INSTITUTO TECNOLÓGICO DE AERONÁUTICA

Marcus Gabriel de Almeida Nunes

PLUGIN FOR ANALYSIS AND MONITORING OF SIMULATION IN THE ARENA CONCEPT.IO

Final Paper
2025

Course of Aeronautical Engineering

CDU 681.3.064

Marcus Gabriel de Almeida Nunes

PLUGIN FOR ANALYSIS AND MONITORING OF SIMULATION IN THE ARENA CONCEPT.IO

Advisor

Maj. Av. Lucas Oliveira Barbacovi (ITA)

Co-advisor

Prof. Dr. Christopher Shneider Cerqueira (ITA)

AERONAUTICAL ENGINEERING

São José dos Campos
Instituto Tecnológico de Aeronáutica

2025

Cataloging-in Publication Data
Documentation and Information Division

de Almeida Nunes, Marcus Gabriel
Plugin for Analysis and Monitoring of Simulation in the Arena Concept.IO / Marcus Gabriel de Almeida Nunes.
São José dos Campos, 2025.
63p.

Final paper (Undergraduation study) – Course of Aeronautical Engineering– Instituto Tecnológico de Aeronáutica, 2025. Advisor: Maj. Av. Lucas Oliveira Barbacovi. Co-advisor: Prof. Dr. Christopher Shneider Cerqueira.

1. Veículos pilotados remotamente. 2. Engenharia de sistemas. 3. Sistemas complexos. 4. Simulação computadorizada. 5. Monitoramento. 6. Aprendizagem (inteligência artificial). 7. Computação. 8. Engenharia aeronáutica. I. Instituto Tecnológico de Aeronáutica. II. Plugin for Analysis and Monitoring of Simulation in the Arena Concept.IO.

BIBLIOGRAPHIC REFERENCE

DE ALMEIDA NUNES, Marcus Gabriel. **Plugin for Analysis and Monitoring of Simulation in the Arena Concept.IO**. 2025. 63p. Final paper (Undergraduation study) – Instituto Tecnológico de Aeronáutica, São José dos Campos.

CESSION OF RIGHTS

AUTHOR'S NAME: Marcus Gabriel de Almeida Nunes
PUBLICATION TITLE: Plugin for Analysis and Monitoring of Simulation in the Arena Concept.IO.
PUBLICATION KIND/YEAR: Final paper (Undergraduation study) / 2025

Marcus Gabriel de Almeida Nunes
Rua H8B, Ap. 209
12228-461 – São José dos Campos–SP

PLUGIN FOR ANALYSIS AND MONITORING OF SIMULATION IN THE ARENA CONCEPT.IO

This publication was accepted like Final Work of Undergraduation Study

---

Marcus Gabriel de Almeida Nunes

Author

Lucas Oliveira Barbacovi (ITA)

Advisor

Christopher Shneider Cerqueira (ITA)

Co-advisor

São José dos Campos: November 07, 2025.

Acknowledgments

First and foremost, I thank God for granting me strength in difficult moments, wisdom to keep moving forward and the opportunities that made the fulfillment of this dream possible.

To my family, for their support and encouragement, especially during the most challenging phases of this journey. Without you, none of this would have been possible.

To my girlfriend, Giovana, for all the love, understanding, and companionship throughout this final stage of my undergraduate studies and for celebrating with me each and every achievement along the way.

To my friends and classmates, for making the years of college lighter and more memorable, including during the famous "viradões" (all-nighters).

To my advisor and co-advisor, whose dedication, patience, and valuable guidance were fundamental to the development of this work and to my academic and professional growth.

To the professors of the Aeronautical Engineering program, for sharing not only technical knowledge, but also enthusiasm and inspiration, both of which were fundamental to my education.

"Wer einst fliegen lernen will, der muss erst stehn und gehn und laufen und klettern und tanzen lernen: – man erfliegt das Fliegen nicht!"

— Friedrich Wilhelm Nietzsche

Abstract

This work aims to develop a plugin for the Arena Concept.IO simulation platform, designed to integrate monitoring, metrics analysis, and machine learning within a Systems Engineering environment. The system was developed to collect and process data transmitted during simulations, recording them in structured files through a logging module, which allows for the assessment of mission performance and the identification of operational patterns based on predefined metrics. The collected information served as the basis for training a supervised model using the k-Nearest Neighbors algorithm, capable of classifying missions into success, partial failure or total failure categories. The results indicate that the model can recognize recurring behaviors and infer mission effectiveness, validating the use of machine learning as a support tool for the automated analysis of simulations. The plugin also incorporates an interactive dashboard that enables the visualization of simulated entities and their evolution over time. In light of these findings, the feasibility of integrating simulation and machine learning is confirmed, highlighting the potential of the proposed approach to support requirements verification and validation as well as the performance analysis of complex systems. The plugin can also be employed for failure studies and the optimization of operational strategies, serving as a foundation for future expansions in more complex and multidomain scenarios.

Resumo

Este trabalho tem como objetivo desenvolver um *plugin* para a plataforma de simulação Arena Concept.IO, voltado à integração de monitoramento, análise de métricas e aprendizado de máquina em um ambiente de Engenharia de Sistemas. O sistema foi projetado para coletar e processar dados transmitidos durante as simulações, registrando-os em arquivos estruturados por meio de um módulo de registrador, o que permite avaliar o desempenho das missões e identificar padrões operacionais com base em métricas previamente definidas. As informações coletadas serviram de base para o treinamento de um modelo supervisionado baseado no algoritmo dos k-vizinhos mais próximos, capaz de classificar as missões em categorias de sucesso, falha parcial ou falha total. Os resultados indicam que o modelo consegue reconhecer comportamentos recorrentes e inferir a eficácia das missões, validando o uso de aprendizado de máquina como ferramenta de apoio à análise automatizada de simulações. O *plugin* também incorpora um *dashboard* que possibilita a visualização do comportamento das entidades simuladas e de sua evolução ao longo do tempo. Diante disso, confirma-se a viabilidade da integração entre simulação e aprendizado de máquina, destacando o potencial da abordagem para apoiar a verificação e validação de requisitos e a análise de desempenho de sistemas complexos. O *plugin* pode ser empregado também para o estudo de falhas e otimização de estratégias operacionais, servindo como base para futuras expansões em cenários mais complexos e multidomínio.

List of Figures

FIGURE 2.1 – Typical lifecycle flow diagram in Systems Engineering. (Fet; Haskins, 2023) 18

FIGURE 2.2 – Concept of Operations (CONOPS) for aircraft integration into urban airspace, SIMUA-VD Project. (Cerqueira, 2025b) 19

FIGURE 2.3 – Flow diagram of lifecycle in Model-Based Systems Engineering. (Villas *et al.*, 2024) 20

FIGURE 2.4 – Verification and validation cycle among real system, conceptual model and simulation model. (Vangheluwe, 2001) 20

FIGURE 2.5 – Diagram illustrating the k-Nearest Neighbors (KNN) algorithm's operation. (IBM, 2025b) 24

FIGURE 3.1 – Functional architecture of Arena Concept.IO, highlighting the simulation, visualization, modeling and Message Queuing Telemetry Transport (MQTT)-based communication modules. (Cerqueira, 2025a) 27

FIGURE 3.2 – Simulation using VR-Forces software. (MAK Technologies, 2025) 29

FIGURE 3.3 – Example of a sales control dashboard built using the Dash library. (Academy, 2025) 30

FIGURE 3.4 – Operational area defined by geofencing, and helipad exclusion zones. 31

FIGURE 3.5 – Simulation with multiple drones operating simultaneously. 35

FIGURE 3.6 – Drone approaching the delivery point. 38

FIGURE 3.7 – Mission time distribution. 42

FIGURE 3.8 – Mission categories distribution. 42

FIGURE 3.9 – Mission time by conclusion category. 43

FIGURE 3.10 –Heatmap of correlations between numerical variables. 43

FIGURE 3.11 –Relationship between total quantity of evasions and mission time. . 44

FIGURE 3.12 –Learning curve of the KNN model varying the number of neighbors (k). 46

FIGURE 3.13 –Confusion matrix of the KNN classifier (5-fold cross-validation). . . 47

FIGURE 4.1 – Snippet of the drone database file, containing evasion metrics and coordinate logs over time. 50

FIGURE 4.2 – Confusion matrix generated from test data. 52

FIGURE 4.3 – Initial interface of the dashboard with the selection of a specific drone for analysis. 54

FIGURE 4.4 – Selection of a specific order associated to the chosen drone. 54

FIGURE 4.5 – Plots of latitude, longitude and altitude as a function of time for the selected mission. 55

FIGURE 4.6 – Display of additional mission information, including destination, status, stages, and evasion metrics. 55

FIGURE 4.7 – Final classification of the mission based on the extracted metrics. . . 56

List of Tables

TABLE 3.1 – Order life cycle stages 33
TABLE 3.2 – Evasion metrics recorded in the orders 34
TABLE 3.3 – Evasion metrics of partial missions 45
TABLE 3.4 – Distance-to-destination metrics of partial missions 45
TABLE 3.5 – Time and Z-Score of partial missions 45
TABLE 3.6 – Performance scores by class 47
TABLE 3.7 – Accuracy achieved in each partition (fold) 47

TABLE 4.1 – Classification report with metrics by class 52

List of Abbreviations and Acronyms

ABS	Agent-Based Simulation.
CONOPS	Concept of Operations.
IoT	Internet of Things.
JSON	JavaScript Object Notation.
KNN	k-Nearest Neighbors.
KPI	Key Performance Indicator.
MQTT	Message Queuing Telemetry Transport.
SoS	System-of-Systems.
V&V	Verification & Validation.

Contents

List of Abbreviations and Acronyms xi

1 Introduction 14

1.1 Motivation 14

1.2 Objectives 15

1.3 Expected Benefits 15

1.4 Challenges 16

1.5 Structure of the Paper 16

2 Literature Review 17

2.1 Systems Engineering and Concept of Operations 17

2.2 Simulation Concepts 19

2.3 Metrics of Effectiveness 21

2.4 Supervised Learning 23

2.5 The K-Nearest Neighbors Algorithm 23

3 Methodology 26

3.1 Tools 28

3.1.1 MQTT Bus 28

3.1.2 VR-Forces 28

3.1.3 Scikit-learn 29

3.1.4 Dashboard 30

3.2 Domain Context and Simulated Scenario 31

3.2.1 Modification in Entity State Variables Publishing 32

3.2.2 Extension of the `Order` Class . . . 33
3.2.3 Drone Collision Detection . . . 34
3.2.4 Intentional Collision Generation Function . . . 35
3.3 Message Logging Module (MQTT Logger) . . . 36
3.4 Metrics for Evaluating Simulated Missions . . . 38
3.4.1 Total Time in Evasive Maneuvers . . . 38
3.4.2 Number of Successful and Failed Evasions . . . 39
3.4.3 Average Distance and Variability Relative to the Destination . . . 39
3.4.4 Total Mission Time . . . 39
3.4.5 Normalized Mission Time (Z-Score) . . . 39
3.4.6 Mission Status . . . 39
3.5 Machine Learning Model . . . 40
3.5.1 Data Collection and Preparation . . . 40
3.5.2 Mission Categorization . . . 41
3.5.3 Exploratory Data Analysis . . . 41
3.5.4 Model Training and Validation . . . 46
3.6 Metric Visualization . . . 48
4 Discussions . . . 49
4.1 Logger Implementation and Metric Definition . . . 49
4.1.1 System Adaptations . . . 49
4.1.2 Logger Operation and Data Storage . . . 50
4.1.3 Derived Metrics and Findings . . . 51
4.1.4 Challenges and Limitations Faced . . . 51
4.2 Supervised Learning Model - KNN . . . 51
4.3 Dashboard Visualization . . . 53
5 Conclusion . . . 57
Appendix A – Storage in the MQTT Logger . . . 62
A.1 Mission Data Storage Structure . . . 62

1 Introduction

1.1 Motivation

The increasing complexity in engineering systems, due to growing demands for performance, safety, and integration across different areas, makes it increasingly important to have tools that help understand how these systems function as a whole, including their operations and naturally emerging behaviors. Systems Engineering, according to (INCOSE, 2015), is an interdisciplinary approach that integrates various disciplines to ensure that systems are successful, meet their objectives, are resilient, and have minimized risks.

In this context, computational simulation is a highly useful tool for testing ideas, predicting how a system might behave in different situations, and verifying whether requirements are being met throughout the entire project lifecycle. However, as per (Madni; Sievers, 2017), the true value of a simulation lies in the use of metrics of effectiveness, which help measure its accuracy, its ability to predict results, and its utility in decision-making. Having these well-defined metrics is essential for evaluating system performance, identifying possible risks, and making design improvements, as pointed out by (Henzinger, 2013) and (Componation; Dorneich; Hansen, 2015).

How can a plugin assist stakeholders in metric analysis, requirements evaluation, and decision support in simulations? Arena Concept.IO offers an advanced simulation environment, with an event-driven architecture and MQTT bus-based communication. However, it still lacks a specific tool for extracting, analyzing, and visualizing metrics, which limits the ability of engineers and stakeholders to monitor mission progress and validate emergent properties of the simulated systems.

In this scenario, the development of a plugin that integrates data analysis, visualization, and automatic system requirements evaluation emerges as a strategic opportunity to support decisions grounded in Systems Engineering principles. The inclusion of a machine learning model to classify simulations based on extracted metrics represents an additional resource for verifying the fulfillment of defined operational objectives and identifying deviations from expected requirements in complex simulated environments.

1.2 Objectives

The objective of this work is to develop a plugin for the Arena Concept.IO platform that offers stakeholders integrated resources for the real-time monitoring of data transmitted via the MQTT bus, the extraction and logging of the position and behavior of simulated entities over time, the automatic evaluation of pre-defined emergent properties, and the presentation of performance metrics extracted throughout the simulation via a dashboard. The metrics developed from this data will be used to build a machine learning model aimed at predicting the success of the simulations, capable of classifying them as successful, partially failed, or failed. This aims to provide objective evidence to support decision-making during the design or analysis process.

1.3 Expected Benefits

The implementation of the proposed plugin yields a series of benefits that encompass technical, operational, and strategic aspects, thus promoting an integrated approach to supporting complex systems. Firstly, the structured extraction of metrics from simulations allows for more informed decisions, based on concrete data, which facilitates both requirements validation and monitoring of systemic goal achievement (Madni; Sievers, 2018). Furthermore, the ability to monitor critical variables from the logged data significantly contributes to the identification of failures and anomalous behaviors, favoring the mitigation of operational and design risks (Brown; Conrad; Beyeler; Glass, 2013; INCOSE, 2015).

Regarding efficiency, the plugin makes it possible to identify bottlenecks and inefficiency points, allowing for adjustments in parameters and strategies that result in performance improvements (Cho; Hurley; Xu, 2016). The automation of emergent property verification, through defined metrics and machine learning models, also represents a relevant advancement by reducing the effort required for manual evaluation in the analysis of requirements compliance.

The consolidation of data into interactive dashboards and reports represents another important benefit, as it strengthens confidence in the decisions made and facilitates the understanding of occurrences during the simulation. The accessibility of information, in turn, favors collaboration among the various stakeholder profiles involved, such as engineers, analysts, and managers, promoting more effective communication.

Finally, the use of metrics in different phases of the system lifecycle, such as validation, operation, and support, reinforces the fundamental principles of Systems Engineering. This data reuse not only adds value to post-simulation analysis but also contributes to

the continuity and consistency of systemic development throughout the entire process (Fet; Haskins, 2023).

1.4 Challenges

The development of this work involves a series of technical and conceptual challenges. Integration with the Concept.IO API requires knowledge of the platform's architecture, including, in some cases, the need for reverse engineering to correctly access and interpret the data transmitted via the MQTT bus. Furthermore, the complexity and volume of the transmitted messages represent another obstacle, as high traffic can compromise analysis performance, requiring the use of more complex processing and filtering strategies. Another relevant challenge lies in defining and validating the metrics used: it is important that these are representative of the system's objectives and reliably measurable (Cho; Hurley; Xu, 2016), in order to ensure the credibility of the results.

With respect to the application of machine learning models, it is necessary to deal with issues such as data quality and labeling, the appropriate selection of attributes, and the risk of overfitting in small or noisy datasets. These aspects directly impact the model's ability to generalize patterns and provide useful classifications to support decisions.

Finally, the transformation of this data into comprehensible visual representations, through interactive and informative dashboards, requires attention to the design and usability of the interface, especially considering the different profiles of the users involved.

1.5 Structure of the Paper

This work is organized into six chapters. Chapter 1 presents the motivation, objectives, benefits, and expected challenges in the construction of the plugin. Chapter 2 addresses the theoretical foundations necessary for understanding the project, including concepts of Systems Engineering, agent-based simulation, metrics of effectiveness, and supervised machine learning. Chapter 3 describes the methodology adopted, including the operational environment, the simulation scenario based on the SIRESANT project, the modeling of drone missions, as well as the processes of simulation instrumentation, data collection, metric definition, and application of the predictive model. Chapter 4 presents the results obtained, including a quantitative analysis of the metrics, the performance of the classifier, and the visualization on the dashboard. Finally, Chapter 5 brings together the conclusions of the work, reviews its limitations and proposes directions for future studies.

2 Literature Review

2.1 Systems Engineering and Concept of Operations

Systems Engineering is an integrative approach focused on the conception, development, operation, and sustainment of complex systems. It promotes the integration of knowledge from various disciplines to generate a holistic view of the system and focuses primarily on the interactions between the parts, bringing new perspectives on system complexity, in addition to studying the emergent behavior that arises from these interrelationships. According to (INCOSE, 2015), the goal is to ensure that the developed systems are fit for the purpose for which they were conceived, minimizing side effects and unintended consequences during their operation in the real world.

In the context of Systems Engineering, there is continuous involvement of interested parties, referred to as stakeholders, throughout the entire system lifecycle. These parties may include end-users, operators, developers, regulatory bodies, among others, and their contributions are fundamental for guiding development from the initial identification of needs to the final system verification. According to (INCOSE, 2015), their needs, expectations, and constraints must be properly understood, analyzed from different perspectives (technical, operational, and regulatory), reconciled through viable compromises between conflicting interests, and then translated into clear, traceable, verifiable, and prioritized requirements, serving as the basis for the development and evaluation of the proposed solutions.

Figure 2.1 illustrates a typical sequence of steps in this process, starting with the identification of stakeholders and the elicitation of their needs, followed by requirements definition, performance specification, analysis and optimization, design and refinement, and verification and reporting. Interaction with stakeholders occurs iteratively and continuously throughout all these phases, ensuring alignment between expectations and developed solutions. This structure is merely a representative example, as the number and nature of the steps may vary depending on the type of system, the application domain, and the specific needs of the project.

Within the initial activities of systems development, the Concept of Operations, also

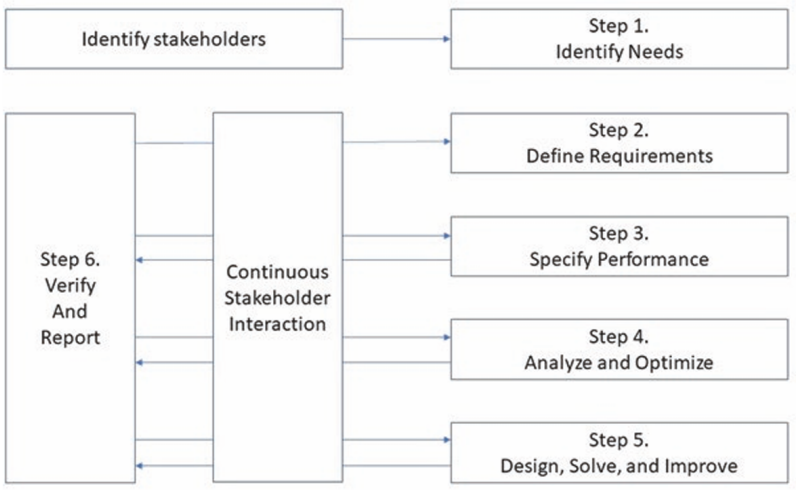

FIGURE 2.1 – Typical lifecycle flow diagram in Systems Engineering. (Fet; Haskins, 2023)

known as CONOPS, constitutes an important element for communication and alignment among the stakeholders. It is a structured document that describes, in accessible language and from the user's perspective, how the system will be utilized in its predicted operational environment. A CONOPS goes beyond a functional description, as it provides a strategic vision that articulates operational objectives, typical and exceptional use scenarios, user profiles, expected functionalities, operational processes, interactions with other systems, and environmental factors that may impact performance (Engineering, 9 fev. 2025).

Furthermore, the CONOPS clearly defines the system's capabilities, both functional and non-functional, the methods of support and maintenance throughout its lifecycle, as well as the performance indicators that will be used to measure its effectiveness and efficiency. Therefore, it acts as a critical reference during all project phases, from initial conception to the system's operation and evolution. Its elaboration contributes to the precise formulation of requirements, underpins acceptance criteria and testing, and guides design decisions based on a unified understanding among all parties. This shared vision helps prevent ambiguities, reduces the risk of undesirable scope changes, and ensures that the final system effectively meets real operational needs.

Figure 2.2 represents the operational flow for the development and validation of systems for aircraft integration into urban environments, supported by the Arena Concept.IO simulation platform. The scheme illustrates how different stages, from initial conception to testing in simulated scenarios, are connected through digital models, communication via the MQTT bus, and dashboards for performance analysis, within the scope of a project. Its objective is, therefore, to ensure that the proposed scenarios can be tested, monitored, and evaluated in real time, focusing on stakeholder decision-making.

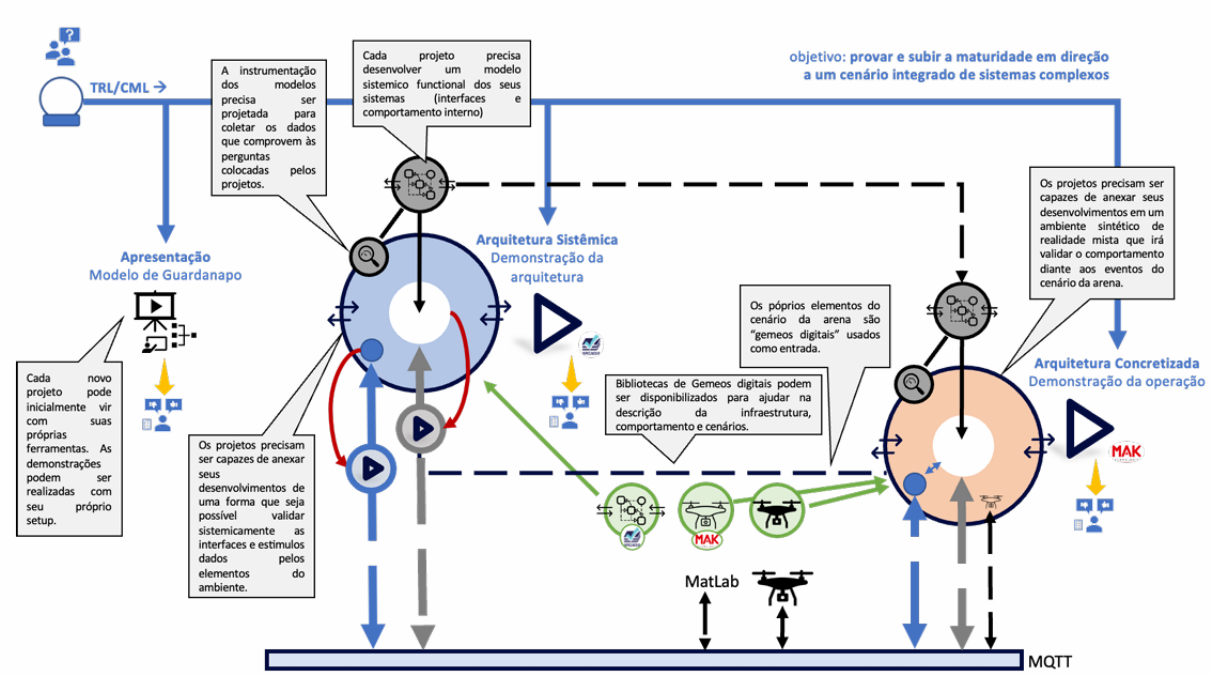

FIGURE 2.2 – CONOPS for aircraft integration into urban airspace, SIMUA-VD Project. (Cerqueira, 2025b)

2.2 Simulation Concepts

Simulations constitute an important tool in the development and evaluation of complex systems, allowing for the execution of virtual experiments in controlled and repeatable environments. In the context of Systems Engineering, simulation is used to anticipate behaviors, validate requirements, explore design alternatives, and support decision-making throughout the entire system lifecycle. According to (Vangheluwe, 2001), to simulate is to conduct virtual experiments with models that are representative of real systems, within an experimental framework that defines the objectives, variables of interest, and boundary conditions of the simulation.

Arena Concept.IO, the focus of this paper, precisely represents this approach: it is a distributed simulation environment that allows for the modeling and observation of complex real-world system operations, offering an interface for stakeholders to configure and monitor varied mission scenarios. These stakeholders can use simulation as a strategic resource to anticipate system behavior under different conditions, assess risks, and validate whether operational objectives are achievable.

The simulation process, as described by (Gianni; D'Ambrogio; Tolk, 2014), begins with the construction of a conceptual model of the system, which is then transformed into a computational model capable of being executed. This cycle includes verification activities (ensuring that the model has been correctly implemented) and validation (confirming that the model adequately represents the reality intended to be simulated). In environ-

ments like Arena Concept.IO, this structure allows scenarios to be created from varied operational configurations, simulating both normal operations and exceptional situations.

Agent-Based Simulation (ABS) holds significant relevance in this context, as it allows for the representation of autonomous entities with their own behaviors and objectives, interacting in a complex and emergent way. As demonstrated by (Villas *et al.*, 2024), this type of simulation is effective for representing System-of-Systems (SoS) type systems, in which multiple independent systems interact to achieve collective objectives, such as in emergency response operations or coordinated missions. The use of structured approaches for operational modeling allows different views of the system to be organized and facilitates the exploration of alternative CONOPS by stakeholders, enabling real-time observation of the impact of their decisions.

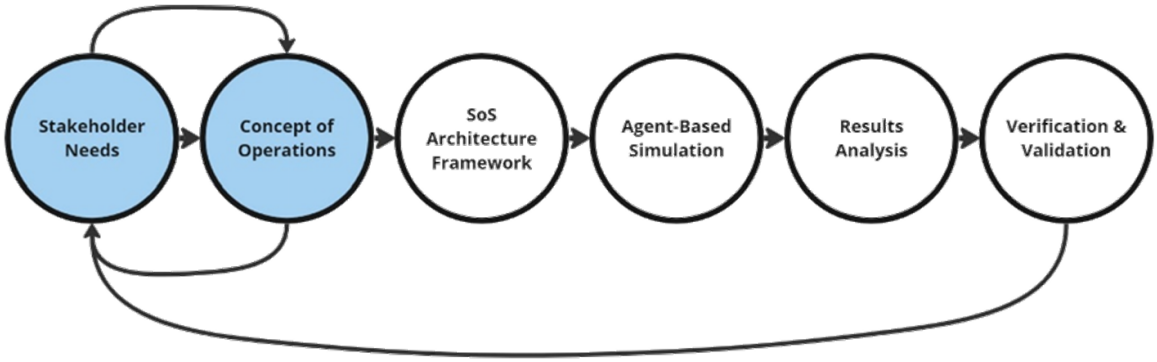

FIGURE 2.3 – Flow diagram of lifecycle in Model-Based Systems Engineering. (Villas *et al.*, 2024)

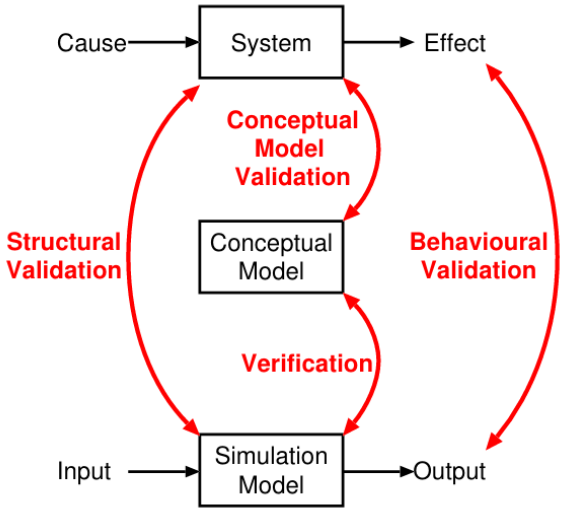

FIGURE 2.4 – Verification and validation cycle among real system, conceptual model and simulation model. (Vangheluwe, 2001)

In this scenario, simulation not only reproduces the system's behavior but also acts as an essential instrument for validating the operational strategies described in the CONOPS.

Arena provides a platform to collaboratively develop, test, and communicate these strategies, integrating multiple stakeholders. The data generated during the simulation, when analyzed with appropriate tools, can provide insights into performance, safety, and operational effectiveness. This data can be used to verify whether objectives are being met as expected (through the use of metrics) or whether adjustments are needed in the CONOPS, the system configuration, or the strategies adopted. This enables iterative cycles of validation, refinement, and informed decision-making, increasing the alignment between planning and the simulated operational reality, as seen in Figure 2.4.

2.3 Metrics of Effectiveness

In the context of Systems Engineering, metrics of effectiveness are important for evaluating a system in relation to its operational objectives. Effectiveness is related to the system's ability to fulfill its mission; that is, to achieve the expected results under determined conditions of use, differing from efficiency, which focuses on the relationship between resources utilized and results obtained (Press, 2025). Effectiveness is concerned with whether the system successfully solves the problem for which it was designed. This distinction is relevant in complex systems (INCOSE, 2015), where it is possible for a solution to be efficient in terms of resources but ineffective from the perspective of the mission or some stakeholders, such as the end-users.

To ensure that a system is effective, it is necessary to define objective criteria that allow its performance to be measured throughout the lifecycle. In this sense, metrics of effectiveness act as essential instruments in the Verification & Validation (V&V) processes. Verification is responsible for ensuring that the system has been built correctly with respect to the specified requirements, while validation confirms that the system meets the stakeholders' needs in the intended operational environment. Based on the principles presented by (Fet; Haskins, 2023), such metrics allow for the evaluation of functional and non-functional requirements fulfillment, validation of design hypotheses, and identification of behavioral deviations in simulations.

In complex systems, simulations offer critical support for decision-making by allowing the anticipation of system behavior under different operational conditions. They make it possible to test strategies, validate requirements, and explore extreme scenarios without the risks or costs of real experimentation. To reliably fulfill this function, it is essential that system performance be monitored continuously during simulated execution. This requires defining appropriate metrics, aligned with objectives and integrated into the virtual environment through tools, which depend on the context in which the simulation is inserted. The correct interpretation of these results relies on a well-defined analytical

structure that directly relates the stimuli applied, the data collected, and the evaluation criteria, ensuring that the simulation provides relevant and applicable information to the project's decision-making context.

The metrics used in simulations vary according to the system objectives and the level of fidelity required for the analysis. Among the most common are operational performance metrics (Fet; Haskins, 2023), which aim to capture the efficiency with which system functions are executed. Typical examples include event response time, area coverage in distributed operations, and processing capacity, which measures the amount of tasks or messages processed per unit of time. These metrics are particularly useful for evaluating aspects such as scalability, latency, and the fluidity of the simulated operation.

Another important group are mission metrics (INCOSE, 2015), aimed at evaluating whether the system's final objectives have been achieved. They include the percentage of task completion success, the number of failures that occurred during execution, and the degree of adherence to the operational plan. These metrics are important in results-oriented scenarios, such as search and rescue missions, cargo delivery, or defense, where the focus is on mission completion with safety and reliability.

Resilience and adaptability metrics complement the analysis by assessing how the system responds to failures, environmental changes, or component degradation. The time required to resume normal operations after an interruption and the ability to adapt to new operating conditions are examples. These metrics are important for systems operating in dynamic or uncertain environments, such as urban operations involving autonomous vehicles and remotely piloted aircraft.

Performance indicators, also known as Key Performance Indicator (KPI)s (Fet; Haskins, 2023), are defined based on system requirements and serve as benchmarks for continuous evaluation. They synthesize operational objectives into measurable values, allowing for comparison between different configurations, architectures, or strategies tested in simulations. The proper choice of KPIs is necessary to ensure that the metrics directly reflect the expectations of the stakeholders and serve as a reliable basis for design decisions.

Defining relevant metrics in simulations requires criteria that ensure their utility and reliability. They must be clear and objective (INCOSE, 2015), avoiding ambiguity in the interpretation of results. According to (Fet; Haskins, 2023), they also need to be measurable during simulation execution, allowing for automatic collection and consistent analysis. Furthermore, they must be directly relevant to the system's objectives, reflecting critical aspects of its performance. Finally, metrics need to have traceability to the defined requirements and expected emergent properties, ensuring coherence between what is measured and what is expected of the system in operation.

2.4 Supervised Learning

Supervised machine learning is one of the main approaches in the field of artificial intelligence, characterized by the use of algorithms trained on labeled data. In this modality, each input in the training set is associated with a known output, allowing the model to learn to map relationships between variables with the goal of making future predictions or classifications (IBM, 2025a).

During the training process, the model seeks to reduce the difference between predicted and actual values through the optimization of error functions. This iterative adjustment allows the system to generalize the acquired knowledge to previously unobserved data. After training, the model's performance is evaluated with separate data, ensuring its capacity for generalization and avoiding overfitting.

Among the main tasks of supervised learning are classification, which involves categorizing inputs into discrete classes, and regression, focused on predicting continuous values. The choice between different algorithms such as decision trees, logistic regression, support vector machines (SVM), or k-nearest neighbors (KNN) depends on factors such as the nature of the data and the analytical objectives.

In the context of evaluating the classification model developed in this work, four main metrics are commonly used (Kashyap, 2024). Precision indicates how many of the predicted positive cases are correct, while recall measures how many of the actual positive cases were correctly identified. The F1-Score combines both measures into a single value, useful when dealing with class imbalance. Lastly, support represents the number of samples belonging to each class in the dataset.

2.5 The K-Nearest Neighbors Algorithm

The k-Nearest Neighbors (KNN) algorithm is a supervised machine learning method that is non-parametric and instance-based, used for both classification and regression tasks (IBM, 2025b). Its operation is based on the assumption that examples close in the feature space tend to belong to the same class or share similar values. In practical terms, the algorithm assigns a label to a new instance based on the labels of the k closest examples in the training set.

In the case of classification, KNN uses a "majority voting" strategy to determine the most frequent class among the neighbors. For regression problems, the output value is obtained by averaging the output values of the k nearest neighbors. An important characteristic is that the algorithm does not perform an explicit training phase: all training data is stored and used directly at the time of prediction, which characterizes it as a lazy

learner method.

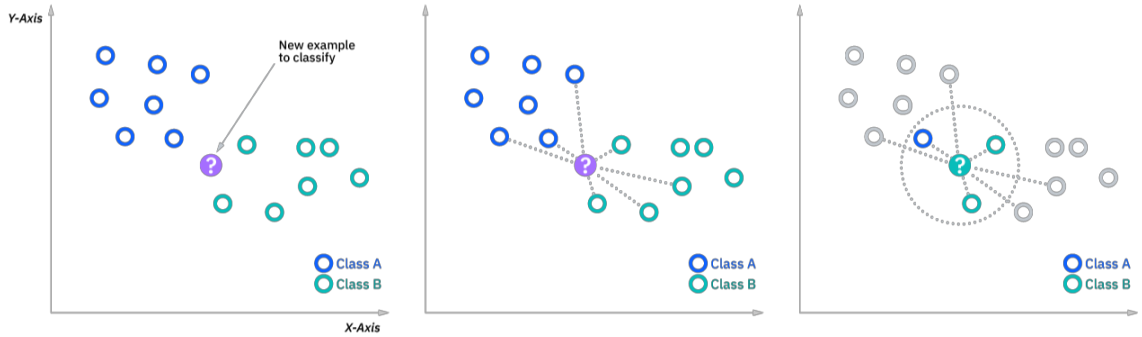

FIGURE 2.5 – Diagram illustrating the KNN algorithm's operation. (IBM, 2025b)

The measure of similarity between samples is a central component in the algorithm's performance. The most common distance metrics include:

$$d(x, y) = \sqrt{\sum_{i=1}^{n}(x_i - y_i)^2} \tag{2.1}$$

Another common option is the Manhattan distance (also known as the L_1 distance), calculated by:

$$d(x, y) = \sum_{i=1}^{n}|x_i - y_i| \tag{2.2}$$

More generally, both can be seen as particular cases of the Minkowski distance, expressed by:

$$d(x, y) = \left(\sum_{i=1}^{n}|x_i - y_i|^p\right)^{\frac{1}{p}} \tag{2.3}$$

The choice of the value of p defines the metric: when $p = 1$, the Manhattan distance is obtained; for $p = 2$, the Euclidean distance.

The choice of both distance metric and the value of k (the number of neighbors) can significantly influence the model's performance, particularly when dealing with high-dimensional data.

Among the algorithm's main advantages are its simplicity of implementation, the absence of assumptions about data distribution (due to being *non-parametric*), and its adaptability to both classification and regression problems. On the other hand, its main limitations include the necessity to store the entire training set in memory, the high computational cost in large data volumes, and sensitivity to irrelevant attributes or variable

scaling, thus recommending use of normalization and selection of characteristics.

The application of this algorithm in the context of simulations in agent-based systems, such as those observed in the Arena Concept.IO platform, presents a possibility of utilizing machine learning methods to capture emergent patterns in complex and dynamic scenarios. Within the scope of Systems Engineering, this type of approach perfectly aligns with systemic thinking and the need for global behavior analysis based on the interaction between individual components.

3 Methodology

Arena Concept.IO is a platform focused on collaborative simulation of complex systems, with an emphasis on integrating multiple operational domains, including land, naval, air, space, social and cyber contexts. This multi-domain approach enables different stakeholders, with systems at various stages of maturity, to interact in a common scenario, fostering the exploration of operational concepts, tactics, and procedures in a controlled and adaptable environment.

Within this architecture, the VR-Forces simulator plays a central role in executing the scenarios, being responsible for simulating the dynamic behavior of entities and generating the events that occur in the virtual environment. It models, in real time, the actions and interactions of the simulated entities and transmits the resulting data through the MQTT communication bus. Arena Concept.IO, in turn, functions as the integration and orchestration platform, connecting different simulation systems and analysis tools within a collaborative multi-domain environment. This infrastructure provides the necessary basis for collecting and processing the information generated during the simulations.

Based on the presented architecture, the objective of this project is to develop a plugin that enables the analysis of data originated from the MQTT, promoting the structured collection of relevant information for monitoring and evaluating the performance of the simulated system. From this data, specific metrics will be developed to quantify emergent properties and assess the fulfillment of the operational objectives defined for the scenario. The metric calculations will be implemented using the Python programming language, leveraging its robustness for data manipulation and analysis, as well as its simplicity for script development.

The information extracted from the bus is currently stored in JavaScript Object Notation (JSON) files, serving as a preliminary structure for future modeling in a document-oriented NoSQL database, such as MongoDB (MONGODB Inc., 2025). This approach is well-suited to the dynamic and heterogeneous nature of the data generated during the simulations, which may vary in structure depending on the scenarios, entities, and metrics involved. The flexibility of this type of system allows for the storage of complete records in independent documents, without the rigidity of fixed schemas, facilitating scalability

and integration with analytical tools.

The ingestion and analysis pipeline will be implemented in Python, with results visualized through an interactive dashboard developed using the Dash library. This interface will enable stakeholders to intuitively explore the data generated during the simulation through the upload of files containing the extracted records from the communication bus, allowing for the post-simulation evaluation of system behavior and informed decision-making.

In addition to data visualization, these records will also serve as the basis for the development of a supervised machine learning model capable of classifying simulations according to their degree of success (as successful, partially failed or failed). This classification will be based on metrics extracted during scenario execution, allowing for the identification of recurring patterns in the behavior and performance of system entities, specifically the drones. A supervised learning algorithm, KNN, will be used due to its simplicity and efficiency in identifying similarities within labeled datasets. The goal is to provide stakeholders with an additional decision-support tool capable of anticipating trends and highlighting potential risks or inefficiencies in the tested operational strategies.

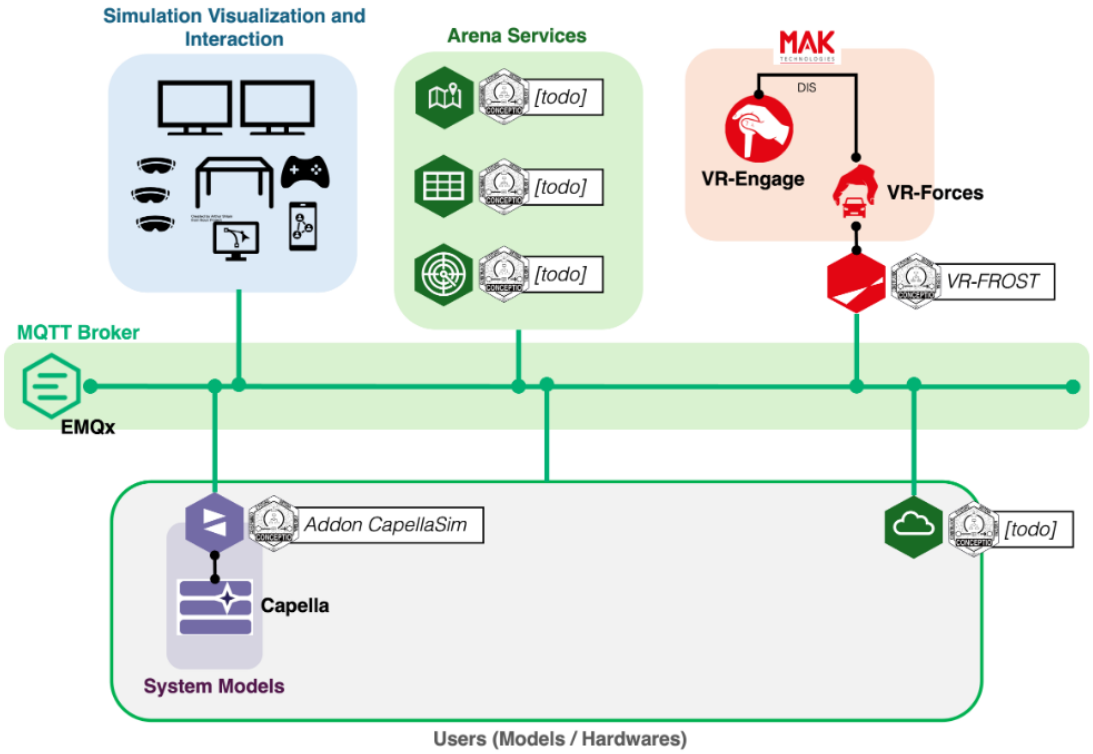

FIGURE 3.1 – Functional architecture of Arena Concept.IO, highlighting the simulation, visualization, modeling and MQTT-based communication modules. (Cerqueira, 2025a)

3.1 Tools

3.1.1 MQTT Bus

From a Systems Engineering perspective, Concept.IO presents itself as a solution for orchestrating distributed simulations, in which each component of the simulated system represents an autonomous functional element (independent node). Communication between these elements is carried out through a data bus based on the MQTT protocol (Cerqueira, 2025a), which enables continuous information exchange among the various subsystems, ensuring interoperability and consistency in the emergent behavior of the system as a whole.

MQTT is a lightweight protocol widely used in distributed systems such as Internet of Things (IoT), and is particularly well-suited for scenarios that require fast and reliable responses even under network constraints (MQTT.org, 2025). Its publish-subscribe architecture promotes decoupling between data producers and consumers, allowing simulation elements to operate in a coordinated yet independent manner. This model facilitates system scalability and supports modularity—key features for managing complexity and requirements improvement.

Furthermore, the structure of MQTT enables continuous message exchange between simulation elements, in accordance with arena's distributed architecture. Since the broker in use does not perform persistent storage, the platform currently prioritizes real-time execution. This limitation hinders post-simulation analysis and the reproducibility of results, which motivated the implementation of a structured data logging mechanism within the plugin, responsible for capturing, organizing and storing the messages exchanged during the simulation.

3.1.2 VR-Forces

VR-Forces, part of the MAK ONE suite of applications, is a simulation platform that covers a wide range of environments—from the ocean floor to outer space (MAK Technologies, 2025). It was designed to represent entities, their relationships and the effects of the environment in complex operations. One of its main features is the ability to simulate multiple domains simultaneously, with a focus on real-time control and monitoring, meeting the stakeholder needs previously mentioned. Its modular structure separates the simulation engine from the user interface, which facilitates the development of tools. This separation allows data to be collected via MQTT and processed analytically.

VR-Forces is capable of creating complex scenarios with multiple entities and automating events, which aligns perfectly with the plugin's objective of identifying emergent prop-

erties and key events during simulations. Moreover, it supports environmental variables that change over time, such as local weather conditions, and is capable of working with real-time data. This enables it to interpret environmental changes that may affect the performance of the simulated system. One of the platform's core functions is the synchronization matrix, which helps divide the scenario into different time phases. This makes it possible to plan coordinated actions based on specific events or conditions over time. In this context, having a well-structured temporal logic is essential to identify patterns and verify whether the established objectives are being achieved.

FIGURE 3.2 – Simulation using VR-Forces software. (MAK Technologies, 2025)

Finally, the architecture employs parallel processing to perform real-time calculations in VR-Forces, serving as an important reference for those looking to implement strategies for processing large volumes of messages transmitted via the MQTT bus. This is particularly relevant given the high-performance requirements for capturing, interpreting, and analyzing data in real time. The combination of these technical capabilities provides a solid foundation for development, especially when it comes to integrating different domains, dynamically evaluating scenarios and delivering qualified analytical support for decision-making in simulated environments.

3.1.3 Scikit-learn

To implement the classifier responsible for predicting the status of missions, the Scikit-learn library was used, which is widely adopted in Python-based machine learning projects. This library provides a unified and efficient interface for various classification, regres-

sion, and clustering algorithms, as well as tools for preprocessing, feature selection, cross-validation, and performance evaluation (Habbema, 2024).

The classifier adopted in this work is the KNN, a supervised algorithm based on similarity between samples in the feature space. The choice of this model is due to its suitability for the type of problem at hand: the classification task involves a compact and well-defined set of metrics extracted directly from the simulations, which favors approaches that operate in low-dimensional spaces. Additionally, KNN is a non-parametric model (Scikit-learn Developers, 2024), which eliminates the need for assumptions about data distribution and allows for quick adaptation to new examples. Its interpretability is also a significant advantage, especially in contexts where decision transparency is desirable, such as in the performance analysis of simulated systems.

3.1.4 Dashboard

Dash is a Python-based library designed for building interactive web applications with a strong emphasis on data visualization (Plotly Technologies Inc., 2025). Developed on top of the Flask framework (for the backend) and React.js (for the frontend), it allows engineers and data scientists to create sophisticated dashboards in a simple and rapid manner. This characteristic makes it ideal for the project proposed in this work, which requires presenting results clearly and dynamically to stakeholders.

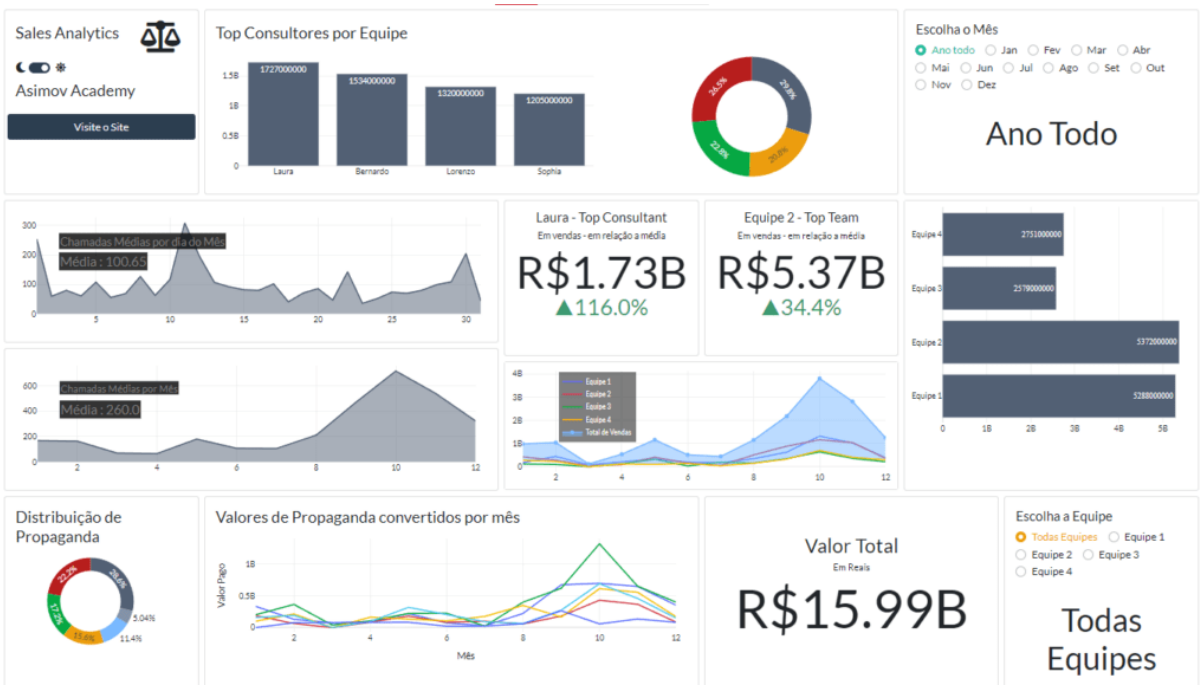

FIGURE 3.3 – Example of a sales control dashboard built using the Dash library. (Academy, 2025)

In the context of this paper, the Dash library will be used to build a graphical interface that enables retrospective analysis of the data generated during simulations, based on files

exported from the MQTT bus. This data will be presented through interactive charts that depict the evolution of spatial coordinates over time, allowing for the monitoring of simulated entity behavior. In addition to visualization, Dash also provides features for interacting with the data, such as filters, sliders, and menus, which allow users to adjust parameters and explore the contents of the simulations.

3.2 Domain Context and Simulated Scenario

FIGURE 3.4 – Operational area defined by geofencing, and helipad exclusion zones.

For the development and validation of the plugin, the scenario of autonomous drone delivery was selected. This is a representative domain of systems with emergent behavior, chosen due to its operational complexity and the presence of characteristic properties of Systems Engineering, such as distributed coordination, variability in mission time, and adaptive response to failures.

The scenario of autonomous drone delivery was selected for the plugin's development and validation. This domain is representative of systems with emergent behavior, chosen owing to its operational complexity and the presence of characteristic Systems Engineering properties, such as distributed coordination, variability in mission time and adaptive response to failures.

The plugin was developed based on a subset of an ongoing project, SIRESANT, by utilizing the provided simulation code specifically aimed at the study of autonomous deliveries. This code models missions where drones perform deliveries in an urban environment, simulating trajectories between stores and drop-off points, and transmitting data via the MQTT bus. The published messages contain information such as the drones' position, delivery status, identification of stores, drop-off points and relevant events over time.

Figure 3.4 shows the simulated region, which corresponds to a section of the urban grid of Curitiba, Brazil. The blue perimeter represents the drone operating area defined by geofencing, while the orange volumes indicate no-fly zones around helipads, where drone traffic is restricted.

To construct the simulation metrics, it was necessary to implement changes and add new features to the original simulation code, as described in the following subsections.

3.2.1 Modification in Entity State Variables Publishing

Two main modifications were made to the `Entity` class to enhance temporal tracking and the association between kinematic data and delivery orders.

The first change involved adding the `timestamp` field, which records the exact moment the data is generated, in nanoseconds. This level of temporal precision enables tighter synchronization of simulation events, allowing for detailed analysis such as measuring intervals between consecutive publications (e.g., entity movements) and calculating the total duration of missions.

The second change introduced the `order_ref` field in the message published by the `publish_kinematics` method. This field references the delivery order ID associated with the drone at the time of publication, making it possible to correlate movement data with the corresponding active order. This modification was essential for the later analysis of delivery-related metrics, such as travel time, route efficiency and mission completion.

In the `Drone` class, which inherits characteristics from the `Entity` class, new fields were added to the message published by the `publish_heartbeat` method to enhance monitoring of the drones' operational state during simulation.

The first addition was the `timestamp` field, which captures the exact moment the data is generated, in nanoseconds, enabling synchronized analysis with other system data.

Next, the `status` field was added to indicate the drone's current state. Possible values include `idle` (available), `busy` (on mission), and `crashed` (collision). This field provides a clear view of the drone's operational cycle and supports the calculation of metrics related to resource utilization and availability.

Finally, the `order_ref` field was included in the heartbeat message, allowing the drone to be associated with a specific order even outside the kinematics stream. This information is essential for tracking the link between entities and tasks over time.

3.2.2 Extension of the `Order` Class

The `Order` class, responsible for representing delivery requests during the simulation, was extended to include detailed information about delivery progress and the operational performance of the associated drone. The implemented modifications aim to make each order instance traceable over time and integrated into the monitoring system.

The first addition was the `stages` attribute, a list that stores, in chronological order, the key events related to the order's lifecycle.

TABLE 3.1 – Order life cycle stages

Stage	**Stage (English)**	**Description**
`preparando_pedido`	preparing_order	The order has been created and is being prepared at the store.
`pedido_pronto`	order_ready	The item is ready for pickup.
`decolando`	taking_off	The drone has begun takeoff.
`entregando`	delivering	The delivery at the destination point has started.
`entregue`	delivered	The item has been delivered.
`entrega_concluida`	delivery_completed	The mission was successfully completed, including the drone's return to the store.
`retornou_loja`	returned_to_store	The drone has completed the logistics process, returned to the store, and is available for new deliveries.

The fields `destination` and `store_name` were also added to enable the publication of this information on the MQTT bus, allowing the origin and destination of each order to be explicitly identified in the transmitted messages.

Another added field is `drone_id`, which explicitly links the order to the drone responsible for its execution. This association is essential for correlating drone movement data with order events.

Additionally, the `evasion_metrics` structure was implemented to record statistics related to evasion maneuvers during the mission.

TABLE 3.2 – Evasion metrics recorded in the orders

Field	Description
`collision_avoidance_times`	Execution times of the maneuvers executed to avoid collisions with other drones; each value corresponds to an occurrence in `collision_evasions`.
`helipoint_avoidance_times`	Execution times of the maneuvers performed to avoid helipad restricted zones; each value corresponds to an evasion recorded in `helipoint_evasions`.
`collision_evasions`	Total quantity of successful maneuvers to avoid collisions.
`helipoint_evasions`	Total quantity of successful maneuvers to avoid helipad exclusion zones.
`failed_collision_evasions`	Counter of failed collision avoidance maneuvers, which result in the drone's movement being interrupted.
`failed_helipoint_evasions`	Counter of failed helipad avoidance maneuvers, also resulting in the interruption of the drone's movement.

To consolidate this information in real time, the `publish_order_update` method was created to publish the updated state of the order to the corresponding topic via the bus. The message includes the order ID, status, stages, evasion metrics, store and destination names and the associated drone.

The extension of the class allows for more detailed tracking of each order's progress, identification of operational failure causes and the collection of relevant metrics to evaluate the performance of each mission.

3.2.3 Drone Collision Detection

To realistically represent the behavior of multiple drones operating simultaneously, a mechanism for detecting direct collisions between aircraft was implemented during the simulation. The `detect_drone_collision` function is designed to identify the moment when two drones occupy positions that are sufficiently close in space, characterizing a collision.

The detection is based on a continuous comparison of the positions of all active drones in the scenario. For each drone, the function iterates through the `all_drones_positions` dictionary, which stores the current coordinates of all vehicles and calculates the geodesic distance between them. A collision is confirmed when two criteria are met simultaneously:

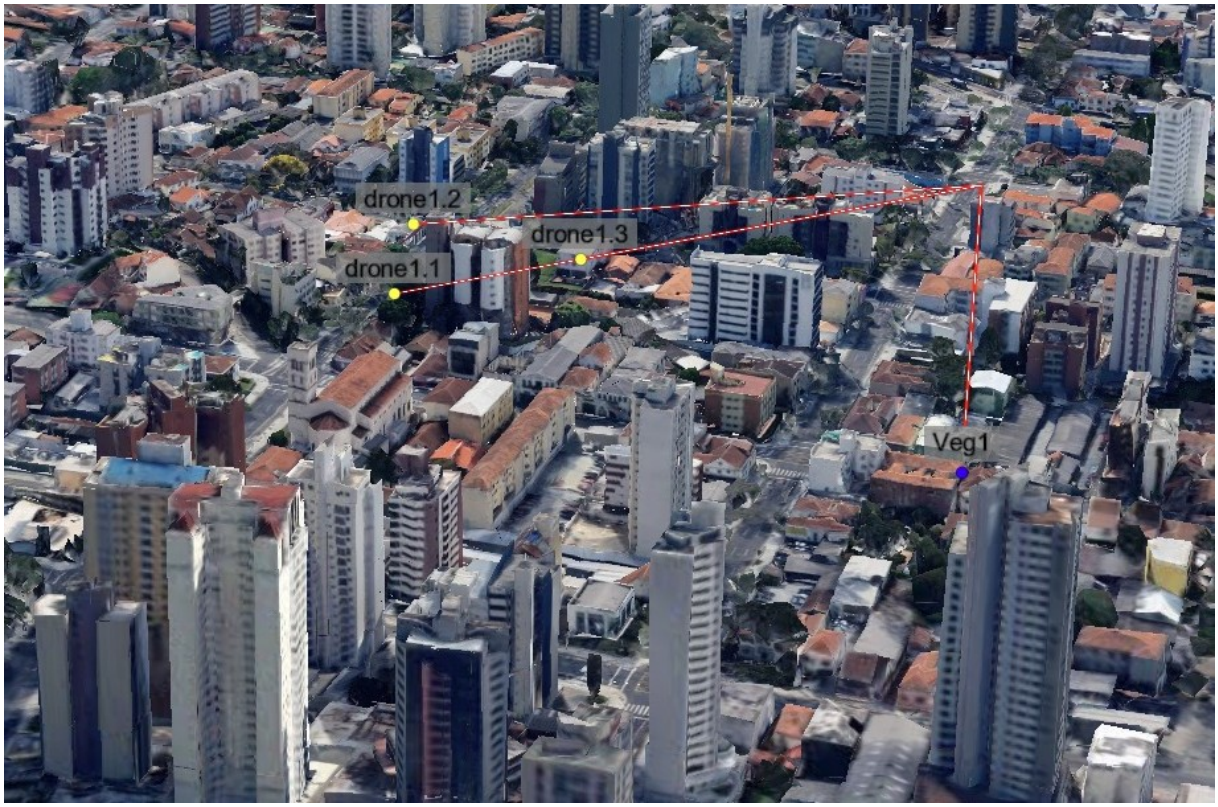

FIGURE 3.5 – Simulation with multiple drones operating simultaneously.

- **Horizontal distance**: the distance between the drones, calculated from their geographic coordinates, is less than or equal to 0.7 meters;
- **Altitude difference**: the difference in altitude is less than 2 units, based on the conversion factor applied to the altitude measurement in the code.

When these conditions are met, the function returns `True`, indicating that a collision has occurred between the drones.

The "True" return value is used to immediately halt the flight routines of the drones involved, update their status to `crashed`, and log the collision event on the MQTT bus so that the monitoring system and the plugin can react and record the incident.

3.2.4 Intentional Collision Generation Function

To validate the robustness of the collision detection and response mechanisms implemented in the system, an auxiliary function named `launch_suicidal_drone` was developed. This function is intended to induce controlled failures in the simulation by intentionally forcing collisions between drones in order to verify the system's response and the accuracy of incident-related metrics.

The function is automatically triggered by the `schedule_suicidal_launch` routine,

which runs periodically every 10 seconds. When activated, this routine dynamically instantiates a suicidal drone 3.5 meters north of a preselected target drone, maintaining the same altitude. The activation of this feature occurs with a 50% probability, randomly defined at the beginning of the simulation. The suicidal drone's trajectory is calculated based on the direction from its initial position to that of the target, and the collision is detected through periodic position overlap checks using the `detect_drone_collision` function.

Once the collision is confirmed, the states of both drones are updated to `crashed`, and the associated orders are marked with the `collided` stage. An event containing collision data, including the identifiers of the drones involved, timestamp and geographic location, is published to the corresponding topic.

Implementing this functionality was important for simulating drone collision scenarios in a controlled manner and analyzing the simulator's behavior in response to critical events. Through these induced failures, it was possible to validate the operation of monitoring, logging and state update mechanisms, as well as to verify the correct generation of incident response metrics within the simulated environment.

3.3 Message Logging Module (MQTT Logger)

To enable comprehensive monitoring of the simulations run on the Arena Concept.IO platform, a dedicated module was developed for the continuous capture and storage of messages transmitted over the MQTT bus. Its main goal is to reliably log all relevant events during the simulation, allowing for detailed analysis of the behavior of simulated entities, especially the drones.

The system is composed of two main components. The first, implemented in the `MessageLogger` class, is a generic logging mechanism responsible for recording all messages published on MQTT, regardless of their type or origin. Messages are organized by topic and stored in a structured file, forming a complete simulation log. This log makes it possible to reconstruct the event timeline, inspect data traffic and support investigations into system behavior under different scenarios.

The second component, implemented in the `DroneDatabase` class, is specifically designed to track drone missions. It extracts and records key mission information such as position over time, operational status, stages and metrics related to obstacle avoidance or collision attempts. This data is stored separately in a local database organized by drone and mission identifiers, enabling in-depth analysis of performance and emergent behavior of the simulated entities.

The data storage structure for drone missions is based on a hierarchical JSON format, in which each drone is identified by its `drone_id` and linked to a specific order through the `order_id` field. Based on this composite key, various aspects of the drone's operation are recorded, forming a complete and individualized history of its activity throughout the simulation.

The stored data includes:

- **Kinematic coordinates** (`coord`): represent the drone's flight path, containing the time-sequenced latitude, longitude, altitude and timestamp. This data is updated by the `update_kinematics` function in the simulation code and allows the reconstruction of movement at any point during the mission.
- **Mission destination** (`destination`): defined through the `set_destination` function, represents the expected final position and is used as a reference to evaluate delivery success.
- **Current aircraft status** (`status`): such as `idle`, `busy`, or `crashed`, updated by the `update_status` function and essential for determining the final state of the flight.
- **Mission stages** (`stages`): recorded via `update_stage`, describe the progress of the operation based on the milestones defined in the delivery flow.
- **Evasion metrics** (`evasion_metrics`): such as attempts and failures to avoid collisions or helipads, updated through `update_evasion_metrics`, reflecting the behavior of the navigation system in the presence of risks.

An example of the data structure stored for a typical drone can be found in Appendix A.1.

`DroneDatabase` acts as a black box for the simulation, continuously and systematically documenting the most relevant parameters of each mission. This local database serves as input for metric generation, failure identification, strategy evaluation, and, most importantly, the development of the machine learning model, aimed at predicting simulation success.

The data collection process is carried out automatically through the module's subscription to specific broker topics. The `on_message` method, which acts as the client's message handler, is responsible for interpreting the topics and forwarding the extracted data to the appropriate database update functions. Information is automatically extracted based on the topic type and stored in the database in a structured manner.

To ensure data integrity and persistence, the module incorporates an autosave mechanism, implemented by the `_start_auto_save` routine, which periodically writes the con-

tent to disk (every 2 seconds). This guarantees data consistency even during long-running simulations or in the event of unexpected interruptions.

3.4 Metrics for Evaluating Simulated Missions

To enable quantitative analysis of the missions simulated by the drones, specific metrics were defined and implemented to capture aspects of efficiency and operational safety. These metrics are extracted from the data transmitted over the bus and reflect the drones' dynamic behavior over time.

FIGURE 3.6 – Drone approaching the delivery point.

3.4.1 Total Time in Evasive Maneuvers

The `total_avoid_time` metric represents the sum, in seconds, of the time the drone spent performing evasive maneuvers, either to avoid collisions with other drones or to avoid flying over helipads. It is calculated as the sum of the durations listed in `collision_avoidance_times` and `helipoint_avoidance_times`. This metric reflects the impact of risk situations on the mission.

3.4.2 Number of Successful and Failed Evasions

Two distinct counts are used: `collisions_evasions`, which represents the number of successful evasive maneuvers (`collision_evasions` + `helipoint_evasions`), and `failed_collisions_evasions`, which counts the failed attempts (`failed_collision_evasions` + `failed_helipoint_evasions`). These metrics indicate the level of risk exposure and the effectiveness of the autonomous evasion system.

3.4.3 Average Distance and Variability Relative to the Destination

During the flight, the geodesic distance between each recorded coordinate and the destination point is calculated. From these measurements, the average distance (`avg_distance_to_dest`) and the standard deviation (`std_distance_to_dest`) are extracted, both expressed in meters. These values represent how close and stable the drone's trajectory was relative to its intended target.

3.4.4 Total Mission Time

The `mission_time` variable represents the elapsed time between the start and end of the mission (based on the timestamps of the recorded coordinates), converted to seconds. This metric includes both the trip to the destination and the return to base, when applicable.

3.4.5 Normalized Mission Time (Z-Score)

To standardize comparisons between missions, the total mission time is converted into a Z-Score based on the average and standard deviation of all mission durations in the simulation. This metric, stored as `mission_time_zscore`, helps identify outlier missions in terms of execution time and will be useful in the classification process.

3.4.6 Mission Status

Although not used as an input metric for the model, the status information is assigned to the training data as a label. This enables the application of a supervised learning model, trained on the previously described metrics, to classify new simulated missions. Each mission can be classified as a "success" (delivery completed and returned to the store without collision), "partial failure" (delivery completed, no return) or "failure" (mission not completed or involved in a collision).

3.5 Machine Learning Model

The supervised learning model developed in this work aims to classify the outcome of each simulated mission into one of the three categories presented in Section 3.4.6 (success, partial failure, or failure). To achieve this, the KNN algorithm was used, trained based on the extracted metrics, as described in Section 3.4.

The development process included the steps of data collection and organization, attribute preprocessing and instance labeling. The resulting structure served as the basis for training and evaluating the classifier, with the goal of providing reliable classifications of the missions.

3.5.1 Data Collection and Preparation

The data used for training the model were extracted directly from the generated simulation files. Each simulation contains JSON records with detailed information about the drones' trajectories, evasion events and mission stages. Based on these records, an analysis routine was developed to automatically iterate through all files in the simulation directory and extract, for each individual mission (identified by the drone-order pair), a set of attributes organized in dict-type structures.

For each mission, the following metrics relevant to the predictive model are recorded:

- Total mission time;
- Total time spent in evasion maneuvers;
- Total number of evasion maneuvers;
- Number of unsuccessful evasion maneuvers;
- Average distance to destination;
- Standard deviation of distance to destination;
- Normalized mission time (*Z-Score*).

The data are organized in a main dictionary, indexed by a unique key in the format `<simulation>|<mission>|<drone>|<order>`, which enables traceability and individualized analysis of each instance.

3.5.2 Mission Categorization

The labeling of each mission was performed automatically during the data extraction stage, based on the events recorded in the `stages` field of the simulation files. This field contains a list of the stages the mission went through, such as `entregue` (delivered), `retornou_loja` (returned to store), and `collided`.

The classification logic is defined as follows:

- **Success**, when the stages include `entregue` and `retornou_loja`, and do not include `collided`;
- **Partial**, when they include `entregue` but not `retornou_loja`, and also not `collided`;
- **Failure**, in all other cases, especially when a collision occurs or the delivery fails.

These categories were automatically assigned to each mission during preprocessing and constitute the target output variable used in training the classifier.

3.5.3 Exploratory Data Analysis

The Exploratory Data Analysis (EDA) stage aimed to understand the behavior of the extracted variables, investigate relevant relationships and drive the selection of features for the model.

Figure 3.7 shows the distribution of mission times. A high concentration of low values can be observed, followed by a long right tail, which is typical of skewed distributions. This indicates the presence of very short missions (usually associated with failures) and greater variation in mission elapsed time among successful missions.

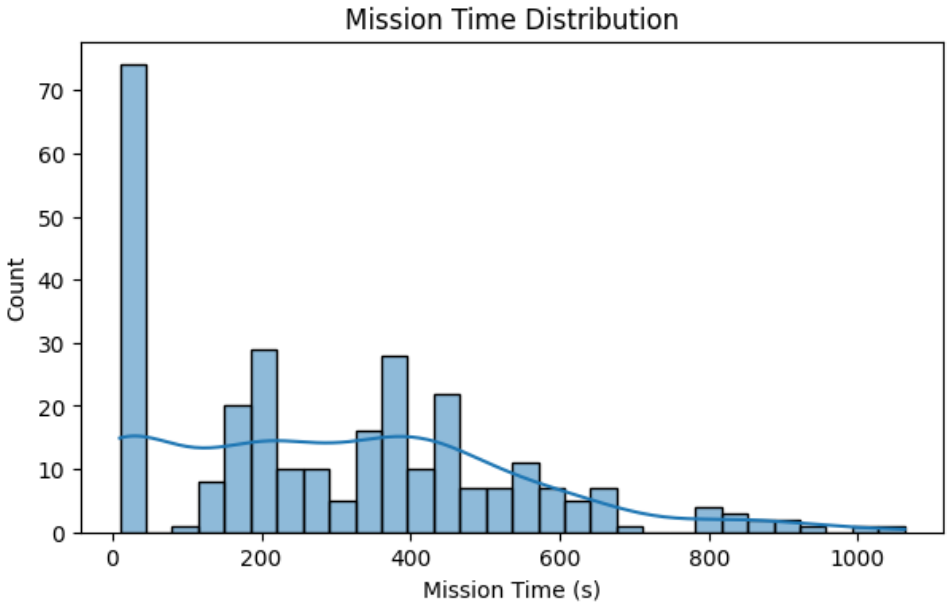

FIGURE 3.7 – Mission time distribution.

Figure 3.8 shows the class imbalance, with a predominance of successful missions and a significantly smaller quantity of partial missions. This asymmetry should be taken into account during the model's validation and evaluation stages, as it may affect performance on minority classes.

FIGURE 3.8 – Mission categories distribution.

The analysis of mission time by category (Figure 3.9) confirms that failed missions tend to have very short durations. Moreover, the greater dispersion in successful cases suggests higher variability in expected behavior.

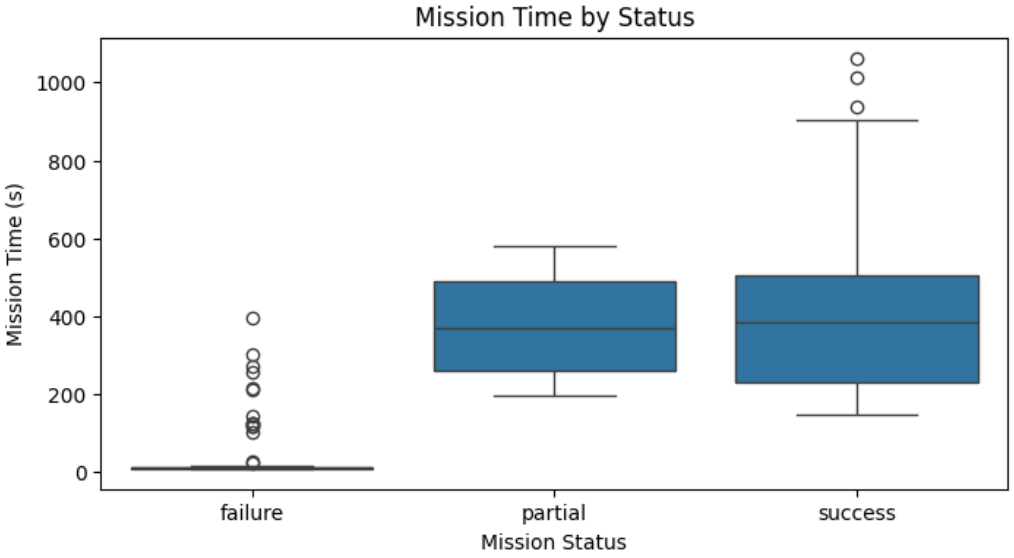

FIGURE 3.9 – Mission time by conclusion category.

Figure 3.10 shows the heatmap of correlations between numerical variables. A strong correlation was observed between `mission_time` and other variables such as `std_distance_to_dest` ($r = 0.92$) and `mission_time_zscore` ($r = 0.80$), indicating redundancy. For this reason, the `mission_time` variable was removed from the feature set, retaining those that provide more specific or normalized information.

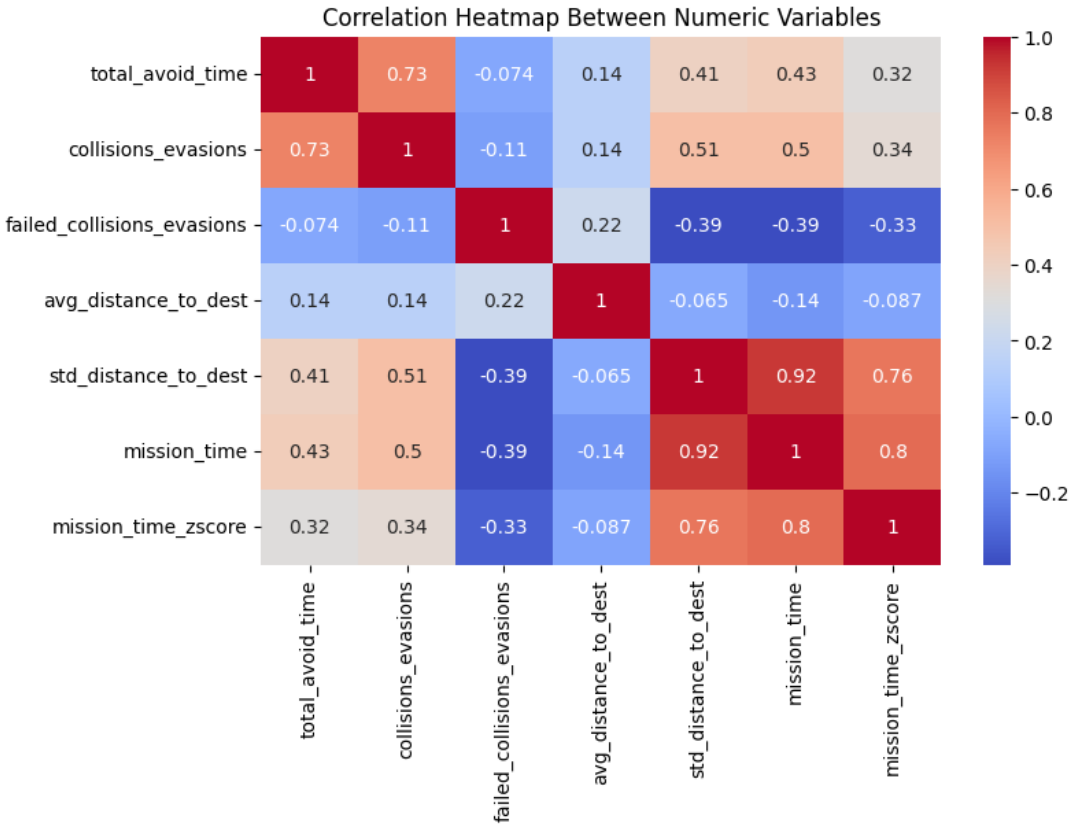

FIGURE 3.10 – Heatmap of correlations between numerical variables.

The relationship between the total number of evasions and mission time was also explored (Figure 3.11). Although most points are concentrated in regions with few evasions, some clusters of successful missions with a high number of evasions were identified, indicating distinct operational patterns.

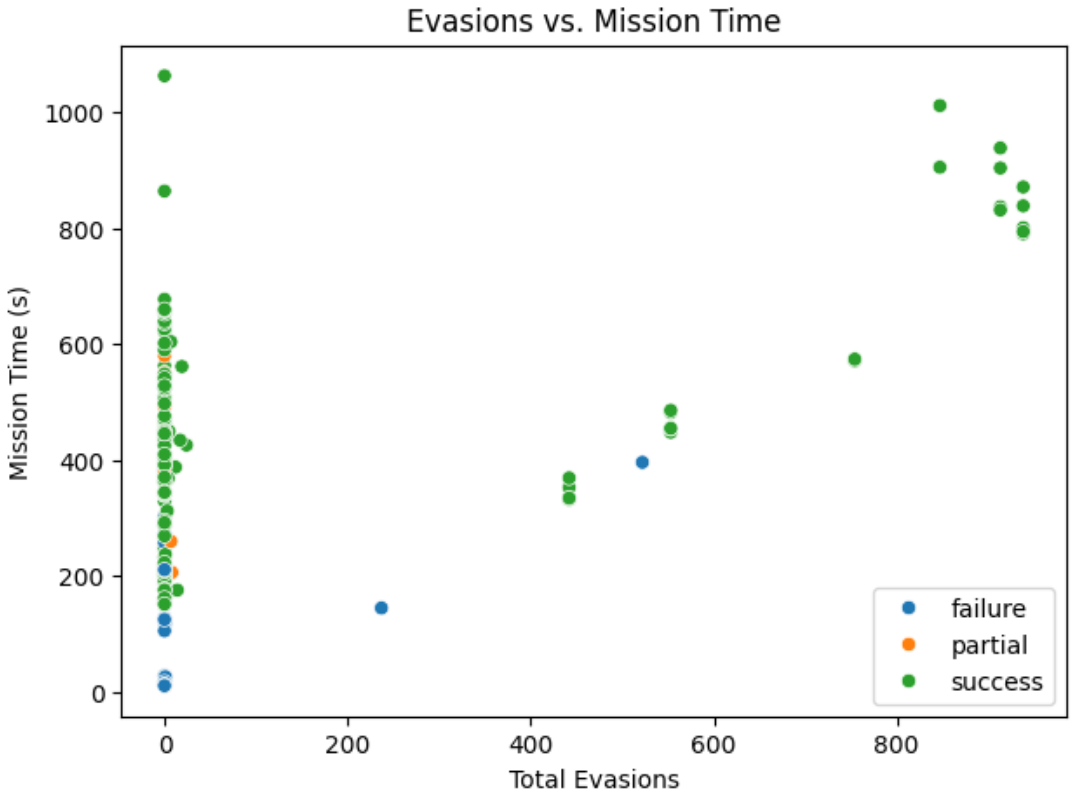

FIGURE 3.11 – Relationship between total quantity of evasions and mission time.

Finally, a closer inspection of missions classified as partial was carried out to better understand their operational characteristics. It was observed that this type of mission represents a small fraction of the total dataset, reinforcing the class imbalance previously discussed. Nevertheless, they were retained in the training set to allow the model to recognize intermediate patterns between success and failure.

Tables 3.3, 3.4 and 3.5 present a sample of these missions.

TABLE 3.3 – Evasion metrics of partial missions

Mission	Evasion Time (s)	Evasions	Failures
10_orders|drone_database2|drone5.1|6907	0.3083	8	3628
10_orders|drone_database2|drone7.1|2949	0.0000	0	3693
10_orders|drone_database4|drone6.1|9208	0.0000	0	2470
10_orders|drone_database4|drone1.1|8906	0.0747	1	3647
10_orders|drone_database5|drone4.1|3881	0.0000	0	10536
20_orders|drone_database1|drone3.1|4542	0.0000	0	10919
4_orders|drone_database12|drone3.1|4056	0.0000	0	0
4_orders|drone_database8|drone1.1|4818	0.0000	0	0
7_orders|drone_database3|drone6.1|3022	0.0581	7	359

TABLE 3.4 – Distance-to-destination metrics of partial missions

Mission	Mean (m)	Standard Deviation (m)
10_orders|drone_database2|drone5.1|6907	395.05	277.73
10_orders|drone_database2|drone7.1|2949	377.66	267.65
10_orders|drone_database4|drone6.1|9208	1098.97	692.55
10_orders|drone_database4|drone1.1|8906	1269.13	848.93
10_orders|drone_database5|drone4.1|3881	814.52	526.47
20_orders|drone_database1|drone3.1|4542	1234.76	835.12
4_orders|drone_database12|drone3.1|4056	1420.15	879.05
4_orders|drone_database8|drone1.1|4818	1082.42	818.70
7_orders|drone_database3|drone6.1|3022	560.98	376.94

TABLE 3.5 – Time and Z-Score of partial missions

Mission	Time (s)	Z-Score
10_orders|drone_database2|drone5.1|6907	205.31	0.32
10_orders|drone_database2|drone7.1|2949	196.84	0.27
10_orders|drone_database4|drone6.1|9208	490.75	1.75
10_orders|drone_database4|drone1.1|8906	305.78	0.65
10_orders|drone_database5|drone4.1|3881	370.14	0.67
20_orders|drone_database1|drone3.1|4542	550.18	0.80
4_orders|drone_database12|drone3.1|4056	579.65	1.14
4_orders|drone_database8|drone1.1|4818	377.60	1.42
7_orders|drone_database3|drone6.1|3022	259.10	0.17

3.5.4 Model Training and Validation

For the task of supervised classification of missions, the algorithm was configured with weights assigned inversely proportional to the distance of the nearest neighbors. The instances were initially split into training (80%) and test (20%) sets, preserving the original class proportions via stratification. Then, the model was trained and evaluated using stratified 5-fold cross-validation to obtain more robust and generalizable performance estimates.

The hyperparameters chosen for the classifier included using the Minkowski distance metric with exponent $p = 2$, equivalent to Euclidean distance, and assigning weights inversely proportional to distance, in order to ensure greater importance to closer neighbors during classification.

The learning curve shown in Figure 3.12 evaluates the model's average performance with different values of k, ranging from 1 to 14, using 5-fold cross-validation. A high and relatively stable average accuracy is observed between $k = 1$ and $k = 5$, with a slight decline from $k = 6$ onward, indicating a loss of discriminative power as more neighbors are considered. Based on this behavior, the value $k = 5$ was chosen as the final parameter, as it represents a balance between performance and robustness, avoiding both overfitting and loss of precision.

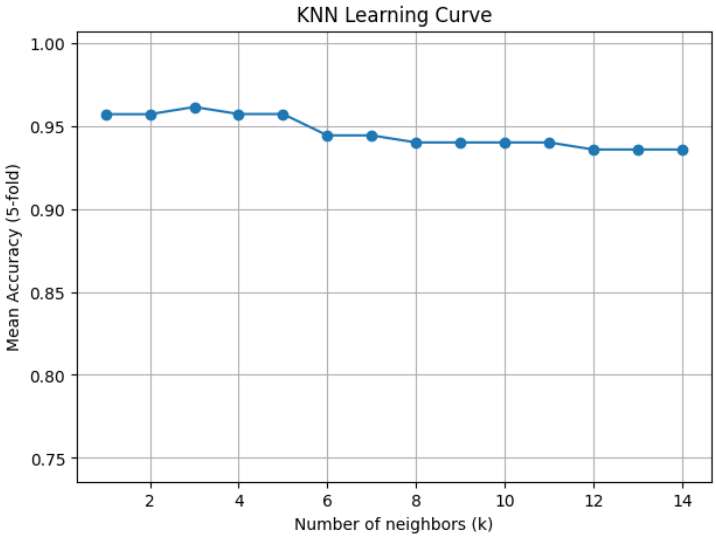

FIGURE 3.12 – Learning curve of the KNN model varying the number of neighbors (k).

The final model was evaluated using stratified 5-fold cross-validation, resulting in an average accuracy of 95.7%. Figure 3.13 presents the confusion matrix obtained during validation, while Tables 3.6 and 3.7 show the performance scores per class and the individual accuracies for each fold, respectively.

The results indicate high performance for the "success" and "failure" classes, with

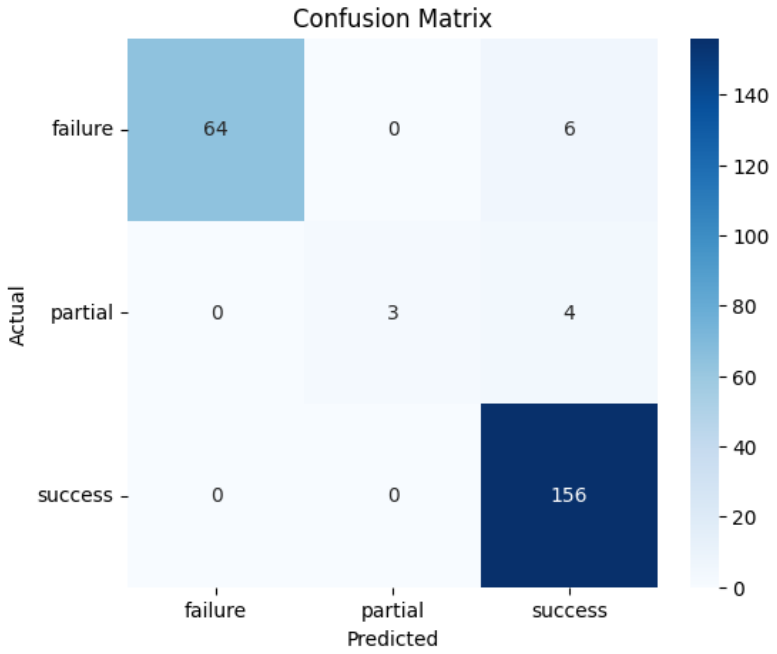

FIGURE 3.13 – Confusion matrix of the KNN classifier (5-fold cross-validation).

precision, recall, and F1-Score values above 0.94. However, the "partial" class showed a lower recall (0.43), indicating that some of these instances were incorrectly assigned to the other categories. This limitation is likely related to the small number of examples from this class in the training set, making it harder for the model to learn accurately.

TABLE 3.6 – Performance scores by class

Class	**Precision**	**Recall**	**F1-Score**	**Support**
Failure	1.00	0.91	0.96	70
Partial	1.00	0.43	0.60	7
Success	0.94	1.00	0.97	156

TABLE 3.7 – Accuracy achieved in each partition (fold)

Fold	**Accuracy**
Fold 1	0.894
Fold 2	0.979
Fold 3	1.000
Fold 4	0.935
Fold 5	0.978
Mean	**0.957**

3.6 Metric Visualization

A dashboard was developed to make the analysis of simulation data more accessible and to facilitate result interpretation by stakeholders. The interface allows users to upload files containing simulation logs, select the desired drone and mission, and view the information in a clear and interactive format.

The dashboard is divided into two main tabs:

- **Simulation Data**: displays graphs of the drone's trajectory over time (latitude, longitude and altitude), with contextual information such as the final destination, mission status, executed stages and a summary of evasive maneuvers. This visualization allows users to closely follow mission behavior and identify relevant situations such as deviations, failures or interruptions.
- **Mission Evaluation**: presents the metrics calculated from the simulation data, as described in Section 3.4, and shows the classification result generated by the machine learning model. This evaluation provides a standardized indication of mission performance, categorizing it as a success, partial failure or failure.

The system automatically processes the data after the file is uploaded, dynamically updating all elements of the dashboard. This functionality enables fast and comparative analysis across different simulations, making it easier to identify patterns and supporting decision-making throughout the mission design and evaluation process. The developed interface will be presented in Section 4.3, where its visual and functional aspects will be discussed.

4 Discussions

4.1 Logger Implementation and Metric Definition

4.1.1 System Adaptations

During the development of the monitoring system, it became necessary to adapt the pre-existing simulation to enable systematic data collection and detailed analysis of operational data. The original architecture, although functional for demonstrative purposes, had not been designed with a focus on traceability, observability or performance evaluation.

Among the most significant modifications is the introduction of specific metrics to monitor evasion behaviors, both regarding potential collisions between drones and the unintentional entry into defined exclusion zones, such as helipad areas. These indicators allowed for the quantification of the navigation algorithms' effectiveness when facing dynamic obstacles and geographical constraints.

Additionally, a failure induction mechanism was inserted, based on the creation of simulated entities with intentionally conflicting behavior, denominated "suicide drones". These agents, with routes programmed for the deliberate crossing of trajectories with other active drones, were used as a resource to generate risk scenarios and validate the response capability of the avoidance system. The inclusion of these entities, although artificial, proved effective in provoking edge-case situations and collecting data representative of operational failures.

Finally, the message structure published via MQTT was expanded to include new topics and informative fields, covering not only the kinematic state of the drones but also mission events (such as delivery stages), updates on the life cycle of requests, and critical operational indicators, such as evasion attempts and collision occurrences. This expansion of the informational mesh enabled the obtaining of more granular indicators, consistent with the event timeline, and proved fundamental for the subsequent automated classification of simulation results.

4.1.2 Logger Operation and Data Storage

The logger was developed as an external module to the main simulation, integrated directly into SIRESANT project. Its main function was to monitor the relevant MQTT topics generated during simulation execution and log the data in real-time in the JSON format. The storage structure adopted was designed to preserve the temporal sequence of events, ensuring compatibility with trajectory reconstruction processes and subsequent analyses.

In addition to kinematic and heartbeat messages, the logger captured publications related to the evolution of requests (including different mission stages), evasion events, and collision detection. Fields for signaling critical operational situations, such as delivery failures or exclusion zone violations, were also included.

The modular structure of the logger, combined with the systematic use of timestamps, allowed for the reconstruction of the simulations' timeline. This approach enabled data collection in a non-intrusive manner and with sufficient granularity to support the metric analysis and predictive models applied in later phases of the project.

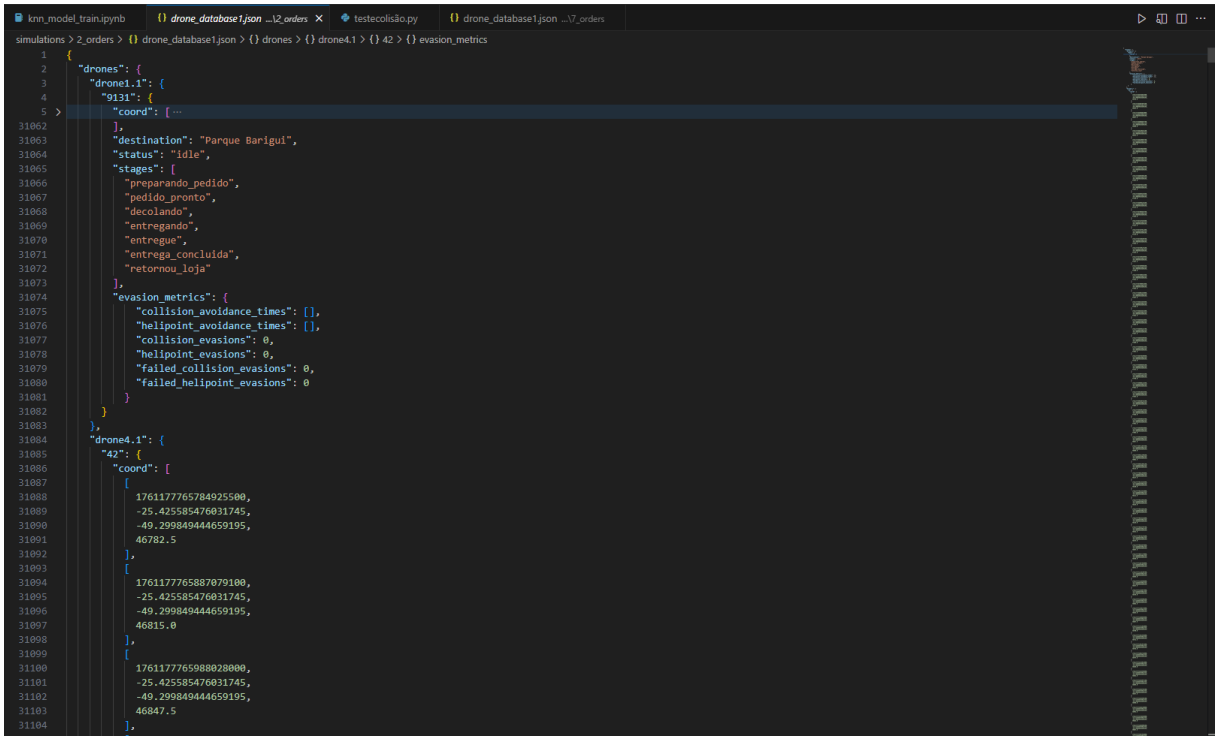

FIGURE 4.1 – Snippet of the drone database file, containing evasion metrics and coordinate logs over time.

4.1.3 Derived Metrics and Findings

With the persisted data, it was possible to derive specific metrics related to drone behavior in risk situations. These metrics included, for example, the time required to complete collision avoidance maneuvers (`collision_avoidance_times`) and helipad avoidance maneuvers (`helipoint_avoidance_times`), in addition to the count of successful or failed evasions in both contexts. Data was logged individually per order, which allowed for analysis segmented by drone, obstacle type and mission context.

The analysis of these indicators revealed, for example, that the frequency and duration of evasive maneuvers increase significantly when multiple drones depart simultaneously from the same store. In these cases, the spatial and temporal overlap of trajectories during the initial delivery phases, especially during takeoff and the first moments of flight, creates zones of higher operational density, raising the risk of collision and, consequently, the need for deviations. This type of metric proved fundamental not only for evaluating the simulation's performance but also for feeding the predictive model, for effectively classifying the outcomes into the defined categories, based on the emergent patterns observed.

4.1.4 Challenges and Limitations Faced

One of the main challenges faced was the instability of the internal network where the MQTT broker was hosted. On certain days, the connection became especially unstable, leading to frequent drops during simulation executions. Consequently, the logging of several published messages was lost, compromising the integrity and completeness of some collected data, especially in scenarios with a high publication rate. This instability also hindered the execution of a larger number of simulations, since, in many cases, the connection made continuous and reliable experiments infeasible.

Furthermore, reading the original simulation source code, especially the `testecolisão.py` file, required additional attention for implementing changes related to instrumentation and data collection. The strongly coupled structure between classes and the logic distributed across multiple *threads* demanded careful analysis to prevent undesirable side effects. The insertion of evasion metrics and the adaptation of MQTT topics to report real-time events were especially challenging, requiring several iterations of testing, fine-tuning, and manual validation of the execution flows.

4.2 Supervised Learning Model - KNN

The analysis of the results obtained with the machine learning model revealed significant contributions to the task of classifying the success of the conducted simulations. The

results were evaluated using two main instruments: the confusion matrix (Figure 4.2) and the classification report (Table 4.1).

The confusion matrix indicates that the model was effective in accurately classifying the majority of instances in the "success" and "failure" classes. However, the "partial" class showed inferior performance, which can be attributed to the class imbalance in the dataset, due to the low frequency of simulations with this status (uncommon occurrence).

Table 4.1 presents the metrics extracted from the classification report, including precision, recall and f1-score. The "success" class achieved the best indicators, with values greater than 0.90. The "failure" class demonstrated equally high performance, with precision and F1-Score values greater than 0.90 and a recall of 0.82. The "partial" class had the lowest values, indicating the model's difficulty in learning representative patterns for this category.

TABLE 4.1 – Classification report with metrics by class

Class	Precision	Recall	F1-Score
Success	0.91	1.00	0.95
Partial	1.00	0.50	0.67
Failure	1.00	0.82	0.90

This limitation is noteworthy, considering that simulations classified as "partial" represent intermediate behaviors which, although not completely failed, may contain relevant deviations to the stakeholders. Thus, expanding the dataset with more occurrences of this class is necessary to increase the model's robustness and generalization.

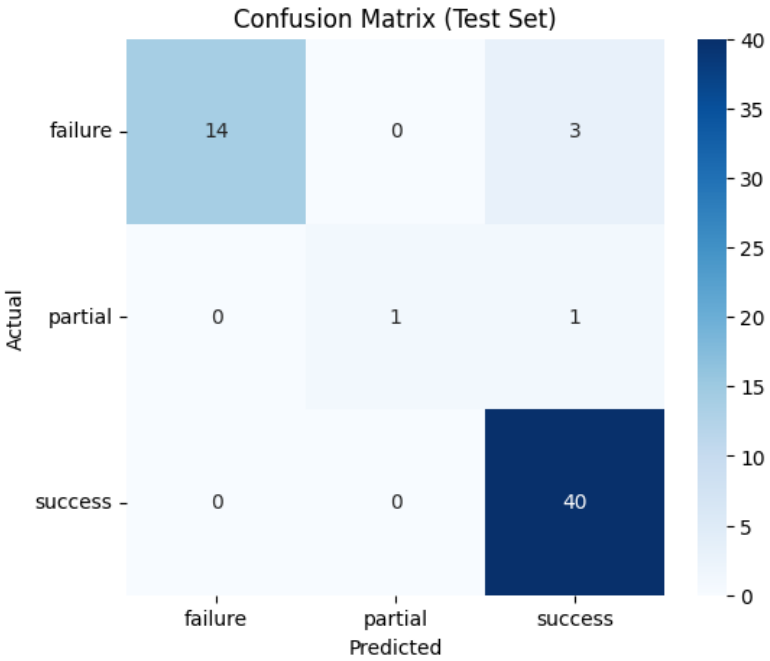

FIGURE 4.2 – Confusion matrix generated from test data.

Nevertheless, the obtained results allow for the use of the model as a preliminary decision-support tool. Its integration into the developed plugin offers an automated and continuous way of evaluating simulations, contributing to the identification of unexpected behaviors during the execution of the simulated scenarios.

4.3 Dashboard Visualization

The implementation of the interface for visualizing the metrics on the dashboard was completed without relevant technical issues. The development occurred seamlessly with the plugin structure, using the data automatically logged by the MQTT Logger, especially those organized by the `DroneDatabase` class, which is responsible for structuring events and information related to the simulated entities.

The metrics displayed encompass operational, behavioral and performance aspects of the entities during simulations. This data is extracted from files in `.json` format generated at the end of each execution and presented in a clear and intuitive manner to facilitate interpretation by stakeholders.

The structured visualization of these data makes it possible to recognize behavioral patterns, evaluate delivery efficiency, detect evasive maneuvers and collisions, and analyze the progression of the simulations over time.

As an example, a file generated at the end of a simulation composed of six delivery orders was used, including the intentional launch of a drone to provoke a collision. This file contains the consolidated data from the `DroneDatabase` class routines, which is responsible for the structured logging of events from the simulated entities. Based on this information, the dashboard presents the calculated metrics for each entity in an organized manner. The following figures illustrate this interface and the arrangement of information in the final visualization.

As an example, a file generated at the end of a simulation involving six delivery orders was used, which also included the intentional launch of a drone in order to provoke a collision. Based on these data, the dashboard displays, in an organized manner, the metrics calculated for each entity. The following figures illustrate this interface and the arrangement of information in the final visualization.

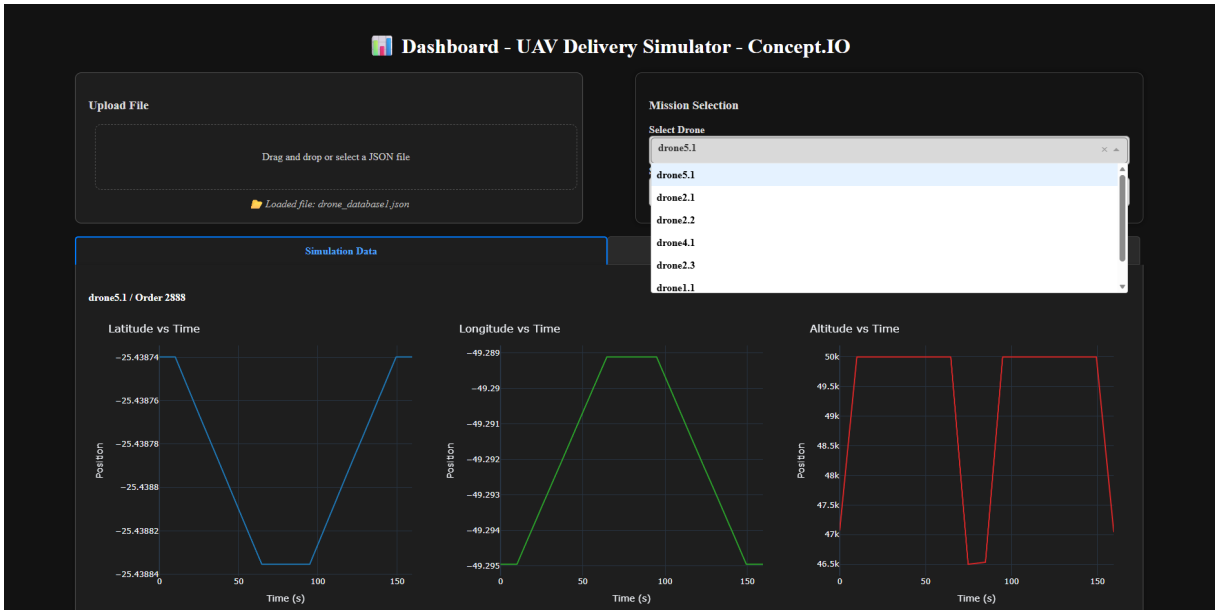

FIGURE 4.3 – Initial interface of the dashboard with the selection of a specific drone for analysis.

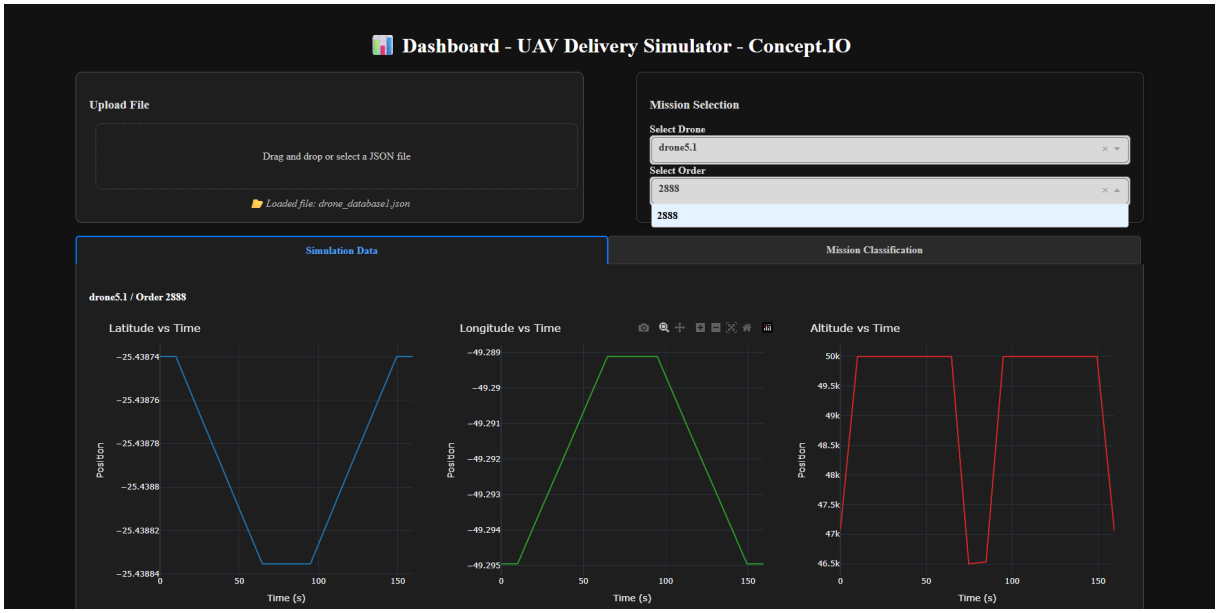

FIGURE 4.4 – Selection of a specific order associated to the chosen drone.

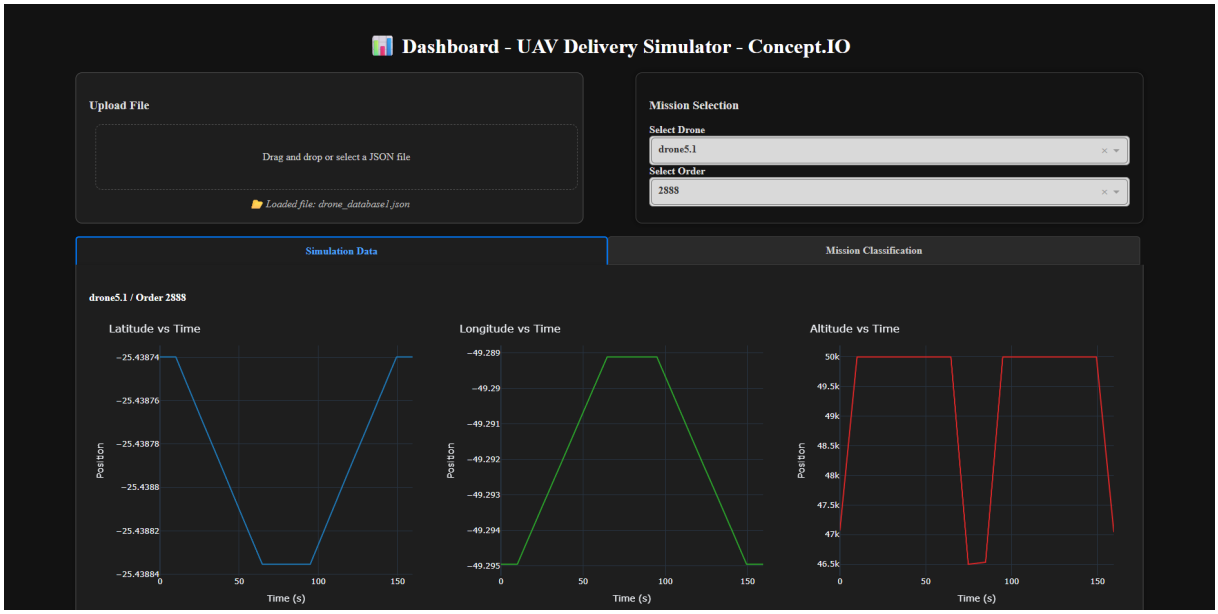

FIGURE 4.5 – Plots of latitude, longitude and altitude as a function of time for the selected mission.

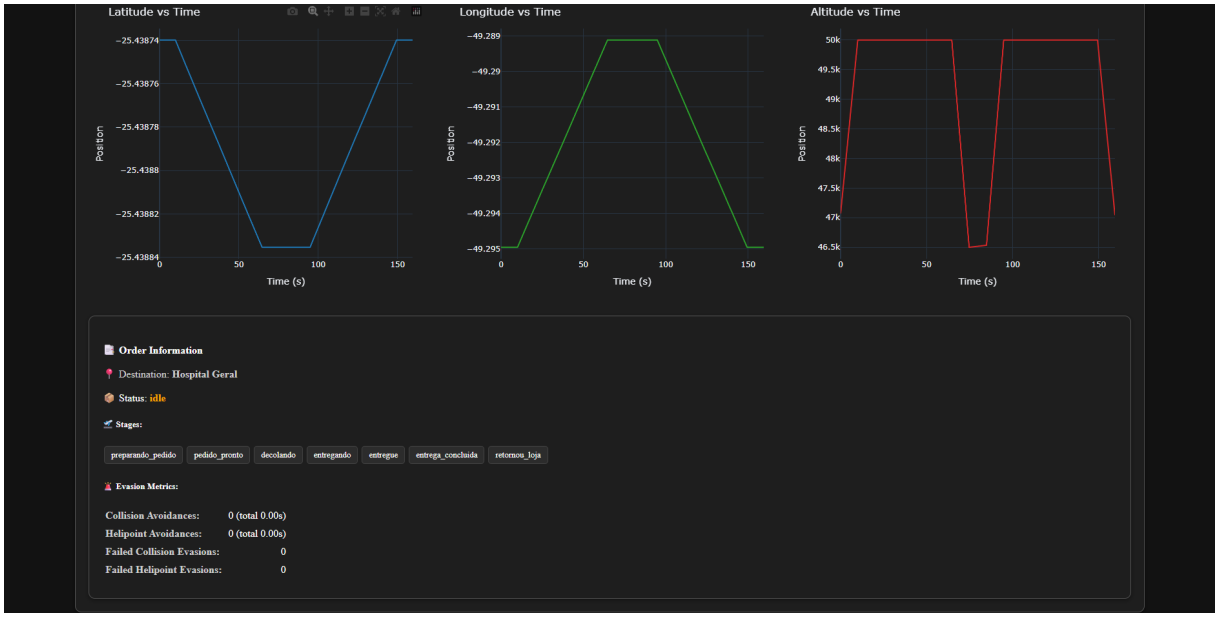

FIGURE 4.6 – Display of additional mission information, including destination, status, stages, and evasion metrics.

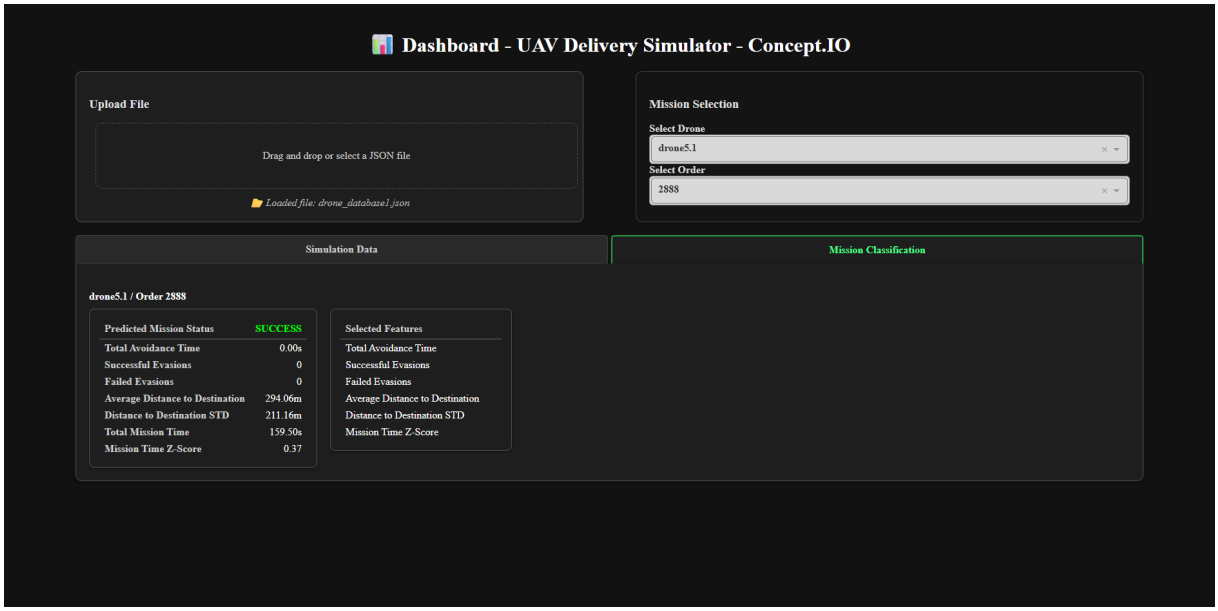

FIGURE 4.7 – Final classification of the mission based on the extracted metrics.

5 Conclusion

The development of the proposed plugin for the Arena Concept.IO platform demonstrated the feasibility of integrating, within a single environment, simulation monitoring resources, quantitative metric analysis and machine learning. This integration allowed the data produced during executions to be transformed into structured and interpretable information, capable of supporting the evaluation of mission performance and the identification of emergent behavior patterns. In addition to strengthening the traceability and observability of the simulated systems, it contributed to the creation of a solid analytical foundation, focused on requirements verification and decision-making in complex Systems Engineering contexts.

From a technical perspective, the implementation of the *logger* module with data storage in JSON files enabled the structured and persistent logging of messages transmitted via the MQTT bus, ensuring the temporal and contextual integrity of the data. This architecture, combined with the interactive visualization offered by the developed dashboard, facilitates the interpretation of simulations, allowing stakeholders and engineers to monitor mission progress. The combined use of tools such as VR-Forces, responsible for executing the simulated scenarios, and Scikit-learn, applied to machine learning analysis, reinforced the integration between the simulation and data processing stages, consolidating the developed ecosystem as a coherent and functional solution.

In relation to the analysis methodology, the developed metrics proved effective in characterizing critical dimensions of mission performance and efficiency. Indicators such as total mission time, average distance to the destination, duration of evasion maneuvers, and the number of registered failures allowed for the quantification of drone behavior with some accuracy and the correlation of operational patterns with the results obtained. This structure was essential not only for the descriptive evaluation of the data but also for feeding the supervised machine learning model.

The application of the KNN algorithm to the set of extracted metrics confirmed the model's ability to distinguish, based on observable parameters, the different mission outcomes, classifying them as *success*, *partial failure*, or *failure*. Even when faced with a limited and partially unbalanced dataset, the model demonstrated satisfactory performance

and behavior consistent with the trends observed during exploratory analysis. This step demonstrated the potential for using supervised learning techniques as an instrument for decision support and automated validation of emergent properties in simulated systems.

The achieved results demonstrate that the proposal fulfills its main objective: to provide an integrated framework for collecting, analyzing, and visualizing simulation data, capable of supporting requirements verification and effectiveness evaluation of complex systems. Furthermore, the modifications made to the original simulation, including collision detection and the introduction of drones with aggressive behavior, brought the scenarios closer to a real context, enabling the controlled study of failures and a more representative experimental analysis.

However, some limitations were identified. In certain executions, instabilities in the connection with the MQTT broker caused interruptions in message transmission, reducing the number of valid simulations and compromising the completeness of part of the collected data. In this sense, expanding the dataset and including new metrics represent important steps to enhance the model's generalization capacity and enable the use of more sophisticated algorithms.

As suggestions for future work, we first highlight the conducting of new experiments with the aim of expanding the database and, consequently, increasing the robustness and reliability of the machine learning model. This expansion will allow for a more consistent analysis and the identification of more complex behavior patterns.

Another relevant proposal is the implementation of the logger in real-time, such that the control and monitoring of the simulation can be integrated directly into the dashboard, providing instant tracking of mission progress. This functionality can be complemented by modeling a dedicated database for storing simulation logs and metrics, allowing for both historical queries and continuous analysis of the collected data.

Furthermore, it is recommended to expand the elements represented in the simulation scenarios incorporating new types of obstacles, such as pedestrians and ground vehicles, as well as other entities in simultaneous operation. This expansion would help bring the simulated environment closer to reality and allow for the study of more complex interactions among heterogeneous systems. Extending the plugin's functionality for other application domains, such as maritime operations or multimodal logistics, is also suggested, broadening its scope and demonstrating its versatility in different operational contexts.

In summary, this work contributes to the advancement of using analytical and computational methods in the evaluation of simulated systems, demonstrating that the combination of well-defined metrics, structured data collection, and machine learning constitutes a promising approach for the analysis and validation of complex systems within the context of Arena Concept.IO.

References

ACADEMY, A. **Trilha Dashboards Interativos com Python**. [*S.l.: s.n.*], 2025. Accessed: 20 May 2025. Available from: https://asimov.academy/dashboards-interativos-com-python/. Cit. on p. 30.

BROWN, T. J.; CONRAD, S. H.; BEYELER, W. E.; GLASS, R. J. Complex adaptive systems engineering and risk reduction. en. **Proceedings of the Institution of Civil Engineers - Engineering Sustainability**, Thomas Telford Ltd., v. 166, n. 5, p. 293–300, Oct. 2013. DOI: 10.1680/ensu.12.00036. Available from: http://dx.doi.org/10.1680/ensu.12.00036. Cit. on p. 15.

CERQUEIRA, C. S. **Arena Concept.IO**. [*S.l.: s.n.*], 2025a. Accessed: 23 May 2025. Available from: https://www.conceptio.ita.br/arena-concept-io/. Cit. on pp. 27, 28.

CERQUEIRA, C. S. **SIMUA-VD: Workshop CET-ADS**. [*S.l.: s.n.*], 2025b. Presentation at workshop. Held on 2 Apt 2025. Cit. on p. 19.

CHO, J.-H.; HURLEY, P. M.; XU, S. Metrics and measurement of trustworthy systems. *In:* MILCOM 2016 - 2016 IEEE Military Communications Conference. [*S.l.*]: IEEE, Nov. 2016. p. 1237–1242. DOI: 10.1109/milcom.2016.7795500. Available from: http://dx.doi.org/10.1109/milcom.2016.7795500. Cit. on pp. 15, 16.

COMPONATION, P. J.; DORNEICH, M.; HANSEN, J. L. Comparing Systems Engineering and Project Success in Commercial-focused versus Government-focused Projects. en. **Procedia Computer Science**, Elsevier BV, v. 44, p. 266–274, 2015. DOI: 10.1016/j.procs.2015.03.006. Available from: http://dx.doi.org/10.1016/j.procs.2015.03.006. Cit. on p. 14.

ENGINEERING, R. S. **CONOPS in Action: Real-World Examples and Best Practices**. [*S.l.: s.n.*], 9 fev. 2025. Accessed: 24 May 2025. Available from: https://reqi.io/articles/conops-in-action-best-practices. Cit. on p. 18.

FET, A.; HASKINS, C. Systems Engineering. *In:* [*s.l.: s.n.*], Feb. 2023. p. 127–136. ISBN 978-3-031-22244-3. DOI: 10.1007/978-3-031-22245-0_12. Cit. on pp. 16, 18, 21, 22.

GIANNI, D.; D'AMBROGIO, A.; TOLK, A. **Modeling and Simulation-Based Systems Engineering Handbook**. [*S.l.*]: Taylor & Francis, 2014. (Engineering Management). ISBN 9781466571457. Available from: https://books.google.com.br/books?id=EoiZBQAAQBAJ. Cit. on p. 19.

HABBEMA, L. **Introdução ao Scikit-learn**. Medium. Accessed: 20 Oct 2025. 2024. Available from: https://medium.com/@habbema/introdu%C3%A7%C3%A3o-ao-scikit-learn-f00b7201dbf7. Cit. on p. 30.

HENZINGER, T. A. Quantitative reactive modeling and verification. en. **Computer Science - Research and Development**, Springer Science and Business Media LLC, v. 28, n. 4, p. 331–344, Oct. 2013. DOI: 10.1007/s00450-013-0251-7. Available from: http://dx.doi.org/10.1007/s00450-013-0251-7. Cit. on p. 14.

IBM. **What is supervised learning?** Accessed: 20 Oct 2025. 2025a. Available from: https://www.ibm.com/think/topics/supervised-learning. Cit. on p. 23.

IBM. **What is the k-nearest neighbors (KNN) algorithm?** Accessed: 20 Oct 2025. 2025b. Available from: https://www.ibm.com/br-pt/think/topics/knn. Cit. on pp. 23, 24.

INCOSE. **INCOSE Systems Engineering Handbook: A Guide for System Life Cycle Processes and Activities**. [*S.l.*]: Wiley, 2015. ISBN 9781118999417. Available from: https://books.google.com.br/books?id=ViDyCQAAQBAJ. Cit. on pp. 14, 15, 17, 21, 22.

KASHYAP, P. **Understanding Precision, Recall, and F1 Score Metrics**. 2024. Available from: https://medium.com/@piyushkashyap045/understanding-precision-recall-and-f1-score-metrics-ea219b908093. Visited on: 10 Nov. 2025. Cit. on p. 23.

MADNI, A. M.; SIEVERS, M. Model-Based Systems Engineering: Motivation, Current Status, and Needed Advances. *In:* DISCIPLINARY Convergence in Systems Engineering Research. [*S.l.*]: Springer International Publishing, Nov. 2017. p. 311–325. DOI: 10.1007/978-3-319-62217-0_22. Available from: http://dx.doi.org/10.1007/978-3-319-62217-0_22. Cit. on p. 14.

MADNI, A. M.; SIEVERS, M. Model-based systems engineering: Motivation, current status, and research opportunities. **Systems Engineering**, v. 21, n. 3, p. 172–190, 2018. DOI: https://doi.org/10.1002/sys.21438. eprint: https://incose.onlinelibrary.wiley.com/doi/pdf/10.1002/sys.21438. Available from: https://incose.onlinelibrary.wiley.com/doi/abs/10.1002/sys.21438. Cit. on p. 15.

MAK TECHNOLOGIES. **VR-Forces**: The MAK ONE Multi-Domain Computer Generated Forces. [*S.l.: s.n.*], 2025. Accessed: 19 May 2025. Available from: https://www.mak.com/mak-one/apps/vr-forces. Cit. on pp. 28, 29.

MONGODB INC. **MongoDB**. [*S.l.: s.n.*], 2025. Accessed: 24 May 2025. Available from: `https://www.mongodb.com`. Cit. on p. 26.

MQTT.ORG. **MQTT**. [*S.l.: s.n.*], 2025. Accessed: 23 May 2025. Available from: `https://mqtt.org/`. Cit. on p. 28.

PLOTLY TECHNOLOGIES INC. **Dash**. [*S.l.: s.n.*], 2025. Accessed: 23 May 2025. Available from: `https://dash.plotly.com/`. Cit. on p. 30.

PRESS, C. U. **Cambridge Dictionary**. [*S.l.: s.n.*]. Website. Available from: `https://dictionary.cambridge.org/`. Visited on: 20 May 2025. Cit. on p. 21.

SCIKIT-LEARN DEVELOPERS. **Nearest Neighbors — scikit-learn 1.4.2 documentation**. [*S.l.: s.n.*], 2024. Accessed: 20 Oct 2025. Available from: `https://scikit-learn.org/stable/modules/neighbors.html`. Cit. on p. 30.

VANGHELUWE, H. **Modelling and Simulation Concepts**: Material didático da disciplina CS522, Fall Term 2001. School of Computer Science, McGill University. [*S.l.: s.n.*], 2001. Accessed: 10 May 2025. Available from: `https://www.cs.mcgill.ca/~hv/classes/MS/MSconcepts.pdf`. Cit. on pp. 19, 20.

VILLAS, F.; KNÖÖS FRANZÉN, L.; JOUANNET, C.; AMADORI, K.; STAACK, I. Concept of Operations in an Agent-Based Simulation: A System-Of-Systems Approach. *In:* PROCEEDINGS of the 34th Congress of the International Council of the Aeronautical Sciences, Florence, Italy. [*S.l.: s.n.*], 2024. p. 9–13. Cit. on p. 20.

Appendix A - Storage in the MQTT Logger

A.1 Mission Data Storage Structure


```
{
  "drone1.1": {
    "order_12345": {
      "coord": [
        [
          63769820611800,
          -25.438739887054627,
          -49.29495369539843,
          47079.5
        ],
        [
          63769923581900,
          -25.438739887054627,
          -49.29495369539843,
          47050.0
        ]
      ],
      "destination": "Praca da Espanha",
      "status": "idle",
      "stages": [
        "preparando_pedido",
        "pedido_pronto",
        "decolando",
        "entregando",
        "entregue",
        "retornou_loja"
      ],
      "evasion_metrics": {
        "collision_avoidance_times": [
          0.42
        ],
```

```
        "helipoint_avoidance_times": [],
        "collision_evasions": 1,
        "helipoint_evasions": 0,
        "failed_collision_evasions": 0,
        "failed_helipoint_evasions": 0
      }
    }
  }
}
```

A.1 –

Example of saved content for a typical drone.

Each entry in the `coord` field presents the elements in the following order: `[timestamp, latitude, longitude, altitude]`.

FOLHA DE REGISTRO DO DOCUMENTO

1. CLASSIFICAÇÃO/TIPO	2. DATA	3. DOCUMENTO Nº	4. Nº DE PÁGINAS
TC	11 de novembro de 2025	DCTA/ITA/TC-050/2025	63

5. TÍTULO E SUBTÍTULO:

Plugin for Analysis and Monitoring of Simulation in the Arena Concept.IO

6. AUTOR(ES):

Marcus Gabriel de Almeida Nunes

7. INSTITUIÇÃO(ÕES)/ÓRGÃO(S) INTERNO(S)/DIVISÃO(ÕES):

Instituto Tecnológico de Aeronáutica – ITA

8. PALAVRAS-CHAVE SUGERIDAS PELO AUTOR:

Systems engineering; Complex systems; Simulation; Machine Learning; Metrics analysis; Drones

9. PALAVRAS-CHAVE RESULTANTES DE INDEXAÇÃO:

Veículos pilotados remotamente; Engenharia de sistemas; Sistemas complexos; Simulação computadorizada; Monitoramento; Aprendizagem (inteligência artificial); Computação; Engenharia aeronáutica.

10. APRESENTAÇÃO: (**X**) **Nacional** ( ) **Internacional**

ITA, São José dos Campos. Curso de Graduação em Engenharia Aeronáutica. Orientador: Maj. Av. Lucas Oliveira Barbacovi; coorientador: Prof. Dr. Christopher Shneider Cerqueira. Publicado em 2025.

11. RESUMO:

This work aims to develop a plugin for the Arena Concept.IO simulation platform, designed to integrate monitoring, metrics analysis, and machine learning within a Systems Engineering environment. The system was developed to collect and process data transmitted during simulations, recording them in structured files through a logging module, which allows for the assessment of mission performance and the identification of operational patterns based on predefined metrics. The collected information served as the basis for training a supervised model using the k-Nearest Neighbors algorithm, capable of classifying missions into success, partial failure or total failure categories. The results indicate that the model can recognize recurring behaviors and infer mission effectiveness, validating the use of machine learning as a support tool for the automated analysis of simulations. The plugin also incorporates an interactive dashboard that enables the visualization of simulated entities and their evolution over time. In light of these findings, the feasibility of integrating simulation and machine learning is confirmed, highlighting the potential of the proposed approach to support requirements verification and validation as well as the performance analysis of complex systems. The plugin can also be employed for failure studies and the optimization of operational strategies, serving as a foundation for future expansions in more complex and multidomain scenarios.

12. GRAU DE SIGILO:

(**X**) **OSTENSIVO** ( ) **RESERVADO** ( ) **SECRETO**