeping smaller rates. Energy has higher values, but in this simulation, the energy consumption was also increased by the substitute radios of the transmission towers that the UAVs are carrying.

Data volume was smaller than observed in the offshore simulation, while throughput followed very close to the same values compared to the same previous scenario. This is a logical result as less data is transmitted over simulation, and the architecture kept is the topology. An interesting result was the fog-to-cloud throughput decrease, which is in line with mission evolutions as a cognitive process in this simulation, making edge and fog less dependent on the cloud server and HITL. These results can be observed in Figure 50.

The packet loss and availability closely follow what was observed in the offshore simulation with slight improvements, which is a logical result since the architecture topology was not changed. Such results can be observed in Figure 51

In the city simulation, ARCog-NET successfully managed UAV swarms for urban

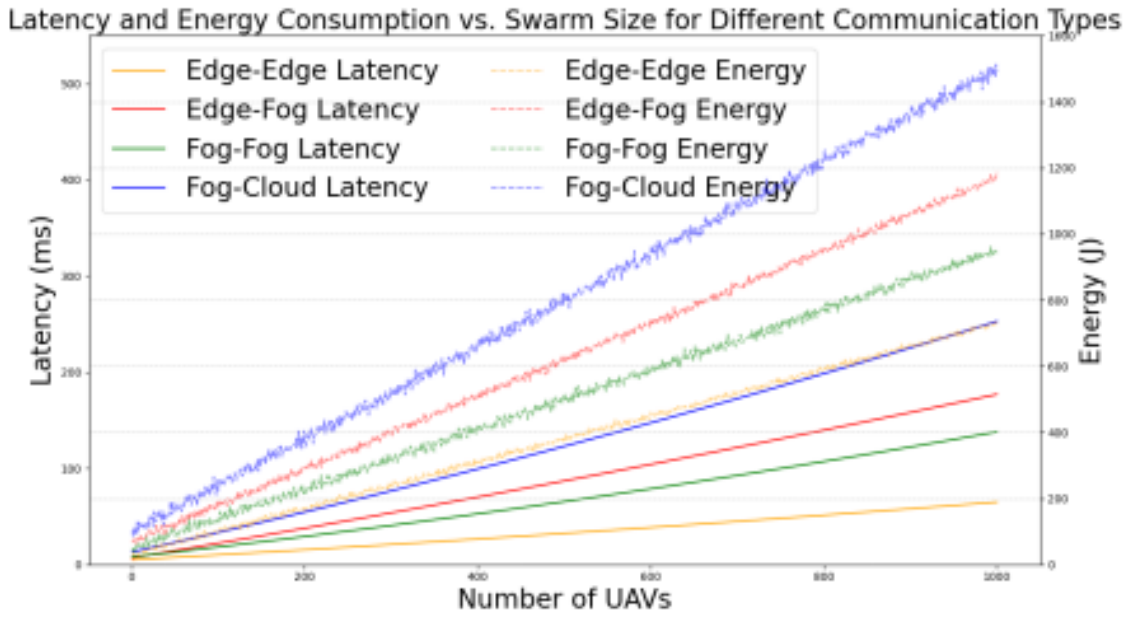


Figure 49 – Average ARCog-NET latency by number of UAVs at each time in the city simulation.

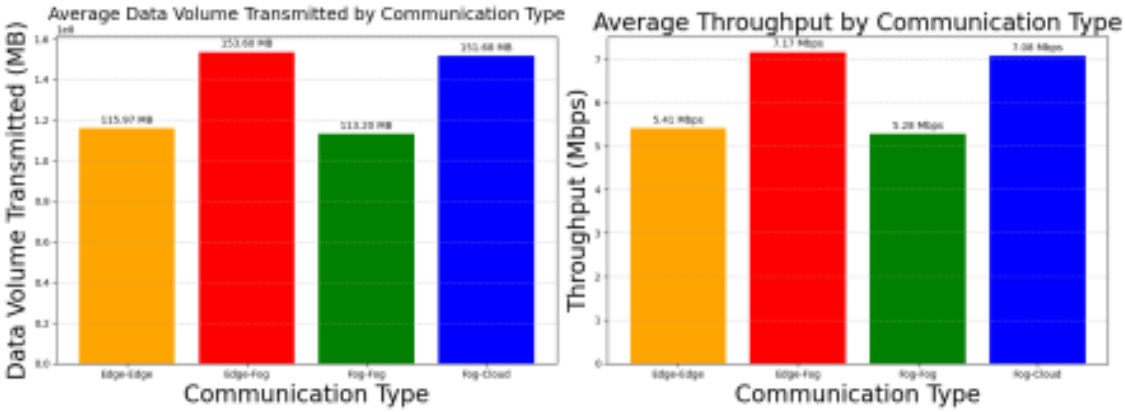


Figure 50 – Average ARCog-NET transmitted data and throughput in city simulation.

communication restoration. The system effectively adapted to dynamic client demand hotspots, ensuring continuous coverage with minimal positioning errors and quick response times. The fog layer primarily handled the decision-making process, with fewer decisions required overall due to the static positioning of UAVs, which only repositioned as needed. This led to efficient task allocation and formation control, though at a higher energy cost than other methods.

The simulation highlighted ARCog-NET's scalability, supporting up to 1000 UAVs, and its ability to maintain low communication overhead and latency, even as the number of UAVs increased. The system's adaptability and robustness were evident, with formation

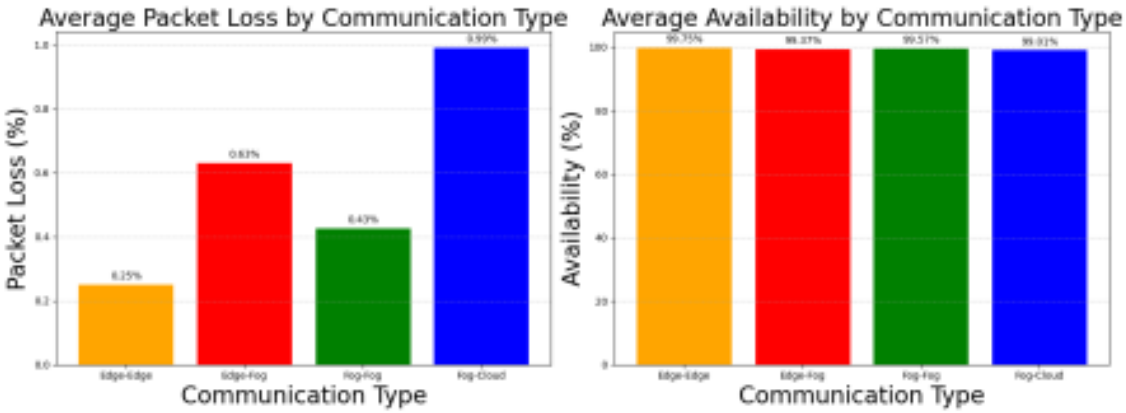


Figure 51 – Average ARCog-NET packet loss and throughput in city simulation.

control and path planning tasks being efficiently managed, though energy consumption remains an area for improvement. Overall, ARCog-NET proved to be a reliable solution for urban scenarios with dynamic mission requirements, balancing efficiency, scalability, and adaptability effectively.

4.4- Environment 4: Industrial plant indoor search and rescue

Once again, graphic simulations were carried out with one to ten vehicles, ten times for each count. In this scenario, the proposal is to emulate an indoor search and rescue mission where the UAVs connected to ARCog-NET should fly inside an industrial environment undergoing some emergency situation (a radiation breakout, for example), looking at all rooms and spaces for victims. The group deployed from a fixed safe point and then started flying around using ultrasonic collision sensors to avoid hitting structures or each other while searching for humans with cameras. The video in Ramos (2024d) shows an example of this mission with 10 UAVs, and Figure 52 shows the executed trajectories in this example.

An interesting result of this application is that robots can "learn" about positioning even if they didn't pass by an unknown region, because if another member of the network scanned a determined area and already gained a solution for continuing to fly, these flight decisions will be saved in the knowledge base of ARCog-NET. The initial decision is to avoid obstacles by flying a straight line if an object is detected too close to the respective

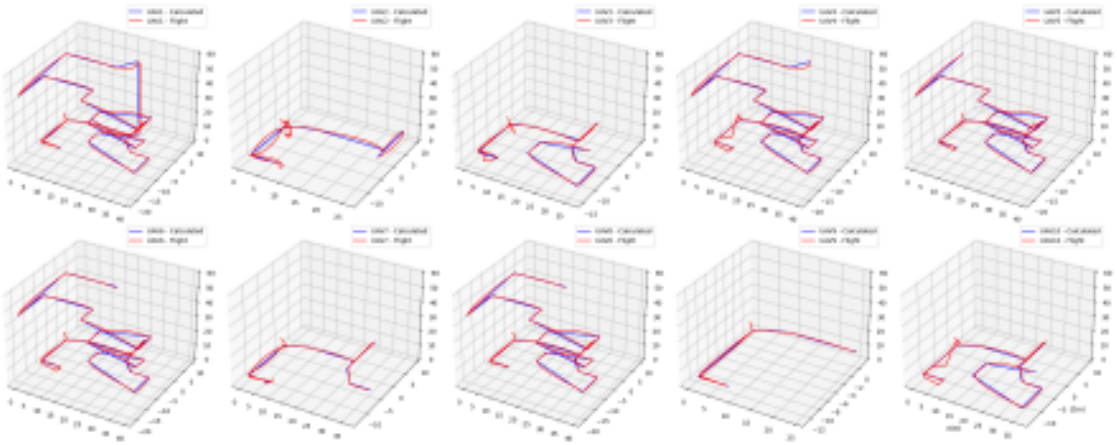


Figure 52 – Comparison between trajectories calculated by ARCog-NET and executed by each UAV for a simulation with 10 robots in an industrial plant environment.

UAV. The cloud HITL can also send manual commands to any robot, an edge agent, or a fog coordinator. Figures 53 and 54 represent how the robots reacted to variations in X and Y points of the trajectory inside the factory for a simulation with 10 robots.

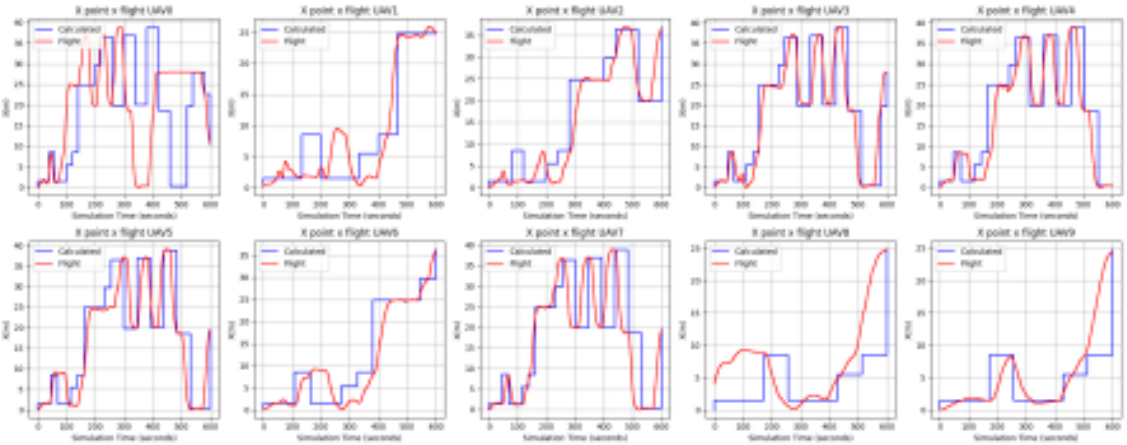


Figure 53 – Response of each UAV to X coordinate setpoint variation in industrial plant simulation.

Note in Figures 53 and 54 that, in some cases, the flight begins before the trajectory calculation changes. This is an example of how a knowledge base is shared between robots, as a robot can make a flight decision by communicating to the network and reacting faster to environmental events. Usually, the UAV behaves as an edge agent in this case and sends a request to the fog coordinator, which delivers a trajectory orientation quicker than the local trajectory PSO loop that the agent runs. Sometimes, the curve is delayed, where the UAV changes its trajectory after the calculations. In this case, the agent is probably passing by a region without mapping in the knowledge base, or maybe

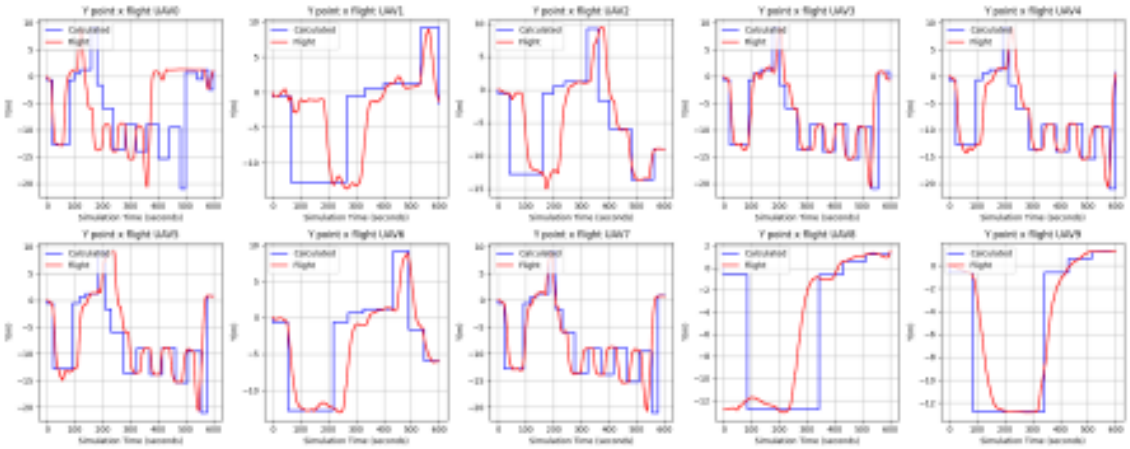


Figure 54 – Response of each UAV to Y coordinate setpoint variation in industrial plant simulation.

there is a communication delay with superior layers so it decreases speed while the results are not delivered. Usually, the robot is a group leader (or a fog coordinator) in this case, and sometimes, requests to the cloud are sent for trajectory orientation. The same flight curve (red) can show a positive or negative time shift when compared to the calculated trajectory (blue) on the same graph, which also represents that the respective UAV probably changed roles during the simulation (it was an edge agent and became a fog coordinator, for example), the cognitive knowledge base increased (making more possible to make even quicker decisions) or the communication latency in that region, packet loss and throughput are severe. Table 16 sums up the average X and Y positioning MAE and RMSE, and average time shifts as well.

Table 16 – MAE and RMSE for X and Y trajectories, and Time Shift for UAVs.

UAV	X MAE	X RMSE	Y MAE	Y RMSE	Average Time Shift (s)
1	0.05	0.03	0.08	0.12	-75.72
2	0.04	0.09	0.07	0.11	63.38
3	0.03	0.08	0.06	0.1	44.92
4	0.04	0.09	0.05	0.09	13.21
5	0.03	0.08	0.07	0.1	23.23
6	0.04	0.1	0.08	0.12	11.38
7	0.03	0.09	0.06	0.11	41.11
8	0.05	0.1	0.07	0.11	-17.70
9	0.04	0.09	0.06	0.1	-77.03
10	0.03	0.08	0.08	0.12	41.15

This results shows satisfactorily how the proposed architecture can be used as a effective UAV swarm control mechanism while is also a good solution to manage communication and data transmission in missions. As in the three previous scenarios, the

decision-making study was also conducted as presented in Figure 55.

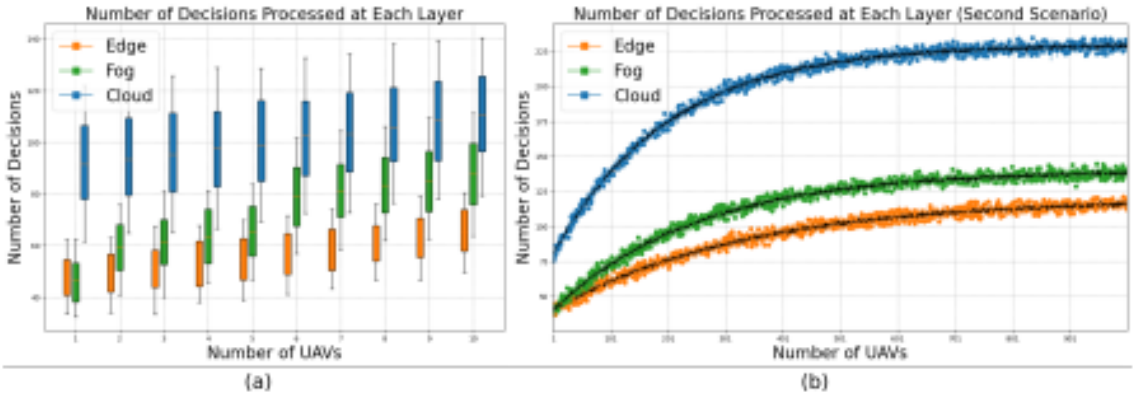


Figure 55 – Number of decisions processed on each ARCog-NET layer in all simulations of the industrial plant environment.

In this indoor search and rescue scenario, the decision-making graphs take more time to saturate because the complexity of the environment and mission demands a higher volume of decisions as more UAVs are added. Unlike the previous urban and offshore scenarios, where UAVs were mostly statically positioned, this mission requires continuous dynamic navigation, obstacle avoidance, and victim detection in a confined, unstructured space. Each additional UAV introduces new sensory data, path planning requirements, and coordination tasks, leading to a logarithmic increase in decisions. The indoor environment's unpredictability, combined with the need for online adjustments to avoid collisions and optimize search patterns, ensures that decision-making scales with the number of UAVs. Furthermore, the cognitive allocation of tasks across edge, fog, and cloud layers allows the system to handle this complexity efficiently, delaying saturation and maintaining performance even as more UAVs are deployed. This behavior reflects ARCog-NET's adaptability to mission-specific demands, where increased decision-making is necessary to ensure effective operation in challenging environments.

The graph indicates that the fog layer is critical in task handling within the ARCog-NET architecture for this indoor application. The high median and wide Interquartile Range (IQR) for the fog layer suggest it handles a significant portion of the computational load and experiences a wide range of decision-processing activities. This could be due to its position in the network hierarchy, balancing between the high-level, computationally intensive tasks offloaded to the cloud and the low-latency, online tasks managed at the edge layer (note that this application is the most active in terms of communication due to the high amount of obstacles and unknown indoor environment). The edge layer's lower

median number of decisions and lower variability indicate it primarily handles less complex, online tasks that do not require intensive computation, which is in line with agent roles in this mission that are mainly charged for navigation and image transmission to the cloud through fog. This is consistent with the role of edge computing, which aims to reduce latency by processing data closer to the source.

The cloud server, while handling more decisions than the edge agents and fog coordinators, doesn't saturate as quickly as other scenarios. The moderate variability and presence of outliers suggest that the cloud layer processes tasks with varying computational demands, possibly involving more complex analytics and long-term storage tasks, including HITL supervision as well, a result confirmed by the Pareto Graph (see Figure 56). Being a small indoor scenario, this makes sense, as navigation tasks are primarily handled by the robots themselves, while the cloud layer is responsible for most of the image processing and target identification. In summary, fog coordinators are the most active in terms of flight decision processing and allocation, but the cloud layer is the one that processes the most decisions. This distribution aligns with the typical roles and responsibilities assigned to each layer in an EFC architecture, reflecting an efficient distribution of computational tasks to optimize performance and resource utilization.

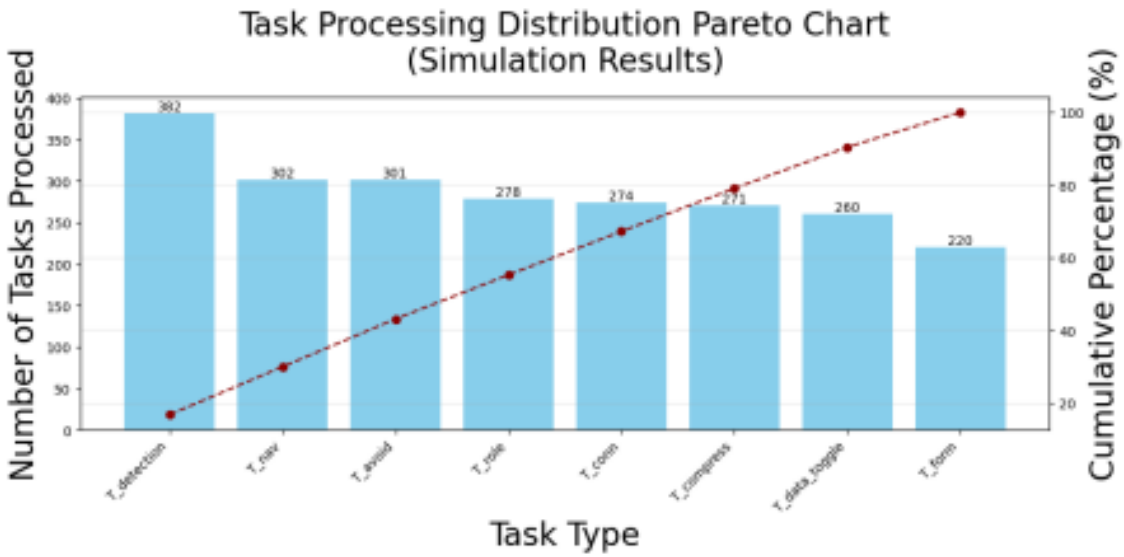


Figure 56 – Pareto graph of the average number of tasks by type in the industrial plant simulation.

In the industrial search and rescue scenario, navigation is a critical component of the mission, encompassing both path planning and formation control. Path planning ensures that each UAV can dynamically avoid obstacles and efficiently explore the environ-

ment, while formation control enables coordinated movement of the swarm to cover larger areas and maintain communication links. These capabilities are essential for navigating the complex, unstructured indoor environment, where obstacles are abundant and the mission requires precise, online adjustments. Integrating these navigation strategies allows the UAVs to operate autonomously, adapt to changing conditions, and ensure effective coverage of the search area.

The path planning comparison in Table 17 shows that ARCog-NET maintains a strong balance between convergence time (14 seconds) and trajectory coverage (87%). While it is slightly slower than A* (12 seconds), it outperforms RRT (20 seconds) and Classical PSO (18 seconds). The computational cost of 160 ms is competitive, being lower than A* (220 ms) and Dijkstra (200 ms), though slightly higher than RRT (150 ms). ARCog-NET's adaptability (72 ms) is better than Dijkstra (60 ms) and A* (65 ms), though slightly below RRT (75 ms). Notably, ARCog-NET supports up to 1000 UAVs, demonstrating superior scalability compared to other methods, making it highly suitable for large-scale indoor search and rescue missions.

Table 17 – Path Planning Comparison in Industrial Plant Scenario.

Metric	Classical PSO	A*	Dijkstra	RRT	ARCog-NET
Convergence Time (s)	18	12	15	20	14
Trajectory Coverage (%)	80	85	88	82	87
Computational Cost (ms)	180	220	200	150	160
Adaptability (ms)	70	65	60	75	72
Scalability (number of UAVs)	450	300	400	500	1000

Besides not being triggered too much, the formation control comparison in Table 18 highlights ARCog-NET's ability to maintain precise formations with a formation error of 0.12 meters, outperforming DMPC (0.15 meters) and DRL-based methods (0.20 meters), and closely approaching consensus-based methods (0.10 meters). However, energy consumption (70 J/formation) remains higher than other methods, indicating a trade-off between precision and energy efficiency. ARCog-NET's adaptability (78 ms) is competitive, close to DRL-based methods (85 ms), and better than consensus-based control (75 ms). Communication overhead (10%) is efficiently managed, though it is slightly higher than that of consensus-based methods (8%). Robustness to delays (88%) is high, comparable to consensus-based methods (90%), demonstrating ARCog-NET's reliability in dynamic,

obstacle-rich environments despite higher energy costs.

Table 18 – Formation Control Comparison in Industrial Plant Scenario.

Metric	**DMPC**	**DRL-Based**	**Consensus -Based**	**ARCog-NET**
Formation Error (m)	0.15	0.20	0.10	0.12
Energy Consumption (J/formation)	50	60	40	70
Adaptability (ms)	80	85	75	78
Communication Overhead (%)	12	15	8	10
Robustness to Delays (%)	85	80	90	88

The task allocation comparison in Table 19 demonstrates ARCog-NET's competitive performance in the industrial plant scenario. With an efficiency of 84%, ARCog-NET closely matches reinforcement learning (85%) and outperforms market-based (68%) and heuristic methods (78%). However, its energy consumption (150 J/task) is higher than all other methods, including reinforcement learning (100 J/task) and bio-inspired methods (110 J/task), indicating a trade-off between efficiency and energy use.

Table 19 – Task Allocation Comparison in Industrial Plant Scenario.

Metric	**Market-Based**	**Heuristic** Methods	**Bio-Inspired** Methods	**Reinforcement** Learning	**ARCog-NET**
Efficiency (%)	68	78	82	85	84
Energy Consumption (J/task)	140	120	110	100	150
Communication Overhead (%)	20	15	12	10	9
Adaptability (ms)	60	70	75	80	78
Scalability (number of UAVs)	350	800	780	700	1000

ARCog-NET excels in communication overhead (9%), outperforming all other methods, which is critical for scalability in complex indoor environments. Its adaptability (78%) is slightly below reinforcement learning (80%) but better than market-based (60%) and heuristic methods (70%). Notably, ARCog-NET supports up to 1000 UAVs, demonstrating superior scalability compared to other methods, including bio-inspired (780 UAVs) and reinforcement learning (700 UAVs). This makes ARCog-NET a robust solution for task allocation in dynamic, obstacle-rich environments, balancing efficiency, communication efficiency, and scalability, though its higher energy consumption remains an area for further optimization. Figure 57 presents the latency and energy consumption graph.

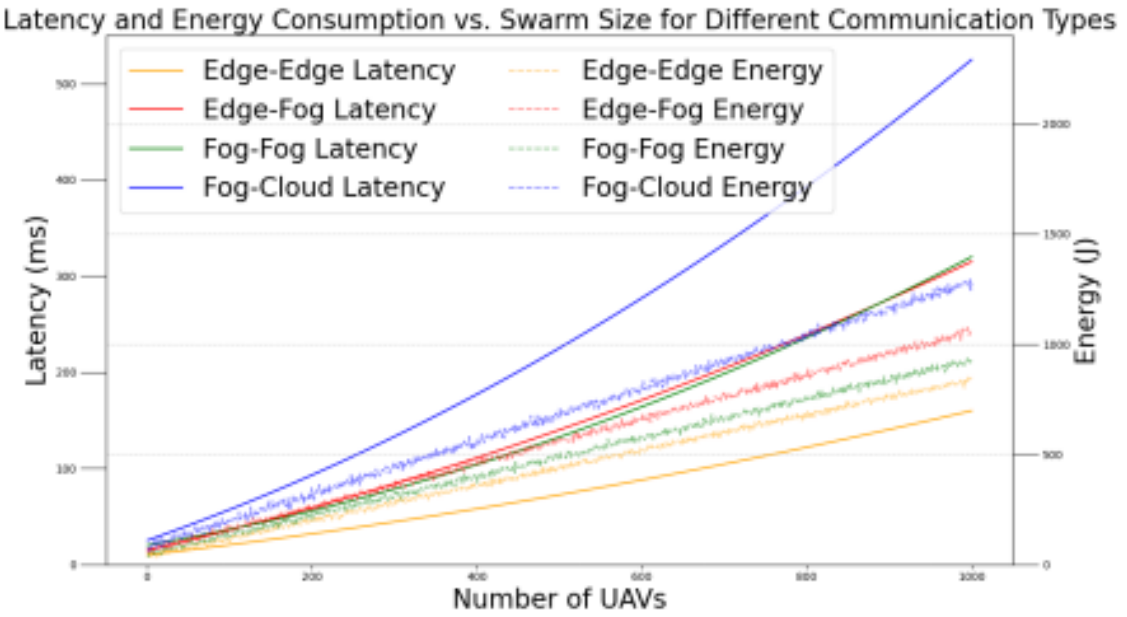


Figure 57 – Average ARCog-NET latency by number of UAVs at each time in the industrial plant simulation.

The indoor environment introduces unique challenges, such as confined spaces, dynamic obstacles, and the need for precise navigation, which significantly impact both latency and energy usage. As the number of UAVs increases, latency grows gradually, reflecting the efficient task allocation and communication management facilitated by ARCog-NET. The fog layer plays a critical role in reducing latency by handling most online flight decisions locally, while the cloud layer is primarily used for complex analytics and long-term storage tasks. This distribution ensures that latency remains manageable even for large swarms, though fog-to-cloud communication exhibits the highest latency due to the additional processing and transmission delays associated with cloud-based tasks. Edge-to-edge communication, on the other hand, offers the lowest latency but is less scalable for large swarms, making it suitable for smaller, localized tasks.

Energy consumption in this indoor scenario is notably higher due to the continuous dynamic navigation, obstacle avoidance, and victim detection tasks required in a confined and obstacle-rich environment. The fog layer's active role in decision-making contributes to increased energy usage, as it processes a significant portion of the computational load. fog-to-cloud communication consumes the most energy, driven by the computational intensity of cloud-based tasks and long-distance data transmission. Despite this, ARCog-NET's ability to distribute tasks across edge, fog, and cloud layers helps optimize energy usage, ensuring that energy consumption increases steadily and is more manageable as

the swarm size grows. The variability in energy consumption reflects the dynamic nature of the indoor environment, where energy usage can fluctuate based on mission requirements and environmental factors.

The trade-off between latency and energy consumption is evident in this scenario. Edge-to-edge communication provides low latency and energy consumption. Still, it is limited in scalability, while fog-to-cloud communication offers advanced computational capabilities and scalability at the cost of higher latency and energy usage. The fog layer strikes a balance between these extremes, providing a middle ground that ensures efficient task allocation and communication. ARCog-NET's scalability is a key strength, as both latency and energy consumption increase at manageable rates even for large swarms, making it well-suited for indoor missions that require hundreds or thousands of UAVs to cover complex environments effectively. Figure 58 presents the data volume and throughput graph.

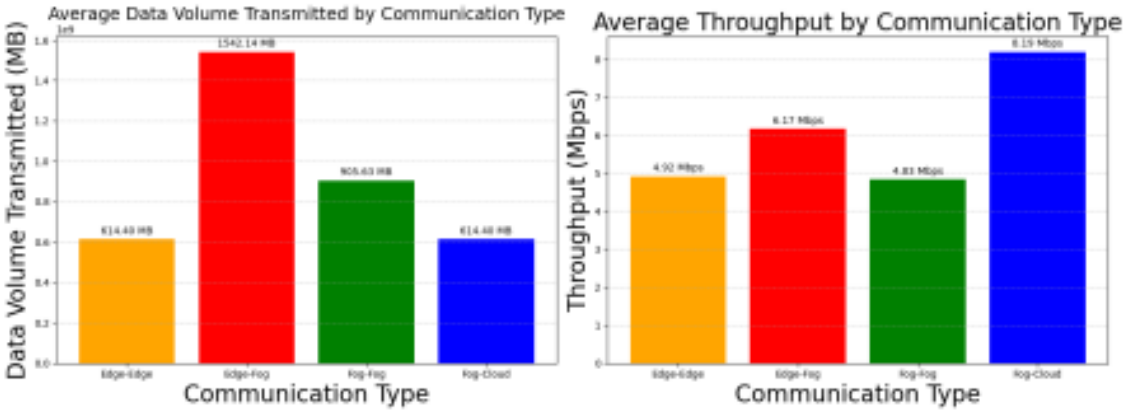


Figure 58 – Average ARCog-NET transmitted data and throughput in industrial plant simulation.

The high throughput in fog-to-cloud communication indicates that, while the cloud layer is used sparingly, it handles critical tasks that require significant data transmission. This ensures that complex analytics and long-term storage tasks are performed efficiently when needed. Edge-to-fog pipelines have high data volume but moderate throughput. This reflects the fog layer's role in handling a large amount of online data for decision-making, but with a focus on efficient, distributed processing rather than high-speed transmission. Edge-to-edge and fog-to-fog communication have low data volume and throughput, involving localized coordination and limited data transmission. This is consistent with the confined indoor environment, where direct communication between UAVs (edge-to-edge) and between fog nodes (fog-to-fog) is limited, but the nodes are usually relatively close to

each other.

The packet loss and availability results, presented in Figure 59, highlight the communication challenges in the industrial plant indoor environment. Edge-to-edge communication exhibits the highest packet loss (1.25%) and the lowest availability (98.75%), as direct UAV-to-UAV communication is prone to interference in confined spaces. In contrast, edge-fog and fog-to-fog communication show lower packet loss (0.75% each) and higher availability (99.25% each), reflecting the fog layer's effectiveness in managing online coordination. Fog-to-cloud communication has moderate packet loss (1.99%) and availability (98.01%), as long-distance transmission to the cloud layer is less reliable indoors. These results underscore the trade-offs between communication types in a dynamic and obstacle-rich environment.

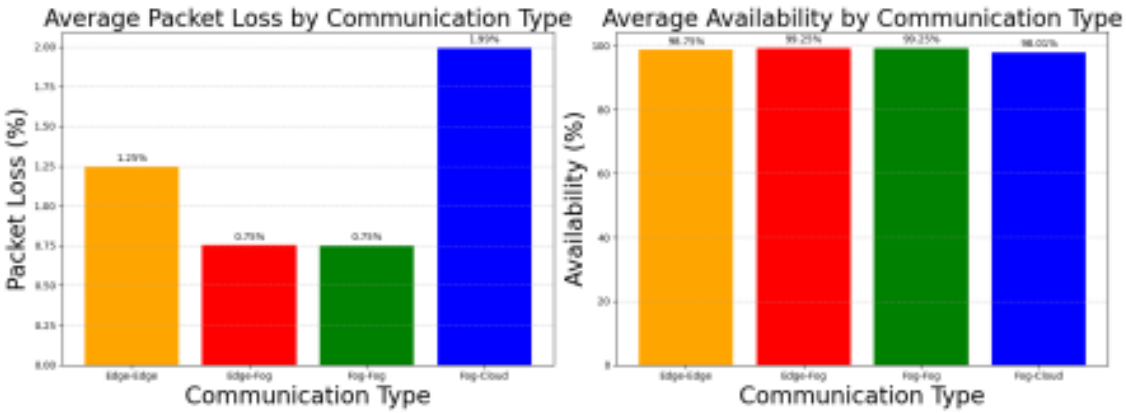


Figure 59 – Average ARCog-NET packet loss and throughput in industrial plant simulation.

In this scenario, the high packet loss in edge-to-edge communication limits its suitability for critical tasks. In contrast, the reliability of edge-fog and fog-to-fog communication demonstrates the fog layer's central role in online decision-making. The moderate packet loss in fog-to-cloud communication indicates that the cloud layer remains viable for complex analytics, albeit with reduced reliability. ARCog-NET's ability to distribute tasks across edge, fog, and cloud layers ensures robust performance, but further optimizations could focus on reducing packet loss in edge-to-edge communication to enhance swarm coordination in indoor environments.

Another factor that may be affected by latency, packet loss, throughput, and availability is the image data stream from edge and fog layers to the cloud human supervisors. Lots of frames are sent at each instant, so compression is an important factor in this fourth application. Figure 60 shows the signal-to-noise ratio and structural similarity between

original raw image data and successive image compression.

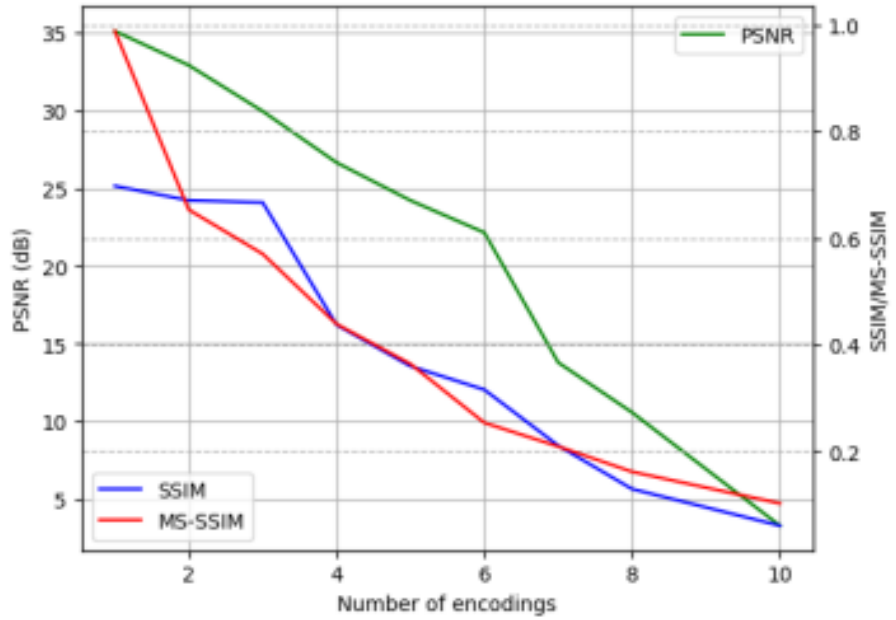


Figure 60 – PSNR, SSIM, and MS-SSIM variation for interlinked compression in industrial plant simulation.

The result is not much different from the other scenarios presented. The relationship between the metrics shows that with each additional encoding, there is a consistent degradation in both the signal-to-noise ratio and the structural similarity of the signal. The Peak Signal to Noise Ratio (PSNR), which measures the overall quality in terms of signal strength, shows a steady decline, while Structural Similarity (SSIM) and Mean Squared Structural Similarity (MS-SSIM), which are more perceptually oriented metrics, reflect how the structure and perceived quality deteriorate with each encoding. This graph highlights the impact of multiple encoding processes on signal quality, emphasizing that higher numbers of encoding lead to significantly reduced signal integrity and perceptual quality. The more the recorded image frame is compressed, the easier it becomes to stream the data. Still, it is more difficult for the human supervisor (or future automatic detection algorithms) to use this image for the mission's goals.

4.5- Collaborative indoor mapping experiment results

The simulation results of the indoor environment mission (4^{th} scenario) were used to perform the ablation studies regarding the impact of cognitive parameters on ARCog-NET's performance, which are presented in Figure 61.

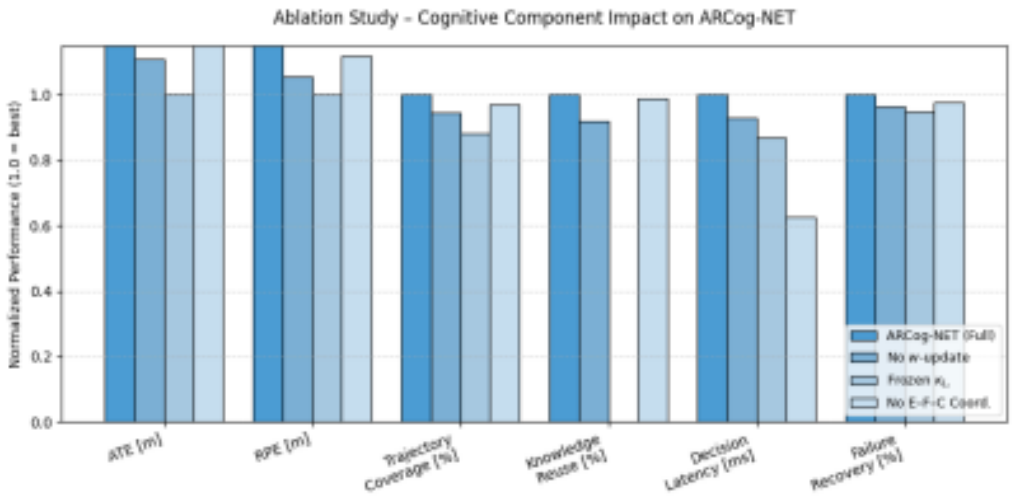


Figure 61 – Ablation study of the simulated ARCog-NET cognitive architecture.

The ablation analysis in Figure 61 demonstrates that the full ARCog-NET configuration outperforms all reduced variants across nearly every metric, confirming the synergistic contribution of its cognitive modules. The absence of decision-weight reinforcement (w-update) leads to moderate degradation in coverage and recovery, while freezing knowledge reuse (κ_{L_i}) causes the most significant drop in overall adaptability, indicating the centrality of experience sharing in cooperative learning. The removal of Edge–Fog–Cloud coordination mainly affects decision latency, validating the role of hierarchical communication in maintaining low network response times. Altogether, these results confirm that reinforcement learning, knowledge reuse, and multi-layer coordination jointly enhance the accuracy, resilience, and efficiency of ARCog-NET in dynamic swarm operations.

To provide a ground-truth reference for SLAM accuracy evaluation, all external and internal walls of the laboratory were precisely measured prior to the experiment, and the initial take-off positions of the six UAVs were marked relative to the lower-left corner of the environment reference frame. In this coordinate system, the global positive x-axis

extends to the right and the positive y-axis extends forward, perpendicularly to x-axis, from that origin. This setup allowed each UAV's estimated trajectory to be compared with its physically measured ground-truth displacement, enabling direct calculation of ATE and RPE values throughout the mission.

The experiment was executed as described in the previous section, was recorded on video (Ramos, 2025c), and analyzed on site. The UAV took flight from the initial positions, from points $\{(1.5, 1.5, 0)$, $(2.0, 1.5, 0)$, $(2.5, 1.5, 0)$, $(3.0, 1.5, 0)$, $(3.5, 1.5, 0)$, $(4.1, 1.5, 0)\}$, defined with reference to the lower-left corner of the laboratory and meticulously measured, for Tello $\#1$ to Tello $\#6$. The UAVs $\#3$ and $\#4$ were initially configured as fog layer nodes, with the first being the fog coordinator of the edge group composed of UAVs $\#1$ and $\#2$ and the second being the fog coordinator of the edge group composed of UAVs $\#5$ and $\#6$. The clusters changed configuration during mission (*e.g.*, an agent became a coordinator and vice-versa), but never changing elements between clusters and never having more than one coordinator per cluster. Figure 62 is a picture of the swarm taken before the test starts, and an image extracted from RQT Graph that illustrates the active ROS nodes and topics at the deployment is available on ARCog-NET's GitHub repository (Ramos, 2025b).



Figure 62 – Swarm picture before the take off.

The UAVs were able to travel and map the entire laboratory environment, forming the 3D point cloud of the whole environment by sharing only the image points, their yaw angles, and forward traveled distances, as presented in the video (Ramos, 2025c). The

swarm took 3 minutes and 15 seconds to scan all environment, totaling 93.98 m^2 in area, gathering 7006 points[1]

5- Conclusions

The research presented demonstrates that ARCog-NET successfully integrates cognitive decision-making with an EFC distributed computing architecture, addressing several critical gaps in the current UAV swarm literature. Unlike most state-of-the-art solutions that focused on isolated aspects such as task allocation, path planning, or formation control, ARCog-NET provides a complete and modular framework capable of dynamically adapting to diverse mission scenarios. The proposed architecture effectively combines decentralized decision-making, hierarchical computing, and cognitive processes, enabling UAV swarms to coordinate autonomously while balancing communication efficiency, scalability, and mission adaptability.

The simulations conducted across four representative environments, i.e., wind farm inspection, offshore environmental monitoring, urban communication restoration, and indoor search and rescue, highlighted ARCog-NET's versatility and robustness. Results consistently showed that the fog layer plays a central role in processing decisions, efficiently offloading edge computations and minimizing unnecessary reliance on cloud resources. This structure ensured low latency in time-sensitive tasks, while selectively leveraging cloud computing for global optimization and reinforcement learning updates.

Although ARCog-NET did not achieve the best numerical results in every benchmark compared to specialized state-of-the-art methods, it demonstrated strong overall performance across all evaluated metrics. For task allocation, it consistently achieved high efficiency and scalability, with minimal communication overhead, outperforming many reference techniques in network management. In path planning, ARCog-NET delivered superior adaptability and scalability, making it more suitable for dynamic, large-scale deployments. For formation control, it maintained low formation error and strong robustness to delays, at the cost of higher energy consumption—a trade-off directly linked to its cognitive processing overhead.

Moreover, ARCog-NET introduced features that are absent in existing frameworks, such as integrated data compression through a built-in autoencoder, dynamic role switching between edge and fog nodes, and the ability to reconfigure its network topology during missions. These contributions allow efficient handling of large volumes of streamed data,

optimize bandwidth utilization, and preserve critical information for mission success. The simulations also revealed ARCog-NET's scalability limits and trade-offs, particularly the balance between energy consumption and network performance, which provides valuable insights for future optimizations.

Experimentally validated, the framework was conceived to be generic and flexible, enabling adaptation across diverse domains, including mapping, inspection, search and rescue, and environmental monitoring. In this work, ARCog-NET was integrated with Monocular Visual SLAM and deployed in a real-world indoor mapping experiment using a swarm of six UAVs. The experiment demonstrated how ARCog-NET's cognitive structure—based on decision weight learning, historical knowledge reuse, and layered coordination—contributed to more efficient and robust swarm behavior. Key contributions of this experimentation work include the successful demonstration of distributed SLAM-based reconstruction using shared visual data, the application of reinforcement learning mechanisms to improve trajectory planning and formation control, and the implementation of cooperative knowledge fusion across Edge, Fog, and Cloud layers. The framework enabled online adaptation to dynamic conditions, efficient bandwidth use, and reduced human intervention through progressive autonomy.

In conclusion, ARCog-NET filled an important gap in the literature by offering a generic, reconfigurable, and programmable architecture for UAV swarm application development. Even when not outperforming all state-of-the-art algorithms in isolated metrics, it stands out as a comprehensive, flexible, and robust framework capable of supporting complex, real-world swarm operations. Its modularity enables seamless adaptation to different mission profiles, from static urban coverage to dynamic indoor search and rescue, making it a solid foundation for future developments in cognitive swarm robotics.

For future work, several improvements can be envisioned to enhance ARCog-NET's capabilities. First, optimization of energy consumption remains a key challenge, particularly in scenarios where high cognitive processing and frequent communication significantly impact UAV endurance. Incorporating energy-aware task allocation and adaptive power management could mitigate this limitation. Second, integration of advanced AI techniques, such as federated learning and graph neural networks, could further enhance distributed decision-making and improve the learning efficiency across large swarms without overloading communication channels. Third, the robustness to communication failures

in highly dynamic or interference-prone environments could be improved by embedding error correction mechanisms and redundancy strategies. Additionally, large-scale field experiments are the next essential steps to validate the framework under real-world constraints, including unpredictable environmental factors and hardware limitations. Finally, exploring heterogeneous swarms that combine UAVs with ground or maritime robots would extend ARCog-NET's applicability to multi-domain missions, further establishing its potential as a universal platform for autonomous swarm coordination.

6- Scientific production

The study conducted around the topic of data compression using deep learning models was published as a conference paper (Ramos et al., 2023) for the 2023 IEEE Latin American Robotics Symposium (LARS). The paper focused on evaluating the implementation and use of versatile data compression models based on deep learning autoencoders and its behavior within the edge-fog-cloud aerial robot swarm framework developed with ROS and simulated with Gazebo. The results showed that this compression methodology is a good choice for improving the system's performance without losing too much information.

The main ARCog-NET developed packages have been patented as a computer software, receiving the certificate $BR512024001222-2$ emitted by the National Institute of Industrial Property of the Federal Brazilian Government at the issue 2781 of the weekly Patent Magazine in April 24, 2024 (Ramos and Pinto, 2024).

The initial version of ARCog-NET, a cognitive architecture for UAV swarms that integrates perception, planning, decision-making, and learning, presented as qualifying phase of this thesis, was published in journal IEEE Xplore and it is available in Ramos et al. (2024). Using the edge-fog-cloud model, it distributed computation across robots (edge), intermediated supervisors (fog), and cloud servers. Implemented with ROS and Gazebo, ARCog-NET was tested in a wind farm inspection scenario, showing improved trajectory planning, collision avoidance, and data processing, resulting in lower latency, higher throughput, and enhanced operational efficiency.

The evolution of ARCog-NET covering more aspects as Task Allocation, Formation Control and the improved Path Planning algorithm was published in Ramos et al. (2025b). A SITL-based simulation with real flight firmware validated ARCog-NET performance in monitoring, SAR, and emergency communication scenarios. Results showed improved energy efficiency, adaptability, and operational effectiveness, bridging simulation and real-world UAV swarm applications.

The extended improvements in SLAM within the architecture alongside experimental results were accepted for publication in Scientific Reports, a journal from the Nature Portfolio. The paper is not yet published as of this date, appearing only as

a preprint in Research Square ("*Gabryel Ramos, Milena Pinto, Fabio Andrade et al. Implementation of Monocular Visual SLAM with ARCog-NET for Aerial Robot Swarm Indoor Mapping, 10 October 2025, PREPRINT (Version 1) available at Research Square [https://doi.org/10.21203/rs.3.rs-7573197/v1]*") (Ramos et al., 2025a). Building upon the core findings of the thesis, this work consolidates the experimental validation of ARCog-NET with monocular SLAM, presenting quantitative benchmarking, ablation studies, scalability analyses, and full 3D reconstruction results from real indoor multi-UAV deployments. The acceptance of this study in a high-impact journal underscores both the novelty and scientific relevance of the proposed architecture, as well as its contribution to advancing distributed cognition, cooperative SLAM, and autonomous swarm navigation in GPS-denied environments.

Bibliography

Abdelkader, M., Güler, S., Jaleel, H., Shamma, J.S., 2021. Aerial swarms: Recent applications and challenges. Current Robotics Reports 2, 309–320. doi:10.1007/s43154-021-00063-4.

Abro, G.E.M., Abdallah, A.M., Zahid, F., Ahmed, S., 2025. A comprehensive review of next-gen UAV swarm robotics: Optimisation techniques and control strategies for dynamic environments. Intelligent Automation & Soft Computing 40, 99–123. URL: http://www.techscience.com/iasc/v40n1/59313, doi:10.32604/iasc.2025.060364.

Acharya, S., Bharadwaj, A., Simmhan, Y., Gopalan, A., Parag, P., Tyagi, H., 2020. Cornet: A co-simulation middleware for robot networks, in: 2020 International Conference on COMmunication Systems NETworkS (COMSNETS), pp. 245–251. doi:10.1109/COMSNETS48256.2020.9027459.

Ahmed, E., Rehmani, M.H., 2017. Mobile edge computing: Opportunities, solutions, and challenges, in: Future Generation Computer Systems, Elsevier. pp. 59–63.

Aldossary, M., et al., 2024. Optimizing Task Offloading for Collaborative Unmanned Aerial Vehicles (UAVs) in Fog–Cloud Computing Environments. IEEE Access 12, 74698–74710. doi:10.1109/ACCESS.2024.3405566.

Alfeo, A.L., Cimino, M.G.C.A., Francesco, N.D., Lazzeri, A., Lega, M., Vaglini, G., 2019. Swarm coordination of mini-UAVs for target search using imperfect sensors. CoRR abs/1901.02885. URL: http://arxiv.org/abs/1901.02885, arXiv:1901.02885.

Alharbi, H.A., Yosuf, B.A., Aldossary, M., Almutairi, J., Elmirghani, J.M.H., 2022. Energy efficient UAV-based service offloading over cloud-fog architectures. IEEE Access 10, 89598–89613. doi:10.1109/ACCESS.2022.3201112.

Alqudsi, Y., Makaraci, M., 2025. UAV swarms: research, challenges, and future directions. Journal of Engineering and Applied Science 72. doi:10.1186/s44147-025-00582-3.

Anagnostopoulos, C., Koulamas, C., Lalos, A., Stylios, C., 2022. Open-source integrated simulation framework for cooperative autonomous vehicles, in: 2022 11th Mediterranean

Conference on Embedded Computing (MECO), pp. 1–4. doi:10.1109/MECO55406.2022.9797115.

Ardupilot, 2021a. Controller diagrams. URL: https://docs.px4.io/master/en/flight_stack/controller_diagrams.html. accessed: 2021-06-15.

Ardupilot, 2021b. Multicopter PID tuning guide (manual/advanced). URL: https://docs.px4.io/master/en/config_mc/pid_tuning_guide_multicopter.html#rate-controller. accessed: 2021-06-15.

Asaamoning, G., Mendes, P., Rosário, D., Cerqueira, E., 2021. Drone swarms as networked control systems by integration of networking and computing. Sensors 21. URL: https://www.mdpi.com/1424-8220/21/8/2642, doi:10.3390/s21082642.

Ball, J.T., 2013. Advantages of ACT-R over prolog for natural language analysis, in: Proceedings of the 22nd Annual Conference on Behavior Representation in Modeling and Simulation, Citeseer.

Berger, G.S., Teixeira, M., Cantieri, A., Lima, J., Pereira, A.I., Valente, A., de Castro, G.G., Pinto, M.F., 2023. Cooperative heterogeneous robots for autonomous insects trap monitoring system in a precision agriculture scenario. Agriculture 13, 239.

Bonomi, F., Milito, R., Zhu, J., Addepalli, S., 2012. Fog computing and its role in the internet of things, in: Proceedings of the first edition of the MCC workshop on Mobile cloud computing, ACM. pp. 13–16. doi:10.1145/2342509.2342513.

Butz, M.V., 2016. Toward a unified sub-symbolic computational theory of cognition. Frontiers in psychology 7, 925.

Calvo-Fullana, M., Mox, D., Pyattaev, A., Fink, J., Kumar, V., Ribeiro, A., 2021. ROS-NetSim: A framework for the integration of robotic and network simulators. CoRR abs/2101.10113. URL: https://arxiv.org/abs/2101.10113, arXiv:2101.10113.

Checker, L., Xie, H., Khaksar, S., Murray, I., 2025. Systematic review of multi-objective UAV swarm mission planning systems from regulatory perspective. Drones 9. URL: https://www.mdpi.com/2504-446X/9/7/509, doi:10.3390/drones9070509.

Chella, A., Cossentino, M., Pirrone, R., Ruisi, A., 2009. Robotic cognitive architectures: A perspective, in: 2009 Annual Conference of the North American Fuzzy Information Processing Society, IEEE. pp. 1–6.

Chen, M., Wan, J., González, S., Liao, X., Leung, V.C., 2018. IoT-based cloud framework to control ebola virus outbreak. Journal of Systems Architecture 86, 93–103.

Chen, Z., Xiao, N., Han, D., 2020. A multilevel mobile fog computing offloading model based on UAV-assisted and heterogeneous network. Wireless Communications and Mobile Computing 2020, 8833722. URL: https://doi.org/10.1155/2020/8833722, doi:10.1155/2020/8833722.

Chiang, M., Zhang, T., 2016. Fog and IoT: An overview of research opportunities. IEEE Internet of Things Journal 3, 854–864.

Coates, A., Ng, A., Lee, H., 2011. An analysis of single-layer networks in unsupervised feature learning, in: Gordon, G., Dunson, D., Dudík, M. (Eds.), Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, PMLR, Fort Lauderdale, FL, USA. pp. 215–223. URL: https://proceedings.mlr.press/v15/coates11a.html.

Cox, M.T., 2005. Metacognition in computation: A selected research review. Artificial Intelligence 169, 104–141.

Dai, X., Liu, K., Xia, Y., 2022. Event-triggered consensus-based formation control for multi-agent systems. Automatica 135, 109987.

Das, T., 2016. Intelligent techniques in decision making: A survey. Indian Journal of Science and Technology 9.

Dastjerdi, A.V., Buyya, R., 2016. Fog computing: Helping the internet of things realize its potential. IEEE Computer 49, 112–116.

Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numerische Mathematik 1, 269–271. doi:10.1007/BF01386390.

Dominey, P.F., Warneken, F., 2011. The basis of shared intentions in human and robot cognition. New Ideas in Psychology 29, 260–274.

Du, Y., de Silva, C.W., Liu, D., 2014. A multi-agent hybrid cognitive architecture with self-awareness for homecare robot, in: Computer Science & Education (ICCSE), 2014 9th International Conference on, IEEE. pp. 223–228.

Emel'yanov, S., Makarov, D., Panov, A.I., Yakovlev, K., 2016. Multilayer cognitive architecture for UAV control. Cognitive Systems Research 39, 58–72.

Erez, T., Tassa, Y., Todorov, E., 2015. Simulation tools for model-based robotics: Comparison of bullet, havok, mujoco, ode and physx, in: 2015 IEEE International Conference on Robotics and Automation (ICRA), pp. 4397–4404. doi:10.1109/ICRA.2015.7139807.

Faria Pinto, M., Melo, A., Marcato, A., Urdiales, C., 2017. Case-based reasoning approach applied to surveillance system using an autonomous unmanned aerial vehicle, in: 2017 IEEE 26th International Symposium on Industrial Electronics (ISIE), pp. 1324–1329. doi:10.1109/ISIE.2017.8001437.

Friess, C., Niculescu, V., Polonelli, T., Magno, M., Benini, L., 2023. Fully Onboard SLAM for Distributed Mapping with a Swarm of Nano-Drones. arXiv preprint arXiv:2309.03678 .

Gazebo, 2021. Gazebo robot simulator. http://gazebosim.org/. Accessed in 26/02/2021.

González-Santamarta, M., Rodríguez-Lera, F.J., Matellán-Olivera, V., 2025. Cognitive architectures in autonomous robotics: A systematic review of behavior generation approaches and evaluation strategies. IEEE Access 13, 191619–191644. doi:10.1109/ACCESS.2025.3630344.

Green, D.R., Hagon, J.J., Gomez, C., Gregory, B.J., 2019. Chapter 21 - using low-cost UAVs for environmental monitoring, mapping, and modelling: Examples from the coastal zone, in: Krishnamurthy, R., Jonathan, M., Srinivasalu, S., Glaeser, B. (Eds.), Coastal Management. Academic Press, pp. 465–501. URL: https://www.sciencedirect.com/science/article/pii/B9780128104736000224, doi:https://doi.org/10.1016/B978-0-12-810473-6.00022-4.

Gupta, A., Gupta, S.K., 2022. UAV aided fog network (uafn): A proposal framework for better QoS, in: 2022 2nd International Conference on Computing and Information Technology (ICCIT), pp. 265–270. doi:10.1109/ICCIT52419.2022.9711624.

Gupta, L., Jain, R., Vaszkun, G., 2016. Survey of important issues in UAV communication networks. IEEE Communications Surveys Tutorials 18, 1123–1152. doi:10.1109/COMST.2016.2530759.

Haazebroek, P., Van Dantzig, S., Hommel, B., 2011. A computational model of perception and action for cognitive robotics. Cognitive processing 12, 355–365.

Hart, P.E., Nilsson, N.J., Raphael, B., 1968. A formal basis for the heuristic determination of minimum cost paths. IEEE Transactions on Systems Science and Cybernetics 4, 100–107. doi:10.1109/TSSC.1968.300136.

Hawes, N., Burbridge, C., Jovan, F., Kunze, L., Lacerda, B., Mudrová, L., Young, J., Wyatt, J.L., 2017. The strands project: Long-term autonomy in everyday environments. IEEE Robotics & Automation Magazine 24, 146–156.

Himabindu, M., V, R., Gupta, M., Rana, A., Chandra, P.K., Abdulaali, H.S., 2023. Neuro-symbolic ai: Integrating symbolic reasoning with deep learning, in: 2023 10th IEEE Uttar Pradesh Section International Conference on Electrical, Electronics and Computer Engineering (UPCON), pp. 1587–1592. doi:10.1109/UPCON59197.2023.10434380.

Hu, Z., Liu, S., Zhou, D., Shen, C., 2025. Task Offloading and Data Compression Collaboration Optimization for UAV Swarm-Enabled Mobile Edge Computing. Drones doi:10.3390/drones9040288.

Insaurralde, C.C., 2014. Service-oriented agent architecture for unmanned air vehicles, in: Digital Avionics Systems Conference (DASC), 2014 IEEE/AIAA 33rd, IEEE. pp. 8B1–1.

Insaurralde, C.C., 2015. Architectural design for intelligent autonomy in unmanned aircraft, in: 2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC), pp. 5C3–1–5C3–12. doi:10.1109/DASC.2015.7311432.

Intel, 2024. Intel galileo board. URL: https://www.intel.com/content/www/us/en/products/details/boards-kits/galileo.html.

Javed, S., Hassan, A., Ahmad, R., Ahmed, W., Ahmed, R., Saadat, A., Guizani, M., 2024. State-of-the-art and future research challenges in UAV swarms. IEEE Internet of Things Journal 11, 19023–19045. doi:10.1109/JIOT.2024.3364230.

Jayasankar, U., Thirumal, V., Ponnurangam, D., 2021. A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications. Journal of King Saud University - Computer and Information Sciences 33, 119–140. URL: https://www.sciencedirect.com/science/article/pii/S1319157818301101, doi:https://doi.org/10.1016/j.jksuci.2018.05.006.

Kalami, S.M.H., 2021. path-planning: A collection of path planning algorithms. https://github.com/smkalami/path-planning. Accessed: 2023-03-24.

Kehoe, B., Patil, S., Abbeel, P., Goldberg, K., 2015. A survey of research on cloud robotics and automation. IEEE Transactions on Automation Science and Engineering 12, 398–409.

Kelley, T.D., 2003. Symbolic and sub-symbolic representations in computational models of human cognition: What can be learned from biology? Theory & Psychology 13, 847–860.

Kennedy, J., Eberhart, R., 1995. Particle swarm optimization, in: Proceedings of ICNN'95 - International Conference on Neural Networks, IEEE. pp. 1942–1948. doi:10.1109/ICNN.1995.488968.

Khamis, A., Hussein, A., Elmogy, A., 2021. Swarm robotics: A survey from a multi-tasking perspective. IEEE Access 9, 133158–133180. doi:10.1109/ACCESS.2021.3114263.

Kinsner, W., Kaynak, O., Zhu, H., Hou, M., Rayz, J., Gavrilova, M., Wang, G., Baciu, G., Peng, J., Chan, C.W., Huang, R., Du, B., 2022. Recent advances in cognitive informatics and cognitive computing towards autonomous systems (plenary panel report-ii of ieee icci*cc'22), in: 2022 IEEE 21st International Conference on Cognitive Informatics Cognitive Computing (ICCI*CC), pp. 268–280. doi:10.1109/ICCICC57084.2022.10101660.

Kotseruba, I., Tsotsos, J.K., 2020. 40 years of cognitive architectures: core cognitive abilities and practical applications. Artificial Intelligence 53, 17–94. doi:10.1007/s10462-018-9646-y.

Krichmar, J.L., 2018. Design principles for biologically inspired cognitive robotics. Biological Cybernetics 112, 97–111.

Kuffner, J., 2010. Cloud-enabled robots. IEEE Internet Computing 14, 44–46.

La Grassa, R., Re, C., Cremonese, G., Gallo, I., 2022. Hyperspectral data compression using fully convolutional autoencoder. Remote Sensing 14, 2472. doi:10.3390/rs14102472.

Laird, J.E., Kinkade, K.R., Mohan, S., Xu, J.Z., 2012. Cognitive robotics using the soar cognitive architecture, in: Workshops at the Twenty-Sixth AAAI Conference on Artificial Intelligence, Cognitive Robotics.

Langley, P., Laird, J.E., Rogers, S., 2009. Cognitive architectures and general intelligent systems. AI Magazine 30, 33.

Langley, P., Laird, J.E., Rogers, S.M., 2020. Cognitive architectures: Research issues and challenges. Cognitive Systems Research 60, 26–41. doi:10.1016/j.cogsys.2006.07.004.

LaValle, S.M., 1998. Rapidly-exploring random trees: A new tool for path planning, in: TR 98-11.

Leonetti, M., Iocchi, L., Stone, P., 2016. A synthesis of automated planning and reinforcement learning for efficient, robust decision-making. Artificial Intelligence 241, 103–130.

Liao, S.H., 2005. Expert system methodologies and applications—a decade review from 1995 to 2004. Expert systems with applications 28, 93–103.

Liu, C., Chen, W.H., Yang, G., 2020. Distributed model predictive control for multi-robot formation. IEEE Transactions on Control Systems Technology 28, 2456–2468.

Liu, T., Wang, J., Liu, Q., Alibhai, S., Lu, T., He, X., 2021. High-ratio lossy compression: Exploring the autoencoder to compress scientific data. IEEE Transactions on Big Data , 1–1doi:10.1109/TBDATA.2021.3066151.

Liu, W., Zhang, T., Huang, J., 2022. Multi-agent reinforcement learning-based task allocation for UAV swarms. IEEE Transactions on Intelligent Transportation Systems 23, 17245–17257. doi:10.1109/TITS.2022.3150637.

Luna, M.A., Ale Isaac, M.S., Ragab, A.R., Campoy, P., Flores Pena, P., Molina, M., 2022. Fast multi-UAV path planning for optimal area coverage in aerial sensing applications. Sensors 22. URL: https://www.mdpi.com/1424-8220/22/6/2297, doi:10.3390/s22062297.

Luthra, M., 2023. Aerial robot swarms: a review. IAES International Journal of Robotics and Automation (IJRA) 12, 137–145. URL: https://ijra.iaescore.com/index.php/IJRA/article/view/20532, doi:10.11591/ijra.v12i2.pp137-145.

Manfreda, S., Dvorak, P., Mullerova, J., Herban, S., Vuono, P., Arranz Justel, J.J., Perks, M., 2019. Assessing the accuracy of digital surface models derived from optical imagery acquired with unmanned aerial systems. Drones 3. URL: https://www.mdpi.com/2504-446X/3/1/15.

MAVLINK, 2022. Mavlink. https://mavlink.io/en/. Accessed in 04/12/2022.

Modha, D.S., Ananthanarayanan, R., Esser, S.K., Ndirango, A., Sherbondy, A.J., Singh, R., 2011. Cognitive computing. Communications of the ACM 54, 62–71.

Mohini, A., 2019. CDSSim - Multi UAV Communication and Control Simulation Framework. Master's thesis. University of Cincinnati. 2600 Clifton Ave, Cincinnati, OH 45221, United States.

NS-3, 2022. NS-3 network simulator. https://www.nsnam.org/. Accessed in 04/12/2022.

Ogata, T., Takahashi, K., Yamada, T., Murata, S., Sasaki, K., 2022. Machine learning for cognitive robotics .

OMNET++, 2022. OMNET++: Discrete event simulator. https://omnetpp.org/. Accessed in 04/12/2022.

OpenCV, 2025a. Camera Calibration and 3D Reconstruction. https://docs.opencv.org/4.x/d9/d0c/group__calib3d.html. Accessed: 2025-08-19.

OpenCV, 2025b. Feature Matching. https://docs.opencv.org/4.x/dc/dc3/tutorial_py_matcher.html. Accessed: 2025-08-19.

OpenCV, 2025c. ORB (Oriented FAST and Rotated BRIEF). https://docs.opencv.org/4.x/d1/d89/tutorial_py_orb.html. Accessed: 2025-08-19.

Parikh, J., Basu, A., 2021. Unmanned Aerial Vehicles: State-of-the-Art, Challenges and Future Scope. John Wiley Sons, Ltd. chapter 2. pp. 29–42. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119769170.ch2, doi:https://doi.org/10.1002/9781119769170.ch2, arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119769170.ch2.

Park, J., Park, H., Choi, Y.J., 2018. Data compression and prediction using machine learning for industrial IoT, in: 2018 International Conference on Information Networking (ICOIN), pp. 818–820. doi:10.1109/ICOIN.2018.8343232.

Pi, R., 2024. Raspberry pi. URL: https://www.raspberrypi.org.

Pinto, M.F., Honório, L.M., Marcato, A.L., Dantas, M.A., Melo, A.G., Capretz, M., Urdiales, C., 2020a. ARCog: An aerial robotics cognitive architecture. Robotica , 1–20.

Pinto, M.F., Honorio, L.M., Melo, A., Marcato, A.L., 2020b. A robotic cognitive architecture for slope and dam inspections. Sensors 20, 4579.

Pinto, M.F., Marcato, A.L., Melo, A.G., Honório, L.M., Urdiales, C., 2019. A framework for analyzing fog-cloud computing cooperation applied to information processing of UAVs. Wireless Communications and Mobile Computing 2019.

Pirmagomedov, R., Moltchanov, D., Samuylov, A., Orsino, A., Torsner, J., Andreev, S., Koucheryavy, Y., 2022. Characterizing throughput and convergence time in dynamic multi-connectivity 5g deployments. Computer Communications 187, 45–58. URL: https://www.sciencedirect.com/science/article/pii/S0140366422000275, doi:https://doi.org/10.1016/j.comcom.2022.01.015.

Premsankar, G., Di Francesco, M., Taleb, T., 2018. Edge computing for the internet of things: A case study. IEEE Internet of Things Journal 5, 1275–1284.

Pu, Z., Zhang, T., Ai, X., Qiu, T., Yi, J., 2023. A deep reinforcement learning approach combined with model-based paradigms for multiagent formation control with collision avoidance. IEEE Transactions on Systems, Man, and Cybernetics: Systems 53, 4189–4204. doi:10.1109/TSMC.2023.3241337.

Pugliese, L.D.P., Guerriero, F., Macrina, G., 2020. Using drones for parcels delivery process. Procedia Manufacturing 42, 488–497. URL: https://www.sciencedirect.com/science/article/pii/S2351978920305928, doi:https://doi.org/10.1016/j.promfg.2020.02.043. international Conference on Industry 4.0 and Smart Manufacturing (ISM 2019).

PX4, 2022. Px4 autopilot. https://px4.io/. Accessed in 04/12/2022.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y., 2009. ROS: an open-source robot operating system, in: ICRA workshop on open source software, Kobe, Japan. p. 5.

Qureshi, B., 2016. Cloud robotics: Future directions and open challenges. Preprints .

Ramos, G.S., 2024a. ARCog-NET - simulation 01 - wind energy turbines field surveyllance. URL: https://youtu.be/LXOZyww0Jhs. accessed: 2024-04-17.

Ramos, G.S., 2024b. ARCog-NET - simulation 02 - offshore environmental monitoring. URL: https://youtu.be/vJP7v9KbKOI. accessed: 2024-04-24.

Ramos, G.S., 2024c. ARCog-NET - simulation 03 - provide mobile phone coverage to a neighborhood. URL: https://youtu.be/TP5_Ld00ou0. accessed: 2024-04-16.

Ramos, G.S., 2024d. ARCog-NET - simulation 04 - industrial plant indoor search and rescue. URL: https://youtu.be/o6rse5FPzo8. accessed: 2024-04-24.

Ramos, G.S., 2025a. ARCog-NET. https://github.com/gabryelsr/ARCog-NET. Accessed: 2025-07-28.

Ramos, G.S., 2025b. ARCog-NET Experiment ROS Nodes and Topics. https://github.com/gabryelsr/ARCog-NET/blob/85521cd151c4bdf3bb956e9c9426a2dfb8581da4/experiment/data/ros_experiment.png. Accessed: 2025-08-19.

Ramos, G.S., 2025c. ARCog-NET Indoor Navigation and Mapping Experiment. https://www.youtube.com/watch?v=YHxJ-1ZHzsw. Accessed: 2025-08-19.

Ramos, G.S., Barreto Haddad, D., Barros, A.L., de Melo Honorio, L., Faria Pinto, M., 2022a. EKF-based vision-assisted target tracking and approaching for autonomous UAV in offshore mooring tasks. IEEE Journal on Miniaturization for Air and Space Systems 3, 53–66. doi:10.1109/JMASS.2022.3195660.

Ramos, G.S., De Lima, A.A., Almeida, L.F., Lima, J., Pinto, M.F., 2023. Simulation and evaluation of deep learning autoencoders for image compression in multi-UAV network systems, in: 2023 Latin American Robotics Symposium (LARS), 2023 Brazilian Symposium on Robotics (SBR), and 2023 Workshop on Robotics in Education (WRE), pp. 41–46. doi:10.1109/LARS/SBR/WRE59448.2023.10332986.

Ramos, G.S., Henriques, F.D.R., Haddad, D.B., Andrade, F.A.A., Pinto, M.F., 2024. ARCog-NET: An Aerial Robot Cognitive Network Architecture for Swarm Applications Development. IEEE Access 12, 129040–129063. doi:10.1109/ACCESS.2024.3456914.

Ramos, G.S., Pinto, M.F., 2024. Aerial robot cognitive network architecture for swarm applications development (ARCog-NET). URL: https://revistas.inpi.gov.br/pdf/Programa_de_computador2781.pdf. process No. BR 51 2024 001222-2.

Ramos, G.S., Pinto, M.F., Andrade, F., Haddad, D.B., 2025a. Implementation of monocular visual slam with ARCog-NET for aerial robot swarm indoor mapping. Scientific Reports doi:https://doi.org/10.21203/rs.3.rs-7573197/v1. 10 October 2025, PREPRINT (Version 1) available at Research Square.

Ramos, G.S., Pinto, M.F., Coelho, F.O., Honório, L.M., Haddad, D.B., 2022b. Hybrid methodology based on computational vision and sensor fusion for assisting autonomous UAV on offshore messenger cable transfer operation. Robotica 40, 2786–2814.

Ramos, G.S., Pinto, M.F., Haddad, D.B., 2025b. Advancing UAV swarm autonomy with ARCog-NET for task allocation, path planning, and formation control. Robotica , 1–45doi:10.1017/S0263574725101914.

Reggia, J.A., 2013. The rise of autonomous artificial cognitive systems: Issues and challenges. IEEE Computational Intelligence Magazine 8, 16–31.

Rodin, C.D., de Lima, L.N., de Alcantara Andrade, F.A., Haddad, D.B., Johansen, T.A., Storvold, R., 2018. Object classification in thermal images using convolutional neural networks for search and rescue missions with unmanned aerial systems, in: 2018 International Joint Conference on Neural Networks (IJCNN), pp. 1–8. doi:10.1109/IJCNN.2018.8489465.

ROS, 2022. MAVROS. http://wiki.ros.org/mavros. Accessed in 04/12/2022.

Roth-Berghofer, T., Recio Garcia, J.A., Sauer, C., Bach, K., Althoff, K.D., Diaz-Agudo, B., Gonzales Calero, P.A., 2012. Building case-based reasoning applications with myCBR and COLIBRI studio. ICCBR 2012 13.

Rumelhart, D.E., 2017. Schemata: The building blocks of cognition, in: Theoretical issues in reading comprehension. Routledge, pp. 33–58.

Ryze Robotics, 2025. Ryze Tello Standard Development Kit. https://www.ryzerobotics.com/tello/downloads. Accessed: 2025-08-19.

Saha, O., Dasgupta, P., Maity, S.P., 2018. A comprehensive survey on cloud robotics. Archives of Computational Methods in Engineering , 1–23.

Sampedro, C., Bavle, H., Sanchez Lopez, J.L., Suarez Fernandez, R., Rodriguez Ramos, A., Molina, M., Campoy Cervera, P., 2016. A flexible and dynamic mission planning

architecture for UAV swarm coordination, in: Proceedings of 2016 International Conference on Unmanned Aircraft Systems (ICUAS), ETSI_Informatica.

Sanchez-Lopez, J.L., Molina, M., Bavle, H., Sampedro, C., Fernández, R.A.S., Campoy, P., 2017. A multi-layered component-based approach for the development of aerial robotic systems: the aerostack framework. Journal of Intelligent & Robotic Systems 88, 683–709.

Satyanarayanan, M., 2017. The emergence of edge computing. IEEE Computer 50, 30–39.

Scherer, J., Rinner, B., 2016. Persistent multi-UAV surveillance with energy and communication constraints, in: 2016 IEEE International Conference on Automation Science and Engineering (CASE), pp. 1225–1230. doi:10.1109/COASE.2016.7743546.

Seleckỳ, M., Rollo, M., Losiewicz, P., Reade, J., Maida, N., 2015. Framework for incremental development of complex unmanned aircraft systems, in: Integrated Communication, Navigation, and Surveillance Conference (ICNS), 2015, IEEE. pp. J3–1.

Shi, W., Cao, J., Zhang, Q., Li, Y., Xu, L., 2016. Edge computing: Vision and challenges. IEEE Internet of Things Journal 3, 637–646.

Shin, H.S., Bang, H.C., 2017. Market-based task assignment for cooperative timing missions in dynamic environments. Journal of Intelligent & Robotic Systems 87, 97–123. doi:10.1007/s10846-017-0493-x.

Stojmenovic, I., Wen, S., 2014. Fog computing: A cloud to the ground support for smart things and machine-to-machine networks. Australasian Telecommunication Networks and Applications Conference (ATNAC), 2014 , 117–122.

Sutton, R.S., Barto, A.G., 2018. Reinforcement Learning: An Introduction. MIT Press.

Tahir, A., Boling, J., Haghbayan, M.H., Toivonen, H.T., Plosila, J., 2019a. Swarms of unmanned aerial vehicles — a survey. Journal of Industrial Information Integration 16, 100106. URL: https://www.sciencedirect.com/science/article/pii/S2452414X18300086, doi:https://doi.org/10.1016/j.jii.2019.100106.

Tahir, A., Boling, J., Haghbayan, M.H., Toivonen, H.T., Plosila, J., 2019b. Swarms of Unmanned Aerial Vehicles — A Survey. Journal of Industrial Information Integration 16, 100106.

Tahir, N., Parasuraman, R., 2025. Edge Computing and Its Application in Robotics: A Survey. Journal of Sensor and Actuator Networks 14, 65–92. doi:10.3390/jsan14040065.

Tweedale, J.W., 2014. A review of cognitive decision-making within future mission systems. Procedia Computer Science 35, 1043–1052.

Vaquero, L.M., Rodero-Merino, L., 2014. Finding your way in the fog: Towards a comprehensive definition of fog computing. ACM SIGCOMM Computer Communication Review 44, 27–32.

Vernon, D., 2016. Artificial Cognitive Systems: A Primer. MIT Press.

Wan, J., Tang, S., Shu, Z., Li, D., Wang, S., Imran, M., Vasilakos, A.V., 2016. Cloud robotics: Current status and open issues. IEEE Access 4, 2797–2807.

Wang, Y., Widrow, B., Pedrycz, W., Kacprzyk, J., Kwong, S., Plataniotis, K., Guan, L., Spachos, P., 2022. Recent breakthroughs in cognitive informatics and cognitive computing towards autonomous ai (plenary panel report-i of ieee icci*cc'22), in: 2022 IEEE 21st International Conference on Cognitive Informatics Cognitive Computing (ICCI*CC), pp. 258–267. doi:https://doi.org/10.1109/ICCICC57084.2022.10101506.

Wu, X., Wu, C., Deng, P., 2023. Prediction of packet loss rate in non-stationary networks based on time-varying autoregressive sequences. Electronics 12. URL: https://www.mdpi.com/2079-9292/12/5/1103, doi:10.3390/electronics12051103.

Xu, H., Liu, P., Chen, X., Shen, S., 2022. D2SLAM: Decentralized and Distributed Collaborative Visual-inertial SLAM System for Aerial Swarm. arXiv preprint arXiv:2211.01538 .

Yang, Y., Guha, A., Fermüller, C., Aloimonos, Y., 2014. A cognitive system for understanding human manipulation actions. Advances in Cognitive Systems 3, 67–86.

Yang, Z., Zhang, H., Xu, L., 2020. Multi-UAV cooperative task allocation based on improved genetic algorithm. IEEE Access 8, 65432–65442. doi:10.1109/ACCESS.2020.2985071.

Yi, S., Li, C., Li, Q., 2015. A survey of fog computing: Concepts, applications and issues, in: Proceedings of the 2015 Workshop on Mobile Big Data, pp. 37–42.

Zebang, S., Sei-ichiro, K., 2019. Densely connected autoencoders for image compression, in: Proceedings of the 2nd International Conference on Image and Graphics Processing,

Association for Computing Machinery, New York, NY, USA. p. 78–83. URL: https://doi.org/10.1145/3313950.3313965, doi:10.1145/3313950.3313965.

Zhai, J., Zhang, S., Chen, J., He, Q., 2018. Autoencoder and its various variants, in: 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), pp. 415–419. doi:10.1109/SMC.2018.00080.

Zhao, X., Liu, Y., Chen, H., 2021. Autonomous task allocation for UAV swarms using heuristic clustering and auction mechanisms. IEEE Transactions on Vehicular Technology 70, 7404–7417. doi:10.1109/TVT.2021.3088656.

Zhong, S., Qi, Y., Chen, Z., Wu, J., Chen, H., Liu, M., 2022. DCL-SLAM: A Distributed Collaborative LiDAR SLAM Framework for a Robotic Swarm. arXiv preprint arXiv:2210.11978 .

Zhou, Y., Rao, B., Wang, W., 2020. UAV Swarm Intelligence: Recent Advances and Future Trends. IEEE Access 8, 183856–183878.

Zuo, K., Xin, L., 2025. Boosting swarm robotics adaptability through transactive memory integration, in: 2025 6th International Conference on Artificial Intelligence and Electromechanical Automation (AIEA), pp. 421–424. doi:10.1109/AIEA66061.2025.11159664.

Attachments

Attachment I - Edge Data Structures

Variable	Data description	Type	Workflow	Workflow description
ID_agent	agent ID code	String	A1	Agent data
status	current agent status (navigating, hovering, returning to home, inspecting, taking off, landing, landed)	Integer	A1	Agent data
pos_x	current x position	Float	A1	Agent data
pos_y	current y position	Float	A1	Agent data
pos_z	current z position	Float	A1	Agent data
heading	current heading	Float	A1	Agent data
vel	current velocity	Float	A1	Agent data
ang_vel	current angular velocity	Float	A1	Agent data
bat	current battery level	Float	A1	Agent data
cur_traj	a dataframe of all points navigated by the agent	Dataframe	A2	Agent processed data
plan_traj	a dataframe of all points planned by the coordinator	Dataframe	C2	Coordinator processed data
send_images	command taken from active coordinator to send or not send sensor data	Integer	C3	Coordinator request to agent
img	an numpy array of a picture taken (it may be empty)	Array Numpy	A1	Agent data
found_obj	an array of found objects, its contours, areas and centroids	List	A2	Agent processed data
dt_sent	date and time the current package is being sent	Datetime	A1	Agent data
dt_last_sent	date and time the last package was sent	Datetime	A1	Agent data
dt_last_rec_contact	list of last contacts received from coordinators	List of Datetime	C1	Coordinator data
ID_coordinator	list of coordinators ID codes	List of Strings	C1	Coordinator data
command_from_coordinator	last command received from coordinator (navigate, hover, return to home, takeoff, land, inspect, switch coordinator)	Integer	C5	Coordinator command to agent
request_to_coordinator	request sent to coordinator (stop, continue, return to home, land, inspect, switch coordinator, send images, switch trajectory)	Integer	A3	Agent request to coordinator
ID_server	server ID tag	String	S1	Server data
command_from_server	last command received from server (navigate, hover, return to home, takeoff, land, inspect, switch coordinator)	Integer	S6	Server command to agent
request_from_server	request received from server	Integer	S3	Server request to agent
request_to_server	request sent to server (stop, continue, return to home, land, inspect, switch coordinator, send images, switch trajectory)	Integer	A4	Agent request to server
processing_time	time taken in all process between two consecutive packages	Float	A2	Agent processed data
com_latency	communication latency accounted from last sent and received packages	Float	A2	Agent processed data
package_volume	information of how many bits the sent package have	Integer	A2	Agent processed data

Attachment II - Fog Data Structures

Variable	Data description	Type	Workflow	Workflow description
ID_coordinator	coordinator ID code	String	C1	Coordinator data
coordinator_status	current coordinator status (navigating, hovering, returning to home, inspecting, taking off, landing, landed, monitoring agents)	Integer	C1	Coordinator data
pos_x	current x position	Float	C1	Coordinator data
pos_y	current y position	Float	C1	Coordinator data
pos_z	current z position	Float	C1	Coordinator data
heading	current heading	Float	C1	Coordinator data
vel	current velocity	Float	C1	Coordinator data
ang_vel	current angular velocity	Float	C1	Coordinator data
bat	current battery level	Float	C1	Coordinator data
ID_coordinators	a list of all coordinators IDs the coordinator is communicating with	List of Strings	C1	Coordinator data
coordinators_pos_x	list of coordinators current x position	Float	C1	Coordinator data
coordinators_pos_y	list of coordinators current y position	Float	C1	Coordinator data
coordinators_pos_z	list of coordinators current z position	Float	C1	Coordinator data
coordinators_heading	list of coordinators current heading	Float	C1	Coordinator data
coordinators_vel	list of coordinators current velocity	Float	C1	Coordinator data
coordinators_ang_vel	list of coordinators current angular velocity	Float	C1	Coordinator data
coordinators_bat	list of coordinators current battery level	Float	C1	Coordinator data
dt_sent	date and time the current package is being sent	Datetime	C1	Coordinator data
dt_last_sent	date and time the last package was sent	Datetime	C1	Coordinator data
dt_last_rec_contacts_coordinators	last contact received from the coordinates of the list	List of Datetime	C1	Coordinator data
com_latency_coordinators	communication latency accounted from last sent and received packages from the coordinators in the list	List of Float	C2	Coordinator processed data
network_coordinators_status	list of current coordinators status (navigating, hovering, returning to home, inspecting, taking off, landing, landed, monitoring agents)	List of Integers	C1	Coordinator data
ID_agents	a list of all agent IDs the coordinator is communicating with	List of Strings	A1	Agent data
agents_status	list of agents status	List of Integers	A1	Agent data
agents_pos_x	list of agents current x positions	List of Float	A1	Agent data
agents_pos_y	list of agents current y positions	List of Float	A1	Agent data
agents_pos_z	list of agents current z positions	List of Float	A1	Agent data
agents_heading	list of agents current heading	List of Float	A1	Agent data
agents_vel	list of agents current velocity	List of Float	A1	Agent data
agents_ang_vel	list of agents current angular velocity	List of Float	A1	Agent data
agents_bat	list of agents current battery	List of Float	A1	Agent data
agents_cur_trjs	dataframes of dataframes with all points navigated by the active agents	Dataframe	A2	Agent processed data
agents_plan_trjs	dataframes of dataframes with of all points planned by the coordinator for active agents	Dataframe	A2	Agent processed data
receive_images	list of which active agent must send sensor data or not	List of Integers	C3	Coordinator request to agent
imgs	dataframe of dataframes of numpy arrays of pictures taken by active agents (it may be empty depending on "receive_images" command)	Dataframe	A2	Agent processed data
found_obj	dataframe of arrays of found objects, its contours, areas and centroids by the active agents	Dataframe	A2	Agent processed data
dt_last_rec_contacts_agents	last contact received from the agents of the list	List of Datetime	A1	Agent data
com_latency_agents	communication latency accounted from last sent and received packages from the agents in the list	List of Float	A2	Agent processed data
last_commands_to_agents	list of last commands sent for active agents (acknowledge, navigate, hover, return to home, takeoff, land, inspect)	List of Integers	C5	Coordinator command to agent
last_command_to_agents_from_server	list of last commands sent for active agents from server (acknowledge, navigate, hover, return to home, takeoff, land, inspect, stop sending images, switch coordinator)	List of Integers	S6	Server command to agent
request_to_agents	request sent to agent (acknowledge, stop, continue, return to home, land, inspect, switch coordinator, send images, switch trajectory)	List of Integers	C3	Coordinator request to agent
request_to_agents_from_server	requests received from server to agents	List of Integers	S3	Server request to agent
requests_from_agents	requests received from agents	List of Integers	A3	Agent request to coordinator
requests_from_agents_to_server	requests received from agents to server	List of Integers	A4	Agent request to server
agent_processing_time	list of time taken in all process between two consecutive sent packages from agents	List of Float	A2	Agent processed data
agent_package_volume	list of how many bits the sent package have	List of Integer	A2	Agent processed data
ID_server	server ID tag	String	S1	Server data
server_status	server status	Integer	S1	Server data
server_position	server geographical coordinates	Dataframe	S1	Server data
command_from_server	list of commands sent from server to coordinator	List of Integers	S5	Server command to coordinator
request_to_server	request sent to server (acknowledge, stop, continue, return to home, land, inspect, switch coordinator, send images, switch trajectory, process data)	List of Integers	C4	Coordinator request to server
request_from_server	requests received from server	List of Integers	S4	Server request to coordinator
dt_last_rec_contact_server	last contact received from the server	Datetime	S1	Server data
com_latency_server	communication latency accounted from last sent and received packages from the server	Float	S2	Server processed data
server_processing_time	time taken in all process between two consecutive packages in server	List of Float	S2	Server processed data
processing_time	time taken in all process between two consecutive packages	Float	C2	Coordinator processed data
package_volume	information of how much bits the sent package have	Integer	C2	Coordinator processed data

Attachment III - Cloud Data Structures

Variable	Data description	Type	Workflow	Workflow description
ID_server	server ID tag	String	S1	Server data
server_status	server status	Integer	S1	Server data
server_position	server geographical coordinates	Dataframe	S1	Server data
dt_sent	date and time the current package is being sent	Datetime	S1	Server data
dt_last_sent	date and time the last package was sent	Datetime	S1	Server data
clusters	server data for which coordinator will be responsible for which agent	Dataframe	S2	Server processed data
processing_time	time taken in all process between two consecutive packages	Float	S2	Server processed data
ID_coordinators	a list of all coordinators IDs the server is communicating with	List of Strings	C1	Coordinator data
ID_agents	a list of all agent IDs the server is communicating with through coordinators	List of Strings	A1	Agent data
dt_last_rec_contacts_agents	last contact received from the agents of the list	List of Datetime	A1	Agent data
dt_last_rec_contacts_coordinat	last contact received from the coordinators of the list	List of Datetime	C1	Coordinator data
com_latency_agents	communication latency accounted from last sent and received packages from the agents in the list	List of Float	S2	Server processed data
com_latency_coordinators	communication latency accounted from last sent and received packages from the coordinators in the list	List of Float	S2	Server processed data
last_commands_to_coordinator	list of last commands sent for active agents from coordinators (acknowledge, navigate, hover, return to home, takeoff, land, inspect)	List of Integers	S5	Server command to coordinator
last_command_to_agents	list of last commands sent for active agents from server (acknowledge, navigate, hover, return to home, takeoff, land, inspect, stop sending images, switch coordinator)	List of Integers	S6	Server command to agent
request_to_coordinators	request sent to coordinator (acknowledge, stop, continue, return to home, land, inspect, switch coordinator, send images, switch trajectory)	List of Integers	S4	Server request to coordinator
request_to_agents	request sent to agent (acknowledge, stop, continue, return to home, land, inspect, switch coordinator, send images, switch trajectory, process data)	List of Integers	S3	Server request to agent
request_from_coordinators	requests received from coordinators	List of Integers	C4	Coordinator request to server
request_from_agents	requests received from agents	List of Integers	A4	Agent request to server
receive_images	list of which active agent must send sensor data or not	List of Integers	S4	Server request to coordinator
network_coordinators_status	list of current coordinators status (navigating, hovering, returning to home, inspecting, taking off, landing, landed, monitoring agents)	List of Integers	C1	Coordinator data
agents_pos_x	list of agents current x positions	List of Float	A1	Agent data
agents_pos_y	list of agents current y positions	List of Float	A1	Agent data
agents_pos_z	list of agents current z positions	List of Float	A1	Agent data
agents_heading	list of agents current heading	List of Float	A1	Agent data
agents_vel	list of agents current velocity	List of Float	A1	Agent data
agents_ang_vel	list of agents current angular velocity	List of Float	A1	Agent data
agents_bat	list of agents current battery	List of Float	A1	Agent data
coordinators_pos_x	list of coordinators current x position	Float	C1	Coordinator data
coordinators_pos_y	list of coordinators current y position	Float	C1	Coordinator data
coordinators_pos_z	list of coordinators current z position	Float	C1	Coordinator data
coordinators_heading	list of coordinators current heading	Float	C1	Coordinator data
coordinators_vel	list of coordinators current velocity	Float	C1	Coordinator data
coordinators_ang_vel	list of coordinators current angular velocity	Float	C1	Coordinator data
coordinators_bat	list of coordinators current battery level	Float	C1	Coordinator data
agents_cur_traj	dataframes of dataframes with all points navigated by the active agents	Dataframe	A1	Agent data
agents_plan_traj	dataframes of dataframes with of all points planned by the coordinator for active agents	Dataframe	C2	Coordinator processed data
imgs	dataframe of dataframes of numpy arrays of pictures taken by active agents (it may be empty depending on "receive_images" command)	Dataframe	A2	Agent processed data
found_obj	dataframe of arrays of found objects, its contours, areas and centroids by the active agents	Dataframe	A2	Agent processed data
package_volume	information of how much bits the sent package have	Integer	S2	Server processed data

Attachment IV - Active ROS Nodes and Messages

