Development of a Multi-UAV Simulator to Analyze the Behavior of Operators in Coastal Surveillance Missions

Author: Víctor Rodríguez Fernández

Advisor: David Camacho Fernández

A thesis submitted in partial fulfillment for the Master Degree on Research and Innovation in Information and Communication Technologies

in the
Escuela Politécnica Superior
Departamento de Ingeniería Informática

April 2015

"Ambition is the path to success. Persistence is the vehicle you arrive in."

Bill Bradley

Abstract

Escuela Politécnica Superior
Departamento de Ingeniería Informática

Master on Research and Innovation in Information and Communication Technologies

by Víctor Rodríguez Fernández

This Master Thesis [1] presents the design and development of a computer simulator created for executing and supervising missions carried out by multiple Unmanned Aerial Vehicles (UAVs). The aim of this simulator is to provide an open, simple and accessible environment to train and analyze the performance and evolution of low-experienced human operators supervising and controlling a team of UAVs.

This work is divided into two parts. The first one is focused on describing the simulator mechanisms and architecture. To accomplish the required accessibility of this tool for novice users, the simulator has been implemented following a web architecture, where only a web browser is needed to execute it. Also, in order to engage and challenge the operator, some gamification elements have been added, bringing the simulation closer to a videogame experience.

The second part of this work uses the developed simulator to carry out several experiments with novice users. A set of performance metrics is designed to define the profile of a user, and based on those profiles, we run and validate some clustering algorithms to obtain groups of users with common performance profiles. These results are analyzed to extract behavioral patterns that distinguish and rank the different users in the experiment, allowing the identification and selection of potential expert operators.

Keywords

Unmanned Aerial Vehicles, Human-Robot Interaction, Computer-based Simulation, HTML5, Web, Videogames, Performance metrics, Clustering, Behavioral patterns

[1]This work has been funded by Airbus Defence & Space (SAVIER Project: FUAM-076914)

Resumen

Escuela Politécnica Superior
Departamento de Ingeniería Informática

Máster en Investigación e Innovación de las Tecnologías de la Información y las Comunicaciones

por Víctor Rodríguez Fernández

El presente Trabajo Fin de Máster [2] presenta el diseño y desarrollo de un simulador creado con el fin de ejecutar y supervisar misiones llevadas a cabo por múltiples Vehículos Aéreos no Tripulados (UAVs). El objetivo de este simulador es ofrecer un entorno simple y accesible donde entrenar y analizar el rendimiento y la evolución de operadores inexpertos mientras supervisan y controlan un equipo de UAVs.

Este trabajo se divide en dos partes. La primera está enfocada en describir el funcionamiento del simulador y su arquitectura. Para lograr la accesibilidad que esta herramienta requiere de cara a usuarios inexpertos, el simulador ha sido implementado siguiendo una arquitectura web, donde sólamente se requiere un navegador web para ejecutarlo. Además, para atraer y retar al operador, se han introducido algunos elementos de gamificación, que acercan este simulador a una experiencia propia del mundo de los videojuegos.

La segunda parte del trabajo se basa en el simulador desarrollado para llevar a cabo varios experimentos con usuarios inexpertos. Se ha diseñado un conjunto de métricas de rendimiento con las cuales se define el perfil de un usuario. Usando estos perfiles, se ejecutan y validan algoritmos de clustering para obtener grupos de usuarios con perfiles de rendimiento comunes. Los resultados se analizan de cara a extraer patrones de comportamiento que distingan a los diferentes usuarios del experimento, permitiendo la identificación y selección de operadores expertos potenciales.

Palabras Clave

Sistemas Aéreos no tripulados, Interacción humano-robot, Simulación por computadora, HTML5, Web, Métricas de rendimiento, Clustering, Patrones de comportamiento

[2]Este trabajo ha sido financiado por Airbus Defence & Space (Proyecto SAVIER: FUAM-076914)

Acknowledgements

First of all, I would like to acknowledge the financial support given by Airbus Defence & Space under the Savier project (FUAM-076914), as well as all the information and help provided from the Savier Open Innovation Project members: José Insenser, Juan Antonio Henríquez and Gemma Blasco.

Secondly, I would like to thank David Camacho for managing this Master Thesis and giving me the opportunity to work in this research group, and Antonio González Pardo, with whom I hope to develop a long and successful career in the next years.

I would like to express my gratitude to all my workmates from the AIDA group, both those who are working now and those who left us last year, specially to Fernando Palero. I am also indebted to all the people who have accompanied me throughout all these years in the university, specially to Daniel G.V and Diego, who has become my personal psychologist during the coffee time.

Special thanks to the three people who have been by my side day after day throughout the course of this Master Thesis: Gema Bello Orgaz, undoubtedly one of the best people I have had the pleasure of meeting this year, Héctor D. Menéndez, who has tutored the whole of this work, and to whom I owe most of what I have learned here, and Cristian Ramírez Atencia, who has become a brother to me after more than 6 years of continuous working, geeking and laughing together.

Finally, I would like to express my deepest gratitude to my family, specially to my mother for being so patient and kind with me, and to Reyes S.G, for giving me, even in the distance, the strength to accomplish all my goals and the smile to go through all of them.

Contents

List of Figures

List of Tables

Abbreviations

GCS	**G**round **C**ontrol **S**tation
HCI	**H**uman **C**omputer **I**nterface
UAS	**U**nmanned **A**ircraft **S**ystem
UAV	**U**nmanned **A**erial **V**ehicle
UMS	**U**nmanned **M**ission **S**imulator
UV	**U**nmanned **V**ehicle
US	**U**nmanned **S**ystem
MIT	**M**assachusetts **I**nstitute of **T**echnology
UUV	**U**nmanned **U**nderwater **V**ehicle
GUI	**G**raphical **U**ser **I**nterface
MUT	**M**anned-**U**nmanned **T**eaming
OS	**O**perative **S**ystem
DWR	**D**rone **W**atch and **R**escue
TCSP	**T**emporal **C**onstraint **S**atisfaction **P**roblem
UML	**U**nified **M**odeling **L**anguage
ERD	**E**ntity **R**elationship **D**iagram
JSON	**J**avascript **O**bject **N**otation

DBMS	**D**ata**B**ase **M**anagement **S**ystem
FPS	**F**rames **P**er **S**econd
UAM	**A**utonomous **U**niversity of **M**adrid
PCA	**P**rincipal **C**omponent **A**nalysis
HMI	**H**uman-**M**achine **I**nteraction
HRI	**H**uman-**R**obot **I**nteraction
UPGMA	**U**nweighted **P**air **G**roup **M**ethod with **A**rithmetic **M**ean
DIANA	**DI**visive **ANA**lysis Clustering
PAM	**P**artition **A**round **M**edoids
HRT	**H**uman-**R**obot **T**eam
HSC	**H**uman **S**upervisory **C**ontrol
KPP	**K**ey **P**erformance **P**arameter
SA	**S**ituational **A**wareness
KNN	**K**-**N**earest **N**eighbors
SVM	**S**upport **V**ector **M**achine

Chapter 1

Introduction

1.1 Motivation

The study of Unmanned Air Vehicles (UAVs) is currently a growing area. These new technologies offer many potential applications in multiple fields such as infrastructure inspection, monitoring coastal zones, traffic and disaster management, agriculture and forestry among others [1–4].

The use of UAVs, and unmanned systems, require the supervision of one or many human operators, responsible for monitoring the mission status continously and avoiding the possible incidents that might alter the execution and success of the operation. The work of these operators is extremely critical due to the high costs involving any UAV mission, both financial and human. Thus, lot of research in the field of human factors, and more specifically, in Human Supervisory Control (HSC) and Human-Robot Interaction (HRI) systems, have been carried out, in order to understand and improve the performance of these operators [5].

One of the key aspects in the field of HRI is the use of computer simulators, and their extension into videogames. There are at least three motivations for robot simulators, that apply to the world of Unmanned Aircraft Systems (UASs). One is the role of simulators in adoption of new technology, in this case the UAV, another is their potential for low-cost operator training, and finally their utility in research [6].

In recent years, two topics are emerging in relation to the study of UAS. One is the effort to design systems such that the current many-to-one ratio of operators to vehicles can be inverted, so that a single operator can control multiple UAVs. The other is related to the fact that accelerated UAS evolution has now outpaced current operator training regimens, leading to a shortage of qualified UAS pilots. Due to this, it is necessary to re-design the current intensive training process to meet that demand, making the UAV operations more accessible and available for a less limited pool of individuals, which may include, for example, high-skilled video-game players [7].

This work is focused on the design and development of a computer simulator that allows an operator to supervise and control the execution of a search and rescue mission carried out by a fleet of multiple UAVs. The goal of this simulator will be training and selecting inexperienced users in the world of the UAS, and thus, it will have the simplicity and accessibility typical from videogames.

Besides, the simulator will extract data from the operator interactions, allowing the measurement and analysis of the user performance and evolution. This performance data will be used to extract behavioral patterns among users, which could be used to select potential UAS operators.

1.2 Objectives

The aim of this project can be divided into two clearly distinguishable parts:

1. *Development of a lightweight multi-UAV simulator*: In this part of the work, there are many milestones that shall be accomplished:
    - To study of the state of the art in the development of computer simulator for flight systems, both manned and unmanned. We must focus in those simulators and videogames which allow an easy and quick understanding of the simulation concept and are suitable for beginners.
    - To choose the best software architecture, following the criteria of accessibility and ease of use.
    - To design a robust and complete set of input and output data.

- To design a set of gamification elements that challenge the operator and bring the simulator closer to the field of videogames.
- To define a set of operator controls for supervising and controlling the simulation.
- To design a simple and user-friendly Graphical User Interface (GUI) to show the execution of a simulation.
- To implement the simulator using the selected architecture.
- To test the simulator against hundreds of simultaneous users and to deploy it.

2. *Measurement and Analysis of the novice user performance using the simulation data*: Once we have designed, implemented and deployed the simulator as required above, we will carry out an experiment to prove its value for training and selecting inexperienced users. To achieve this, the following milestones will be accomplished:

    - To study the related work in the performance measurement in HRI systems, and the techniques to analyze and discover patterns from user activities.
    - To create a simulations dataset testing the simulator with a group of novice users.
    - To design a set of metrics that will define the performance in our simulator.
    - To use valid data mining techniques for extracting common groups of users, given its performance. Those groups will be analyzed to extract conclusions about the behavioral patterns in the dataset and the value of each group of users.

1.3 Document structure

This document is structured as follows: Chapter 2 reviews the state of the art in the aforementioned topics of UAV simulators and performance analysis on UAS operators. Then, Chapter 3 describes the simulator developed for this work, detailing its architecture, main elements, input and output data. In Chapter 4 we test the simulator and use it in a experiment with novice users. The entire process followed in the experiment is detailed, including the dataset, the design of performance metrics, the use and validation of clustering techniques over the data and the analysis of the experiment results. Finally, Chapter 5 presents the conclusions and future lines of research.

Chapter 2

Related Work

In this chapter, we introduce a state of the art on UAV Mission Simulators (UMSs), focusing on their main features, objectives, complexity and accessibility.

2.1 UAV Simulators

Computer simulations, and their extension into videogames, of Unmanned Systems (USs) are an emerging topic. There are at least three motivations for these type of simulators. One is the role of simulators in adoption of new technology, another is their potential for low cost training, and finally their utility in research. The four critera used to jugde the quality of any virtual simulator are defined in [8]:

1. *Physical Fidelity*: It can be described as the extent to which the virtual environment emulates the real world. A simulator with high physical fidelity is able to render the environment with high resolution textures, shaders, lighting, reflection, and bump mapping. A low physical fidelity simulator uses 2D rendering and no sound is required.
2. *Functional Fidelity*: The degree to which the simulation acts like the operational equipment in reacting to the tasks executed by the trainee. High functional fidelity is defined as the simulation of most of the forces acting on a vehicle and its actuators including gravity, drag, and accelerations from motors and collisions

on specific elements of the vehicle. A low functional fidelity simulator does not simulate forces applied to the vehicle but only velocities or absolute position.

3. *Ease of Development*: It is defined by how easy/difficult the simulator can be modified, and the available documentation from the author.

4. *Cost*: For a simulator to be useful it must not be time consuming to install or run and accessible in terms of initial monetary cost for both the developer and end user. The simulator developed in this work is focused on maximizing this criteria.

In [6], Craighead et al. survey multiple US simulators, both commercial and open-source, and provide a subjective rating of capabilities in terms of physical fidelity, functional fidelity, ease of use, and cost. For the purposes of this work, we focus only on those rated as "Low" in the Cost criteria. Table 2.1 summarizes the rating results of the aforementioned "Low cost" US Simulators:

- *FlightGear*: FlightGear [9] is a 3D open source simulator, very realistic and focused on simulating the flight of a single aircraft vehicle. It is available as a free download under GPL license. The entire source code is available for modification and is under constant development. The application runs on Windows, Mac, and Linux operating systems. It has been used for various academic projects. For example, Summers, et al. in [10] used FlightGear to simulate a UAV carrying environmental sensors and Cervin, et al. in [11] used FlightGear to create an interface for a real UAV.

- *Simbad*: Simbad [12] is a Java 3D robot simulator for scientific and educational purposes. It is mainly dedicated to researchers/programmers who want a simple basis for studying Situated Artificial Intelligence, Machine Learning, and more generally AI algorithms, in the context of Autonomous Robotics and Autonomous Agents. It is not intended to provide a real world simulation and is kept voluntarily readable and simple.

- *SimRobot*: SimRobot [13] is a physics based robot simulator with a 3D OpenGL based display. Several sensor types are supported, including cameras, range sensors, touch sensors, and actuator state. It was used by the German team for the 2005 RoboCup competition [14].

Table 2.1: A comparison of available Low-Cost Unmanned Vehicle Simulators.

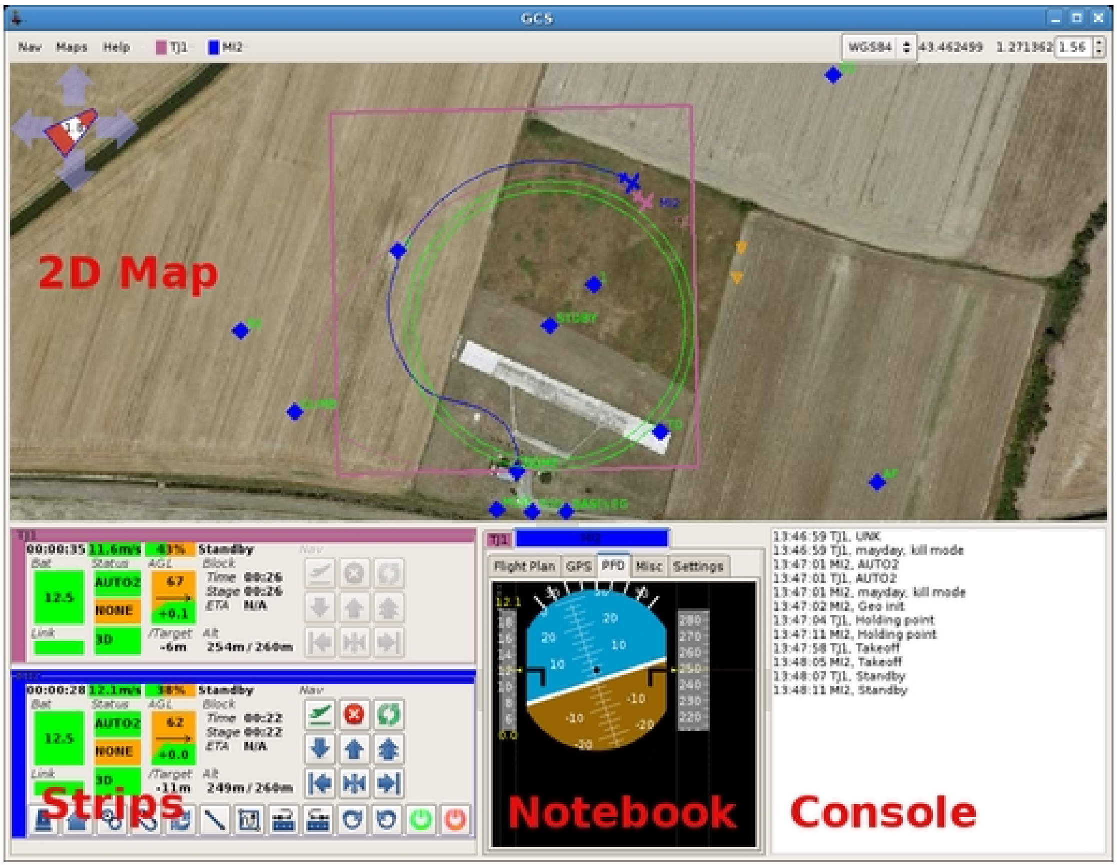
Simulator	**Physical Fidelity**	**Functional Fidelity**	**Ease of Development**	**Cost**
FlightGear	High	Medium	Medium	Low
Simbad	Medium	Low	Medium	Low
SimRobot	Medium	Low	Medium	Low

Analyzing the simulators detailed above, it is appreciable that the Functional Fidelity rating for all of them is not high, thus we cannot use them to easily analyze human control over them. Also, none of them focuses on the field of UAVs purely, but cover general unmanned robots or aircrafts instead. This is because at the time when these simulators were released, UAVs did not have as much importance as now.

Recently, the rapidly increasing interest in UAVs has caused that they are no longer part of a flight simulator or a type of robot in a general robot simulator. Small/micro UAVs have become applicable in civilian circumstances like remote sensing, mapping, traffic monitoring, search and rescue, etc. They are expendable, easy to be built and operated. Most of them can be operated by one or two people, or even be hand-carried and hand-launched [15, 16]. This has caused a large increase in the development of the so-called *Autopilots*.

Autopilots are systems to guide the UAVs in flight with no assistance from human operators, consisting of both hardware and its supporting software. In [17], both commercial and research autopilot systems for small UAVs are reviewed and discussed in detail. Since this work is not emphasized on hardware, the most remarkable autopilot from that survey, in terms of software development, is *Paparazzi*.

Paparazzi [18] is an open-source project, very popular among researchers, highlighted by offering good flexibility and ease to modify the autopilot based on own requirements (High "Ease of development" rate, following the criteria of [8]). For the software, it achieves waypoints tracking, auto-takeoff and landing, and altitude hold. Figure 2.1 shows how this software tries to imitate a real Ground Control Station (GCS). A disadvantage of Paparazzi (and more generally, of all autopilots surveyed in [17]), is the lack of support for cooperative control functions, required for some large area tasks that need multiple UAVs to perform them.



```
13:46:59 TJ1, UNK
13:46:59 TJ1, mayday, kill mode
13:47:01 MI2, AUTO2
13:47:01 TJ1, AUTO2
13:47:01 MI2, mayday, kill mode
13:47:02 MI2, Geo init
13:47:04 TJ1, Holding point
13:47:11 MI2, Holding point
13:47:58 TJ1, Takeoff
13:48:05 MI2, Takeoff
13:48:07 TJ1, Standby
13:48:11 MI2, Standby
```

FIGURE 2.1: Paparazzi GCS. The Paparazzi Ground Control Station is the heart of the system and the user's primary interaction interface.

2.1.1 Multi-UAV Simulators

The increasing demand and complexity of UAV applications has brought into focus several challenges associated with multiple UAVs [19]. Although several researchers have done quite some experiments in this topic [20–22], few research-focused simulators or autopilots have true multi-UAV functions built in.

The main research line concerned to the study of multi-UAV systems focuses on modelling the problem as a Multi-agent system. Thus, most multi-UAV simulators are used only as testbeds for cooperative models and algorithms. In [23], the commercially available *X-Plane* flight simulator (rated in the survey described above in [6]), together with MATLAB, are used to create a simulator framework for studying multi-UAV control algorithms. Likewise, in 2014, Pujol et al. developed *MAS-Planes* [24],a Multi-Agent Simulation Environment to investigate decentralized coordination for teams of UAVs. As can be seen, the operator interactions in this type of simulators takes second place.

However, recent works from Massachusetts Institute of Technology (MIT) [25–28] have studied and modeled the operator's behavior while using a simulator called RESCHU (Research Environment for Supervisory Control of Heterogeneous Unmanned Vehicles). In that simulator, the user gives commands to a relatively small number of UAVs and Unmanned Underwater Vehicles (UUVs), guiding them to stationary ground targets while avoiding hazard areas. It has been developed in Java, and the source code can be requested to the Human Automation Lab in MIT. In fact, some other works like [29] have customed and extended RESCHU to allow Manned-Unmanned Teaming (MUT) researches.

Other research lines around the field of multi-UAV simulations include the study of the best interface or set of interfaces for the operator to monitor the status of all UAVs. Related to this, the company Silicon Valley Simulation, specialized in real time visual simulation since 1996, has developed MUSIM (Multiple UAV Simulation) [30]), a flexible and modular UAV simulation environment used for research into the operator interface.

At the commercial level, the development of multi-UAV simulators focuses on getting closer to reality in terms of the management and control of UAVs. For example, the company *DreamHammer* [31] has developed *Ballista* [32], an Operative System (OS) for drones that allows one person to simultaneously control multiple drones of any type.

2.2 Performance Analysis of UAV operators

The human operator is a key component of unmanned systems. Historically, these systems have required a disproportionate degree of human involvement in their operations. For example, even more than 4 operators are needed to control a single Predator for most missions [33].

For these reasons, Human-Machine Interaction (HMI), and more specifically, HRI research, which are both subcategories of traditional human factors research, are emerging topics in the field of UAVs, and UAS in general.

2.2.1 Measuring the performance of a Human-Robot Team

A key obstacle in the growth of unmanned vehicle operations is the number of operators required to supervise and control an unmanned vehicle [34]. Increasing the *operator-vehicle ratio* is an open and desirable topic among researches nowadays, in order to reduce costs, extend human capabilities and improve system efficiency [35].

Operators in multi-UAV systems must be evaluated following the criteria of the field of HSC in Human-Robot Teams (HRTs) systems. According to the research of Crandall et al. in [36], the different *metric classes* (set of metrics) defining the effectiveness of a HRT should:

- *Contain the Key performance parameters (KPPs)*: A KPP is a measurable quantity that, while often only measuring a sub-portion of the system, indicates the overall effectiveness of the team.
- *Identify the limits of the agents in the team*: It is needed to measure the capacity of both human operator and robots in the team.
- *Have predictive power*: It is needed that the metrics have ability to generalize and predict the effectiveness of the system under uncertain or untested conditions.

For the goals of this work, we focus on the *metric class* of human performance. The most common are metrics based on the operator workload and Situational Awareness (SA). On the one hand, metrics for measuring operator workload include subjective methods [37], secondary task methods, as a chat interface [38], and psychophysiological methods [39]. On the other hand, SA, which is defined in [40], is still an open question when trying to measure it in an objective and non-intrusive manner [41]. However, in the field of HRI there have been many efforts to formalize the SA, including the works of Drury et al. in [42, 43], which establish a set of definitions for SA in a HRI environment, and determine that most critical accidents in the environment are directly attributable to lack of one or more of those definitions.

Apart from the workload and the SA, it is also interesting to define some metrics that collect the performance of an operator in a HSC environment in a direct way, as a type of global *score* indicating the quality of the performance. This work is focused on this type of measures, which are also known as *Direct measures of performance quality*, and are linked to the world of videogames, where these quality metrics create an *user profile*, which allow, on the one hand, to distinguish and group users by common skills, and on the other hand, to adapt the game based on the user expertise [44].

2.2.2 Extracting patterns in Human-Robot Interaction systems

When dealing with HMI, and more specifically with HRI systems, measuring the human performance, the system quality or other metric classes is just the beginning of all possible analysis that can be made over these type of systems.

The information given by the different metric classes, or just the information given by the human interactions in the system can help to recognize and extract some hidden information about the general use of the system, the different operator cognitive states during a mission, etc...

Here, the field of *data mining and machine learning* takes much importance, since it tries to extract valuable information and models from raw data [45]. An example of this can be seen in the works of Rani et al. in [46], which study different machine learning techniques, as K-Nearest Neighbors (KNN), Regression Trees, Bayesian Networks and Support Vector Machines (SVMs) to recognize affect states using physiological signals in a HRI environment.

In the field of UAV operations, the study of HRI pattern recognition and operator modelling is undoubtedly led by M.L. Cummings and the Massachusetts Institute of Technology. Their work to model and predict the operator behavioral patterns from HRI systems, and more specifically from HRT environments, consist of building *Hidden Markov Models* [47] representing behavioral states from the clicks that an operator make during a multi-UAV simulation [27, 28]. Apart from the good results shown in these works, it is remarkable to notice the conclusions they reach when comparing supervised vs unsupervised learning techniques when creating the operator models for multi-UAV systems. They say that, due to the fact that multi-UAV systems are still futurist developments, it is impossible to trust any expert trying to label the operator interactions in order to make an objective supervised analysis, hence we can only work in this field by using unsupervised learning techniques.

For this reason, the analysis made in this work is focused on one popular unsupervised technique: Clustering, which is detailed in next section.

2.2.3 Clustering in profile analysis

Clustering is an unsupervised technique used to group together, in a blindly way, objects which are similar to one another usually for the purpose of uncovering some inherent structure which the data possesses [48].

This technique is related to many disciplines and plays an important role in a broad range of applications, usually involving large datasets and many attributes. From biological fields, where it is commonly used with the aim of grouping together genes or proteins which have similar expression patterns [49, 50], to technological fields, where, for example, it is widely used to group Wireless Sensor Network nodes into disjoint clusters [51]. Other important clustering applications include time series-clustering [52] and text-mining [53].

Regarding to this work, clustering can be seen as a way to discover patterns among user activities. One popular example of this application is *Web usage mining*, which consists in applying data mining techniques (including clustering) to discover usage patterns from Web data, in order to understand and better serve the needs of Web-based applications [54].

There are a lot of clustering algorithms, and deciding which to use might be a difficult task for a research conducting a experiment. For the goals of this work, we introduce five clustering methods from the state of the art:

1. *Agglomerative Nesting*: Also called Unweighted Pair Group Method with Arithmetic Mean (UPGMA), or just *Hierarchical*, this is one of the most frequently used clustering algorithms [55]. It is a bottom-up, non-parametric hierarchical algorithm, which seeks to build a hierarchy of clusters. Each observation is initially placed in its own cluster, and the clusters are iteratively joined together according to their closeness. This closeness of any two clusters is measured by a dissimilarity matrix between sets of observations, usually achieved by use of an appropriate metric (Euclidean distance or Manhattan distance, among others). The results of

this algorithm (and all hierarchical methods) are usually presented in a *dendrogram*, as shown in Figure 2.2. This dendrogram can be cut at a chosen height to produce the desired number of clusters.

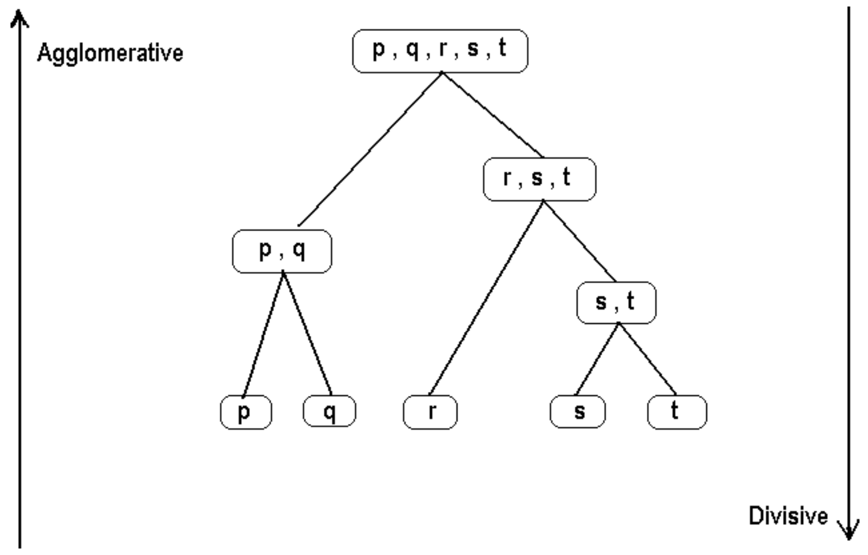


FIGURE 2.2: Example of a dendrogram resulted from a Hierarchical Clustering method.

2. *Divisive Analysis Clustering (DIANA)*: DIANA [55] is a divisive hierarchical algorithm that constructs the hierarchy in the inverse order (*top-down*). It initially starts with all observations in a single cluster, and successively divides the clusters until each cluster contains a single observation. The results are presented in a dendrogram, as in the case of UPGMA (See Figure 2.2). Although it is usually less efficient than the agglomerative nesting, DIANA stands out as a competitive clustering algorithm for many fields [56].

3. *K-Means*: K-Means [57] is one of the most popular methods for cluster analysis, belonging to the family of *partitional* clustering methods. The algorithm starts with an initial guess for the cluster centers, and each observation is placed in the cluster to which it is closest. The cluster centers are then updated, and the entire process is repeated until the cluster centers no longer move. In the end, each observation belongs to the cluster with the nearest center, resulting in a partitioning of the data space (See Figure 2.3). The problem is computationally difficult (*NP-hard*), but there are efficient heuristic algorithms that are commonly employed and converge quickly to a local optimum [58]. It is well-known that the K-Means algorithm suffers from initial starting conditions effects (initial clustering

and instance order effects), and many techniques have been developed to avoid this issue [59].

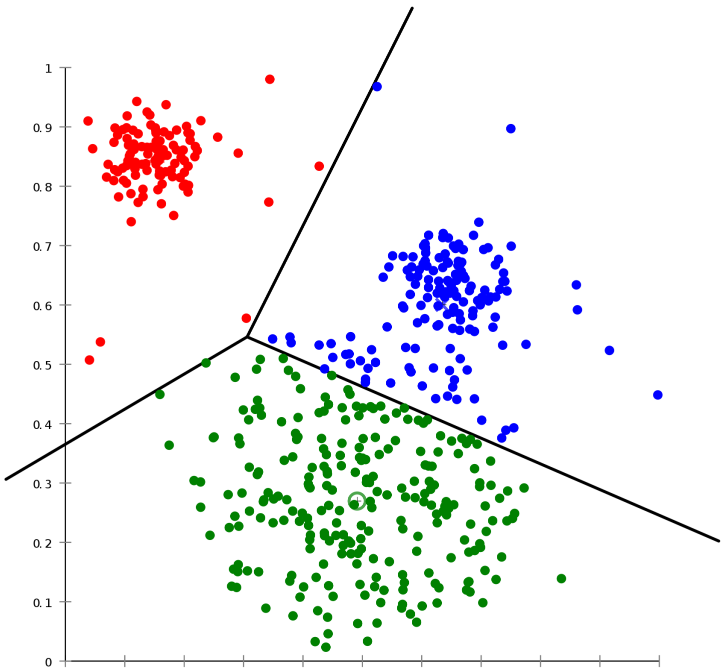


FIGURE 2.3: Example of a K-means clusterization.

4. *Partition Around Medoids (PAM)*: Proposed by Kaufman et al. in [60], this algorithm is similar to K-means, but is considered more robust because it admits the use of other dissimilarities besides Euclidean distance. In contrast to the k-means algorithm, PAM chooses datapoints as centers (called *medoids*) instead of centroids.

5. *Model-based clustering*: This algorithm, proposed by Fraley et al. in [61], fits the dataset using a mixture of Gaussian distributions. Each distribution represents a cluster, and its members are estimated using maximum likelihood estimations (MLE), via the popular Expectation Maximization (EM) algorithm [62].

In order to assess the quality of a clusterization, and to compare and decide which clustering algorithm is better for a specific dataset, the data-mining literature provides a range of different validation techniques, with the main line of distinction between *external* and *internal* validation measures [63].

External validation measures comprise all those methods that evaluate a clustering result based on the knowledge of the correct class labels. Obviously, this is only useful when the class labels are available. This type of measures are commonly used to compare clustering algorithms on benchmark data, not in real datasets. Some of the best known external metrics are the *F-measure* [64], that assesses the quality of a clustering result at the level of the entire partitioning, the *Rand Index* [65], which determines the similarity between two partitions as a function of positive and negative agreements in pairwise cluster assignments, and the *Jaccard coefficient* [66], in which only positive agreements are rewarded.

For the goals of this work, the most important clustering validation measures to help us to choose an algorithm are the **internal validation measures**. These measures take a clusterization and use information intrinsic to the data to assess the quality of the clustering. Handl et al. in [67] give an overview of some quality notions that internal measures usually employ:

- *Compactness*: Assesses cluster homogeneity by looking at the intra-cluster variance. The less variance a cluster has, the more homogeneous it is considered. This is the criterion followed by the K-means algorithm to build the clusters.
- *Connectedness*: Assesses to what extent observations are placed in the same cluster as their nearest neighbors in the data space.
- *Separation*: Quantifies the degree of separation between clusters (usually by measuring the distance between cluster centroids). A good clusterization should maximize this value.

Based on these quality notions, there are multiple internal validation measures that not only focuses on one of them, but also make non-linear combinations between them, specially between compactness and separation, since they demonstrate opposite trends (compactness increases with the number of clusters but separation decreases). For this work, three internal validation measures from the state of the art are studied and used:

1. *Connectivity*: It is focused on improving the clustering quality in terms of connectedness. Given a clusterization $C = C_1 \ldots, C_k$ of N observations into K clusters, and let $nn_{i(j)}$ be the jth nearest neighbor of observation i, the connectivity is defined as:

$$Conn(C) = \sum_{i=1}^{N} \sum_{j=1}^{L} x_{i,nn_{i(j)}}, \tag{2.1}$$

where $x_{i,nn_{i(j)}}$ is 0 if i and j belong to the same cluster and $1/j$ otherwise. L controls the number of neighbors evaluated for each observation (regulates the accuracy of the metric). Higher values in this measure mean that clusters are not represented by near elements in the data space, so a good clusterization should minimize this value.

2. *Dunn Index*: The Dunn Index [68] combines separation and compactness in the same measure. It is defined as the ratio of the smallest distance between observations not in the same cluster to the largest intra-cluster distance:

$$D(C) = \frac{\min_{C_k,C_l \in C, C_k \neq C_l} \left(\min_{i \in C_k, j \in C_l} dist(i,j)\right)}{\max_{C_m \in C} diam(C_m)}, \tag{2.2}$$

where $diam(C_m)$ represents the diameter of C_m, i.e. the maximum distance between observations in the cluster. Since a good clusterization requires high levels of separation (measured in the denominator) and low levels of intra-cluster distance (high connectedness), this measure should be maximized in good clustering results (It takes values between zero and ∞).

3. *Silhouette Width*: The silhouette of an observation in a specific clusterization measures the degree of confidence with which we can ensure that the observation really belongs to the cluster it is assigned [69]. Given an observation i the silhouette for that observation, $S(i)$, is defined as:

$$s(i) = \frac{b_i - a_i}{\max\left(b_i, a_i\right)}, \tag{2.3}$$

where a_i is the average intra-cluster distance for i, and b_i the average inter-cluster distance with respect to the nearest cluster to i, i.e:

$$b_i = \min_{C_k \in C \setminus C(i)} \sum_{j \in C_k} \frac{dist(i,j)}{n(C_k)}, \tag{2.4}$$

where $C(i)$ represents the cluster to which i is assigned, and $n(C_k)$ the number of observations contained in cluster C_k. The closer $s(i)$ gets to 1, the more confidence we have of i as well-assigned, and viceversa if $s(i)$ gets close to -1. Finally, to compute the Silhouette width of a clusterization, we simply compute the average

Silhouette value for each observation:

$$S(C) = \frac{\sum_{C_k \in C} \sum_{i \in C_k} s(i)}{|C|} \tag{2.5}$$

The result lies in $[-1, 1]$, and should be maximized in order to achieve a good clusterization.

Chapter 3

Design and Development of a lightweight multi-UAV simulator

This chapter details the different processes involved in the design and development of a Multiple Unmanned Aircraft Vehicles (multi-UAV) mission simulator, which has been named as *Drone Watch And Rescue* (a.k.a DWR). The following sections are structured as follows: First we introduce the backgrounds and requisites that the simulator development must accomplish. Then, Drone Watch And Rescue (DWR) is presented and described, detailing all the elements comprising it and taking part of a simulation. After that, we show the simulator GUI and list the different interactions that can be made by an operator/player during the execution of a mission. Finally, we will detail the web architecture on which the simulator is built, necessary to achieve the lightness and accessibility required.

3.1 Requisites

The simulator to develop is not intended to achieve a high simulation fidelity level in terms of graphics, physics, and technical topics related to UAVs. According to the four criteria defined in [8] to evaluate a simulator (See Chapter 2), this simulator should be rated as : [*Physical Fidelity*: LOW, *Functional Fidelity*: LOW, *Ease of Development*: MEDIUM, *Cost*: HIGH].

The main goal of DWR is to collect easily big amounts of data from the interactions made by UAV operators (regardless of their level of expertise) during the execution of a multi-UAV mission. Below are described the main issues to address in order to achieve this goal.

3.1.1 Mission Planning Load

As was noted above, the simulator to develop does not focus on achieving high levels of fidelity in technical aspects of UAVs. However, the issue of *Mission Planning* for multiple UAVs is critical for this simulator, since it represents the logical core in terms of movement, cooperation and control of each UAV. Due to this, it is required that the simulator is able to load sophisticated mission plans.

A mission for multiple UAVs is usually planned following three steps [70, 71]:

1. The operator or *mission manager* establish, inside a mission map, which are the map zones where UAVs will flight, and the type of task to perform in each of them (Photographing the zone, Surveillance, Mapping, etc.).
2. Data obtained in step 1 is input to the *Mission Planner*, along with information about the mission environment (No Flight Zones, Refueling Stations...) and the available resources (UAVs, available fuel for each UAV, airports...).
3. The *Mission Planner* computes one or many feasible mission plans to accomplish the defined tasks using the available resources. Given that, each UAV is assigned to a set of waypoints that will guide its flying path throughout the different task zones. Each waypoint contains the time when the UAV must reach a position, and the action to perform when the waypoint is arrived.

For this work, the mission plans that feed the simulator will be generated utilizing the *Mission Planner* developed by Ramirez-Atencia et al. in [72–74]. This work models the problem of *Mission Planning* as a Temporal Constraint Satisfaction Problem (TCSP), and returns, for an specific mission, a list of task assignments for each UAV taking part of the mission.

3.1.2 Gamification Elements

A pure multi-UAV simulator could be reduced only to the execution of a mission plan, with any possible interaction, in order to check if the mission is correctly designed. However, since this work is aimed to analyze data from operators, the simulator to develop must allow control of both the UAVs and the mission plan of a simulation. This makes the simulator an interactive tool, and brings it closer to the world of videogames, which is commonly known as a process of *gamification* [75]. This is a term for the use of video game elements in non-gaming systems to improve user experience and user engagement. Following the *gamification* idea, there are some elements that should be added to this simulator.

As a way of engaging the operators and focusing them on a main challenge to comply during a simulation, the simulator to develop will include mobile *targets* to watch and rescue. Therefore, the main goal of the execution of the simulator will consist in finding the maximum number of targets while consuming the minimum possible resources.

Furthermore, during the execution of a mission with multiple UAVs, several **incidents** may occur, altering both the mission environment and the UAVs performing it. Below are listed some examples of possible incidents:

- Apparition of a hazard flight zone, due to meteorological conditions or other type of threat.
- A UAV's sensor breaks and stops working (radar, camera...). This may make the UAV unable to perform some tasks.
- Data link loss between a UAV and the GCS controlling it. This turns the UAV invisible, and unable to receive any operator command.

  When an incident appears, the operator must respond by replanning the UAV's path in order to guarantee that the mission goal is accomplished, and no UAV is destroyed. These actions are extremely fragile and decisive, and suppose an extra challenge for the operators. In fact, there are numerous studies that aim to facilitate these replanning actions and reduce the stress levels of the operators performing them [76, 77].

3.1.3 Data extraction

The main motivation to develop this simulator lies in the necessity to collect big amounts of data from the simulations. To achieve this, the simulator to develop must fulfill the following properties:

- *Accessibility*: The simulator must be easily and quickly accessible, in terms of installation and deployment for anyone trying to use it. Likewise, the system requirements must be low, so that any personal computer or laptop can execute simulations.
- *Portability*: The simulator must be ready to execute in any platform and operative system.
- *Simplicity*: The simulation mechanisms, the GUI and the controls must be simple, basic and clear, even for non-expert users.

The data collected by this simulator must represent a simulation *robustly*. This means that, knowing the simulation mechanisms and rules, every simulation instance could be completely replayed inductively only by observing the data stored.

3.2 Simulator Description

This section describes the mechanism and elements comprising the simulator created in this work, in accordance with the requisites defined in Section 3.1. As was said at the beginning of this chapter, the name of the developed simulator is *Drone Watch And Rescue* (DWR), in relation with the main goal of the simulator, which is detecting a set of targets using multiple UAVs.

3.2.1 Simulation Elements

Below are described the different elements that compose DWR:

3.2.1.1 UAVs

UAVs are the primary element of the simulator. The main features of each UAV involved in an specific mission are described as part of the Mission Scenario, loaded as part of the input data (See Section 3.2.2). In order to perform the mission tasks, each UAV may have one or more *sensors*. There are many type of *sensors*, and although many of them are loaded into the simulator, only **radars** are considered during the execution of a simulation.

Radars detect mobile targets around the zones overflown by UAVs. Although there exist many radar types in the database from which the input data is loaded, the simulator does not distinguish between them. A radar is defined only by a fixed detection ratio, within which any target placed inside will be considered as detected.

Each UAV starts a simulation on an Airport (See 3.2.1.5), and follows a flight path composed of a list of waypoints. There are two main classes of waypoints: those that are as part of the mission plan, called *pre-planned waypoints*, and those added or modified by the operator during the simulation, called *operator-waypoints*. Each waypoint is defined by the following attributes:

- *Position*: Given by the duple $(Latitude, Longitude)$ or (x, y), depending on the coordinate system used in the Mission Scenario. The altitude of a waypoint is not modelled in DWR. This is because we make the assumption that each of the UAVs taking part of a simulation flies around an independent altitude range, so there is no collision risk among UAVs. Therefore, we can conclude that the trajectory that will be simulated for each UAV, despite being loaded as a three-dimensional trajectory, will be computed as a plane trajectory.
- *Type*: The waypoint type will define which kind of action will be performed by the UAV when it reaches it. The possible values for this attribute are: *Route, Refueling, Landing, Take-off, Task*. A *Task* waypoint includes all possible tasks to perform (See 3.2.1.2).
- *Estimated time of Arrival*: Measured from the beginning of the mission (time 0), this attribute indicates the time in which the UAV is expected to reach the waypoint. This value is given as part of the computations made by the *Mission*

Planner developed by Ramirez-Atencia et.al in [72–74] (See 3.2.2). If the user adds or modifies a waypoint during the simulation, the simulator does not recalculate this value.

- *Action*: Each waypoint can be associated to an action. This action will be executed by the UAV when it reaches the waypoint. Waypoints may have a default associated action type in accordance with its type:
  - *Route, Take-off*: No action associated.
  - *Task*: The action associated triggers the beginning of a task, which has been previously assigned to the UAV in the Mission Plan.
  - *Refueling*: Refueling waypoints always have a "Refueling" action associated, whether they are loaded as part of the mission plan (pre-planned waypoints) or they are generated by the simulator (operator waypoints)
  - *Landing*: Similar to Refueling waypoints, in this case with the action "Landing"

3.2.1.2 Actions

An **action** encompasses all possible things that a UAV can do during the execution of a simulation, apart from flying from point to point. In DWR, actions are associated to waypoints, so they are implicitly linked to a specific position on the map. Every action has a finite duration defined either by default or by the mission input data. Below are detailed the different actions modeled in DWR:

- *Landing*: This action is always associated to waypoints positioned in airports (See 3.2.1.5). When a UAV lands, its sensors will be fixed in case they were broken by an incident (See 3.2.1.7). If this action is commanded by an operator, the duration considered is 0, which means that once the UAV reaches the airport, it is considered as landed instantly.
- *Refueling*: This action is always associated to waypoints positioned in refueling stations (See 3.2.1.4). Depending on the duration of the action, the UAV will charge more or less amount of fuel. In case this action is commanded by an operator during the simulation, the refueling duration is always 20 seconds. On

the contrary, if the action is pre-planned by the *Mission Planner*, this value is variable.

- *Task*: A task is a special type of action, and as an action, it is triggered when an UAV reaches the associated waypoint. Every task has a scope zone (area) associated, where the task is executed, and a time interval in which it must be completed. Tasks are always part of the mission plan loaded as part of the input data (See 3.2.2). This means that an operator cannot create tasks on the fly, during the execution of a simulation. DWR only models one type of task: **Surveillance**, which consists in exploring an area searching for targets. The set of tasks assigned to a UAV is commonly called as **payload**

3.2.1.3 No Flight Zones

No Flight Zones are volumes and areas where the flight of any UAV is forbidden. They are defined in the Mission Scenario loaded as part of the input data for any simulation (See 3.2.2). During the simulation, if a UAV flies within any of these zones, it will be immediately destroyed.

3.2.1.4 Refueling Stations

Refueling Stations charge the UAV's fuel. They are defined in the Mission Scenario loaded as part of the input data for any simulation (See 3.2.2). Each of them is described by the position they are located on the map (geodesic or cartesian coordinates), the maximum amount of fuel they can store and the refueling speed they can achieve. During the simulation, a *Refueling* action is triggered whenever a UAV reaches a waypoint of type Refueling, and it ends when the UAV gets full or when the refueling station runs out of fuel.

3.2.1.5 Airports

Airports are the starting and ending point for every UAV taking part of a mission. They are defined in the Mission Scenario loaded as part of the input data for any simulation (See 3.2.2). An airport can be defined by a point on the map or an area.

3.2.1.6 Targets

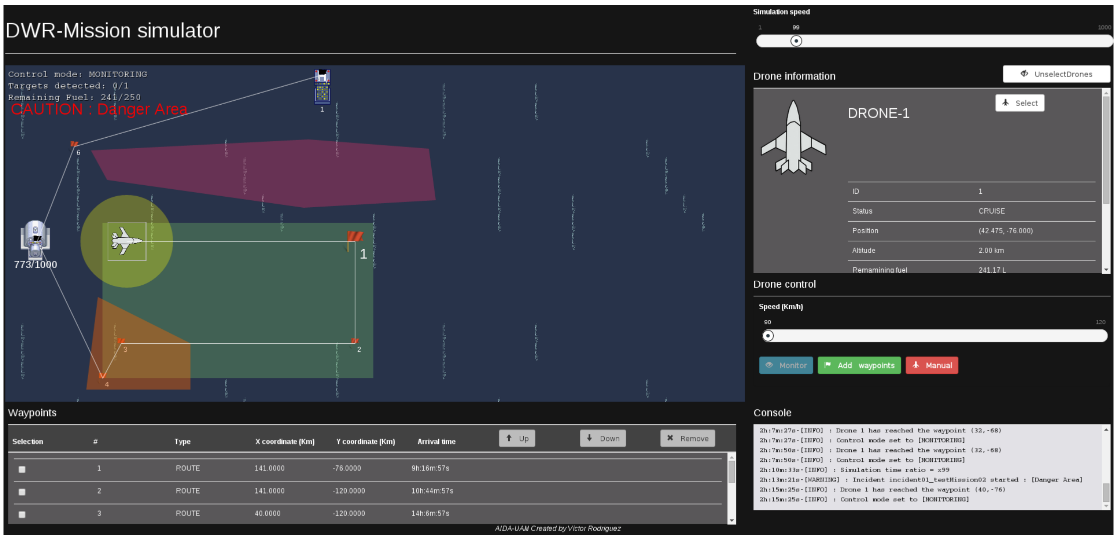
Targets are one of the gamification elements of DWR. They are not part of either the Mission Scenario or the Mission Plan, so they must be loaded from a different source (See 3.2.2). The trajectory they follow will be generated randomly by the simulator, and it will always be bounded to a specific area. During the simulation, the operator will not be able to see the targets moving, only the areas where they could be found. When a UAV detects a target, the latter is immediately removed from the simulation.

3.2.1.7 Incidents

Incidents in DWR are asynchronous events that occur during the course of a simulation and alter, either temporarily or permanently, both the environment and the UAVs taking part in the mission. DWR loads an *Incident Plan* from the input data (See 3.2.2). The aim of adding incidents to the simulation is to challenge the operator, forcing him to make use of the simulation controls in order to adjust some parameters and avoid the incidents successfully.

An Incidents Plan is composed of a list of incidents scheduled over the mission time. Appendix A contains the information included in each of these incidents. Among this information, two attributes are remarkable:

- *Start Time*: Defines the time, measured from the beginning of the mission (time 0) in which the incident will start. When DWR runs this mission, and the timeline reaches this start time, the simulator will show the incident in screen. Figures 3.1,3.2 show how DWR displays the incidents.
- *Incident Type*: There are two types of incidents defined in DWR:
    1. *Danger Area*: Due to a heavy storm or any other reason, a new danger area appears somewhere in the map. When a UAV overflies it, it will be automatically destroyed. To overcome this incident, an operator must change the flying path of the UAVs taking part in the mission (See Figure 3.1).
    2. *Payload Breakdown*: The sensors conforming the UAV's payload stop working. From this moment, the UAV is not able to perform any task successfully



```
Control mode: MONITORING
Targets detected: 0/1
Remaining Fuel: 241/250
CAUTION : Danger Area
```

DRONE-1

ID	1
Status	CRUISE
Position	(42.475, -76.000)
Altitude	2.00 km
Remamining fuel	241.17 L

Waypoints

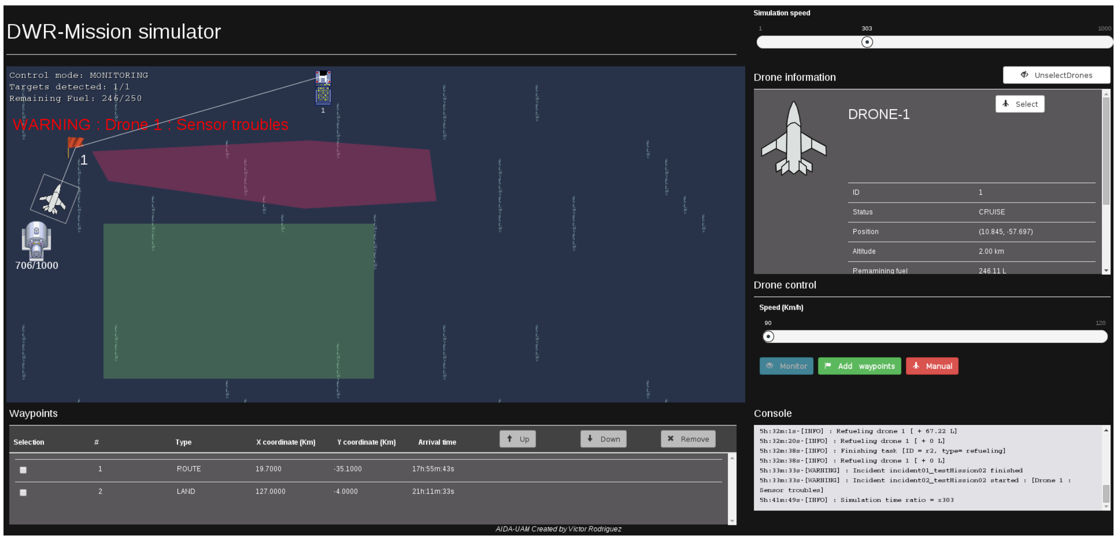
Selection	#	Type	X coordinate (Km)	Y coordinate (Km)	Arrival time
☐	1	ROUTE	141.0000	-76.0000	9h:16m:57s
☐	2	ROUTE	141.0000	-120.0000	10h:44m:57s
☐	3	ROUTE	40.0000	-120.0000	14h:6m:57s

Console

```
2h:7m:27s-[INFO] : Drone 1 has reached the waypoint (32,-68)
2h:7m:27s-[INFO] : Control mode set to [MONITORING]
2h:7m:50s-[INFO] : Drone 1 has reached the waypoint (32,-68)
2h:7m:50s-[INFO] : Control mode set to [MONITORING]
2h:10m:33s-[INFO] : Simulation time ratio = x99
2h:13m:21s-[WARNING] : Incident incident01_testMission02 started : [Danger Area]
2h:15m:25s-[INFO] : Drone 1 has reached the waypoint (40,-76)
2h:15m:25s-[INFO] : Control mode set to [MONITORING]
```

FIGURE 3.1: DWR screenshot of a *Danger Area Incident*. The orange area represents the new No Flight Zone generated by the incident.

nor detect any target. To overcome this incident, the operator must command the affected UAV to return to its base airport, where it will be repaired (See Figure 3.2).



```
Control mode: MONITORING
Targets detected: 1/1
Remaining Fuel: 246/250
WARNING : Drone 1 : Sensor troubles
```

DRONE-1

ID	1
Status	CRUISE
Position	(10.845, -57.697)
Altitude	2.00 km
Remamining fuel	246.11 L

Waypoints

Selection	#	Type	X coordinate (Km)	Y coordinate (Km)	Arrival time
☐	1	ROUTE	19.7000	-35.1000	17h:55m:43s
☐	2	LAND	127.0000	-4.0000	21h:11m:33s

Console

```
5h:32m:1s-[INFO] : Refueling drone 1 [ + 67.22 L]
5h:32m:20s-[INFO] : Refueling drone 1 [ + 0 L]
5h:32m:38s-[INFO] : Finishing task [ID = r2, type= refueling]
5h:32m:38s-[INFO] : Refueling drone 1 [ + 0 L]
5h:33m:33s-[WARNING] : Incident incident01_testMission02 finished
5h:33m:33s-[WARNING] : Incident incident02_testMission02 started : [Drone 1 :
Sensor troubles]
5h:41m:49s-[INFO] : Simulation time ratio = x303
```

FIGURE 3.2: DWR screenshot of a *Payload Breakdown Incident*. Note how the yellow circle around the UAV representing the Radar has disappeared.

3.2.2 Input Data

Before starting a simulation, DWR must load all data related to the mission to simulate. A mission is composed of two main objects: the *Mission Scenario* and the *Mission Plan*. Both of them are **essential components** of any UAV mission in any UAV simulator. Besides this, DWR adds another interesting components to a mission, in order to achieve the gamification process required in 3.1. Those are the *Incidents Plan* and the *Targets definition*, which can be seen as **gamification components**. Together, essential components and gamification components compound the mission input, necessary for DWR to start a simulation (See Figure 3.3). Below are detailed each of the mission components.

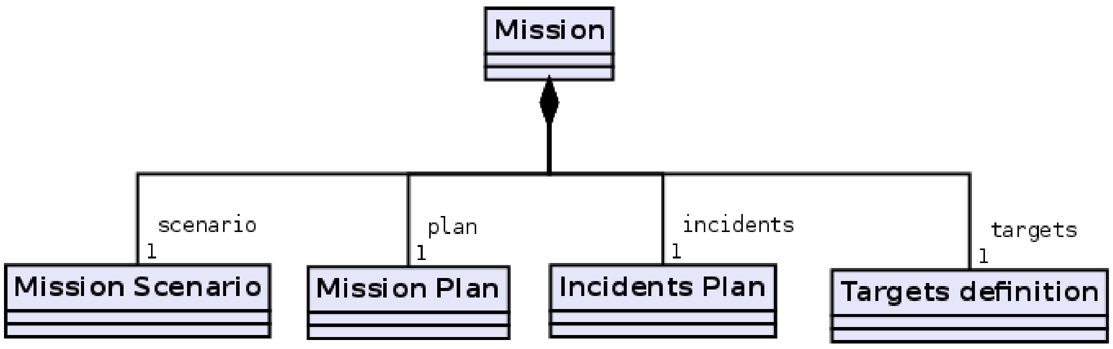


FIGURE 3.3: General schema of DWR mission input, showing the four components compounding a mission.

3.2.2.1 Mission Scenario

The Mission Scenario gathers all the information about the map, UAVs and other environmental elements concerning to a specific mission. **This scenario is the same that uses the *Mission Planner* developed by Ramirez-Atencia et al. in [72–74] as input data**. This way we ensure that DWR missions are defined in a robust way, in accordance with current works in this field. Figure 3.4 shows a Unified Modeling Language (UML) diagram describing all elements compounding a Mission Scenario. It is important to note that some elements showed in this figure, despite being loaded, are not used during the simulation execution (e.g: Cameras).

Hence, based on figure 3.4, we can describe which elements comprise a Mission Scenario:

Hence, based on these diagrams, we can describe what elements comprise a Mission Scenario:

- *No Flight Zones*: Loaded as a polygonal area.
- *Airports*: The simulator loads its identifier and position.
- *UAVs*: All attributes showed in figure 3.4 are loaded and used by the simulator.
- *Refueling Stations*

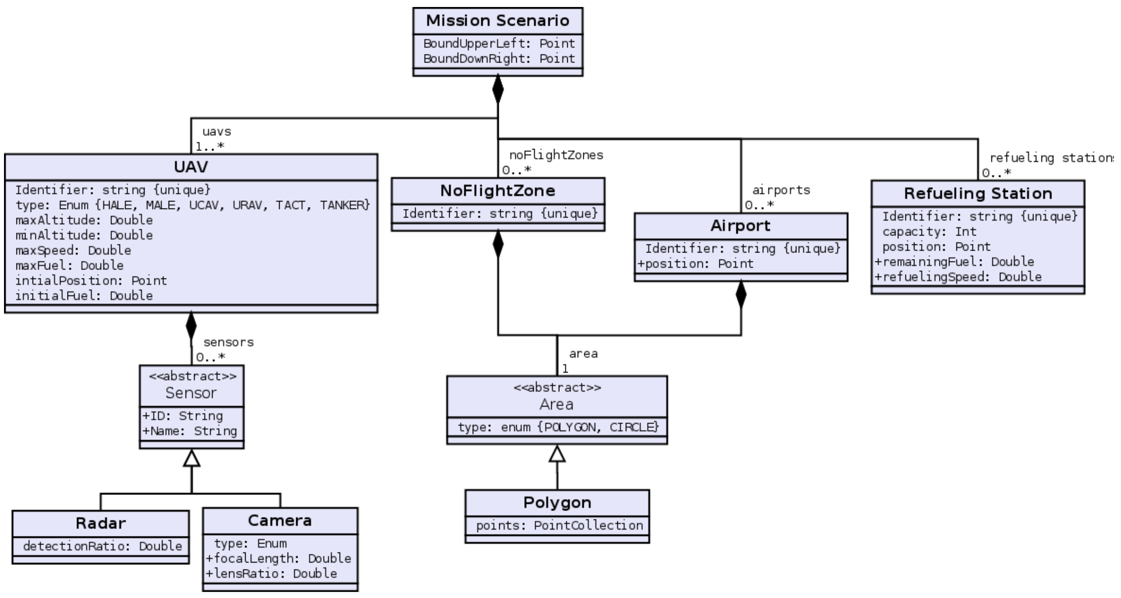


FIGURE 3.4: Mission Scenario Data diagram. Classes and attributes are deeply detailed in Appendix A

Appendix A contains a deeper description of each attribute comprising a Mission Scenario.

3.2.2.2 Mission Plan

The *Mission Plan* is the result of running the Mission Planner developed by Ramirez-Atencia et al. in [72–74]. It assigns, for each UAV, the list of tasks it must perform, and the time window $(starttime, endtime)$ in which this tasks should be completed. The simulator must load this plan and transform the task assignation of each UAV into a set of waypoints that will guide the UAVs flying path. Figure 3.5 shows, diagrammatically, the elements comprising a Mission Plan.

As can be seen in the figure, a Mission Plan consists of a set of *Objectives*, each of them composed of a set of *Tasks*. Each Task is assigned to the *zone* where it will be performed (defined as an area), the list of sensors that a UAV needs to complete it and the specific UAV that will perform it (assignation). So far, the simulator only works with 1-task objectives, and does not treat any objective nor task time dependencies. In fact, as described in 3.2.1.2, the only task type modeled DWR so far is the *Surveillance* task.

The simulator loads the Mission Plan as follows: For each Task, it gets the assigned UAV and appends two waypoints to its flying path: The first one is located at the Task

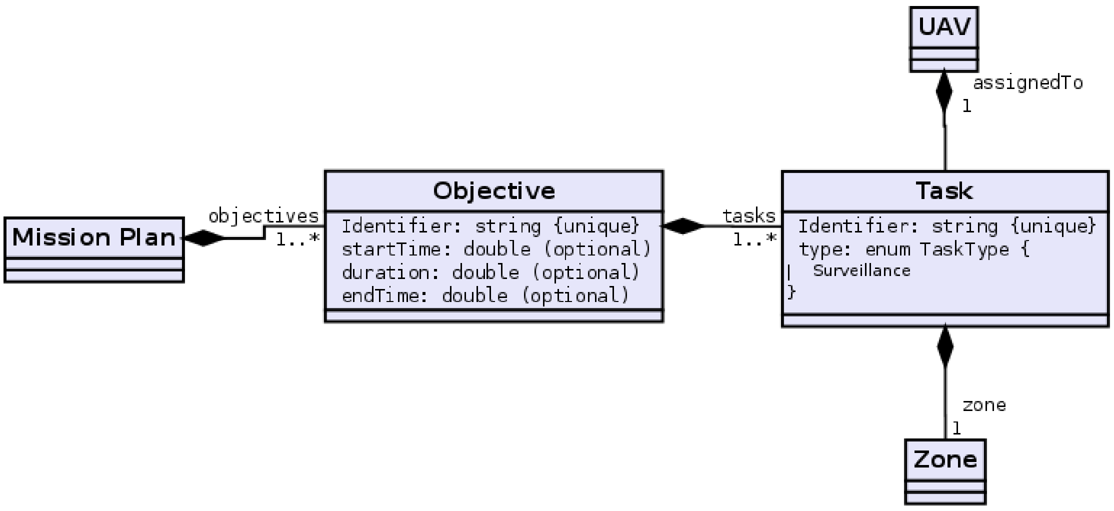

UAV
assignedTo
1
Mission Plan
objectives
1..*
Objective
Identifier: string {unique}
startTime: double (optional)
duration: double (optional)
endTime: double (optional)
tasks
1..*
Task
Identifier: string {unique}
type: enum TaskType {
| Surveillance
}
zone
1
Zone

FIGURE 3.5: Mission Plan Data diagram. Classes and attributes are deeply detailed in Appendix A

entry point, and marks the task beginning. The second one is located at the Task exit point and marks the end of the task. When all tasks have been loaded, a final return waypoint is added to the flying path of every UAV, so that they finish the mission in the same point that they started it (usually an airport).

3.2.2.3 Incidents Plan

An *Incidents Plan* schedules, for a specific mission, all incidents that will appear during the mission simulation. It is designed in order to test and train the skills of operators. Figure 3.6 shows a UML data diagram for an Incidents Plan. An Incidents Plan is composed of a list of incidents scheduled over the mission time. Appendix A contains the information included in each of these incidents.

3.2.2.4 Targets definition

As was discussed in section 3.1, it is needed to introduce *targets* to detect into a simulation as a way to challenge the operators. The number and description of these targets is given by the **targets definition** entity, as showed in figure 3.7.

Basically, a *targets definition* entity is composed of a list of identified targets. Each target is associated to an area inside the mission map. During the simulation, targets will move randomly (controlled by the simulator), but they will never go outside its associated area.

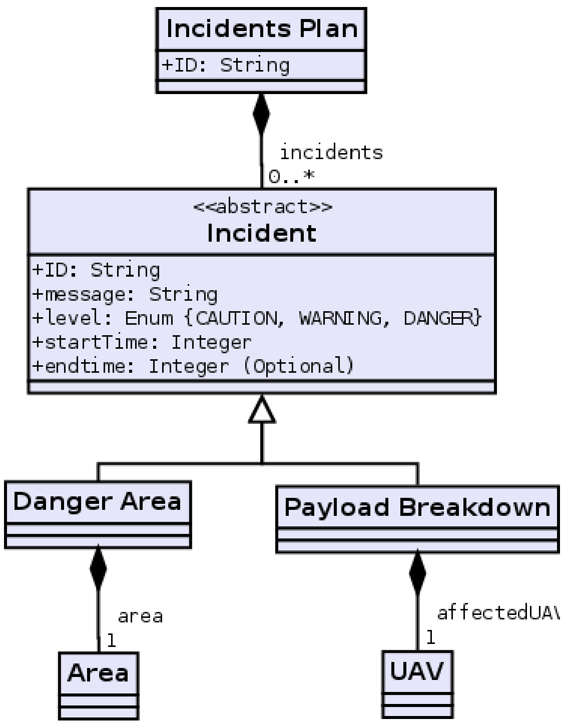


FIGURE 3.6: Incidents plan data diagram. Classes and attributes are deeply detailed in Appendix A

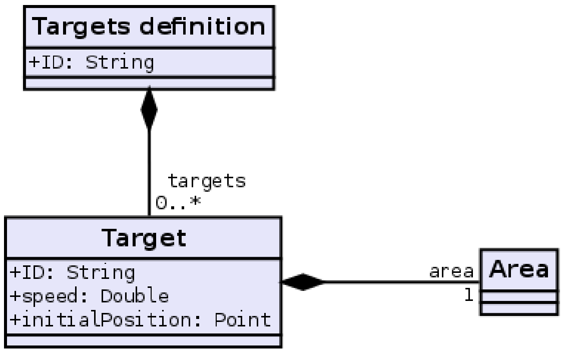


FIGURE 3.7: Targets definition data diagram. Classes and attributes are deeply detailed in Appendix A

3.2.3 Output Data

Undoubtedly, retrieving data from simulations is a key factor in the design and development of DWR. The experimentation made in this work (See chapter 4) needs a dataset containing all relevant data extracted by the simulations run in DWR, so it is critical to define the output data so that no information is missed throughout the execution of the simulator.

Figure 3.8 shows a Entity Relationship Diagram (ERD) diagram containing the designed output data scheme for DWR, which will be implemented in a database (See architecture in section 3.5). The data organization has been designed following an **event-driven** design pattern. Whenever an event occurs during a simulation run, we store the simulation status in that moment, making what is commonly called a *Simulation Snapshot*. This snapshot contains all relevant information of the current status of every element taking part in the simulation (See appendix A), as well as a set of *Drone Snapshots*, representing the current status of each of the UAVs still participating in the mission (If a UAV has been destroyed, it will have no associated *Drone Snapshots* after its destruction). Storing the data in this way, we ensure that no data is lost, thus we obtain a complete track of the simulation run and we are able to develop a robust analysis. In fact, we could make a re-execution of a simulation inducted only by the data stored.

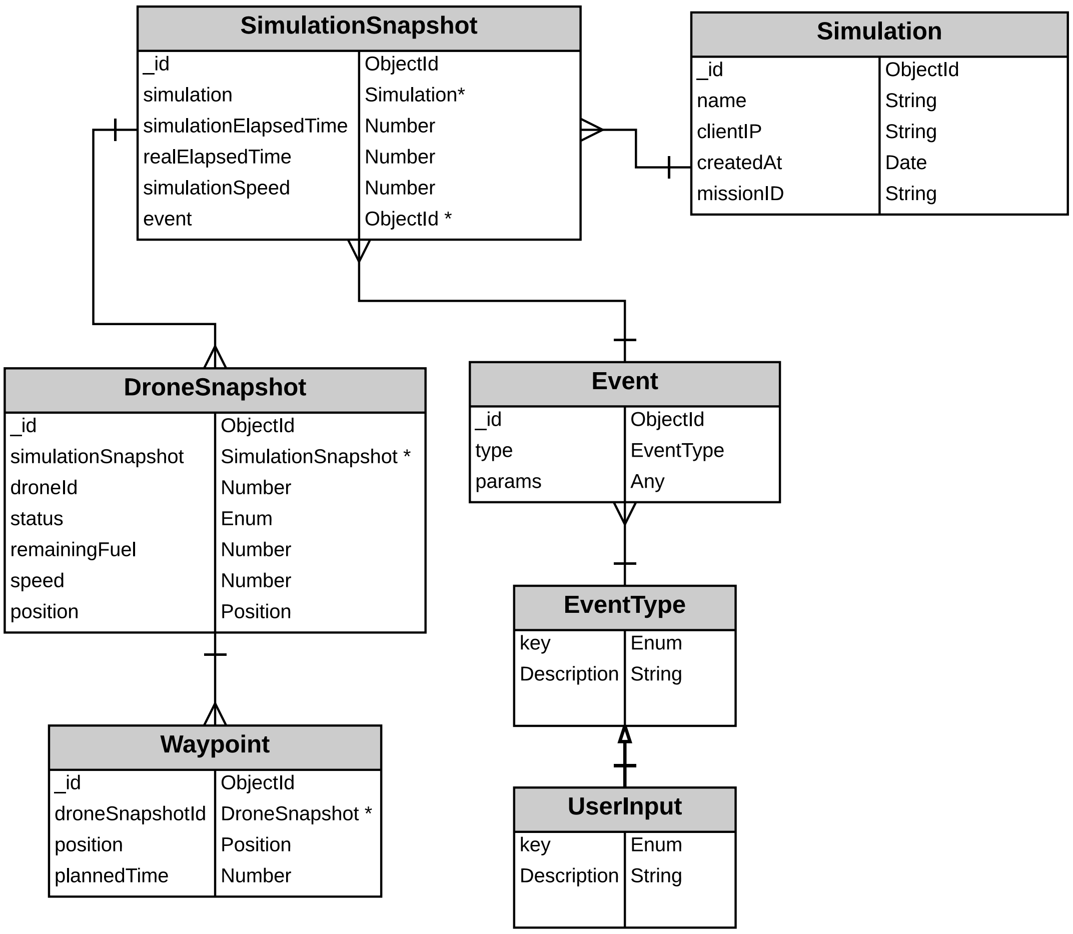


Figure 3.8: Output ERD. Entities and attributes are deeply detailed in Appendix A

Each of the events causing a Simulation Snapshot in DWR has an associated *Event Type*, indicating the reason why the event was generated. Appendix A contains the list of all possible *Event Types*. Some of the most relevant ones are:

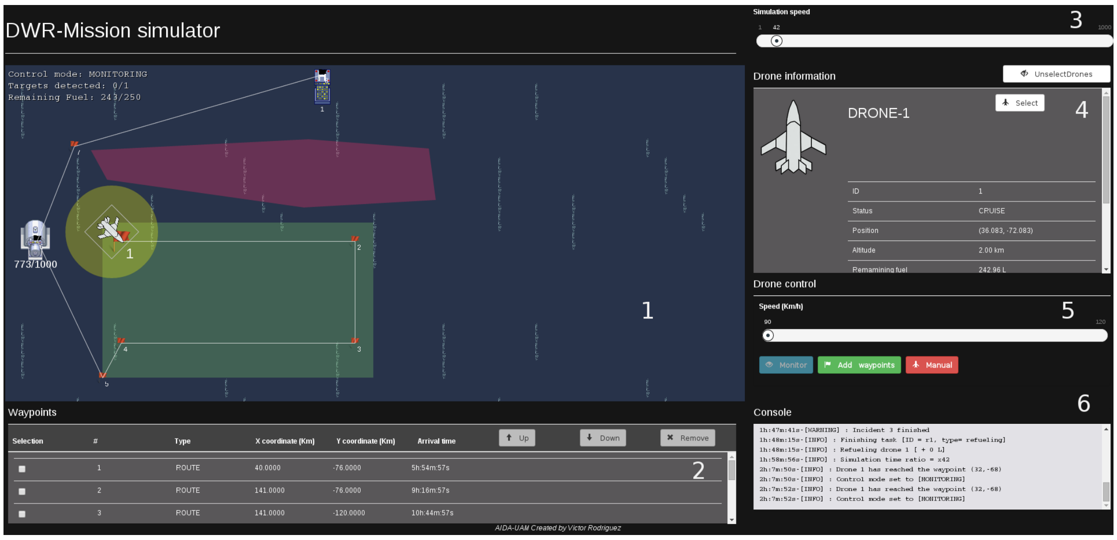
- A UAV reaches a waypoint.
- An incident starts/end.
- A target is detected.
- A UAV is destroyed.
- A UAV starts/finishes an action.
- The operator performs an interaction over the simulation. This is the most important Event Type in terms of analyzing operator's behavior, and therefore is modeled as a specific entity: *User Input* (See figure 3.8). The operator control interactions can be of several types, as detailed in Section 3.4.

Appendix A details deeper the attributes of each of the entities comprising the output data.

3.3 Graphical User Interface

One of the most important aspects consider in order to achieve a balance between usability and complexity in simulators is to design a good GUI, that arrange the mission information neatly and clarify what actions can be done.

Figure 3.9 shows a general screenshot of the execution of a simulation. It is appreciable that the GUI designed can be divided into multiple frames:



```
Control mode: MONITORING
Targets detected: 0/1
Remaining Fuel: 243/250
```

ID	1
Status	CRUISE
Position	(36.083, -72.083)
Altitude	2.00 km
Remamining fuel	242.96 L

Selection	#	Type	X coordinate (Km)	Y coordinate (Km)	Arrival time
☐	1	ROUTE	40.0000	-76.0000	5h:54m:57s
☐	2	ROUTE	141.0000	-76.0000	9h:16m:57s
☐	3	ROUTE	141.0000	-120.0000	10h:44m:57s

```
1h:47m:41s-[WARNING] : Incident 3 finished
1h:48m:15s-[INFO] : Finishing task [ID = r1, type= refueling]
1h:48m:15s-[INFO] : Refueling drone 1 [ + 0 L]
1h:58m:56s-[INFO] : Simulation time ratio = x42
2h:7m:50s-[INFO] : Drone 1 has reached the waypoint (32,-68)
2h:7m:50s-[INFO] : Control mode set to [MONITORING]
2h:7m:52s-[INFO] : Drone 1 has reached the waypoint (32,-68)
2h:7m:52s-[INFO] : Control mode set to [MONITORING]
```

FIGURE 3.9: Simulator screenshot. Numbers represent the different parts of the GUI

3.3.1 Main Screen

Displays graphically the Mission Scenario. It is the most important screen in terms of visualization and control of the simulation. Below are detailed the simulation elements represented in this screen.

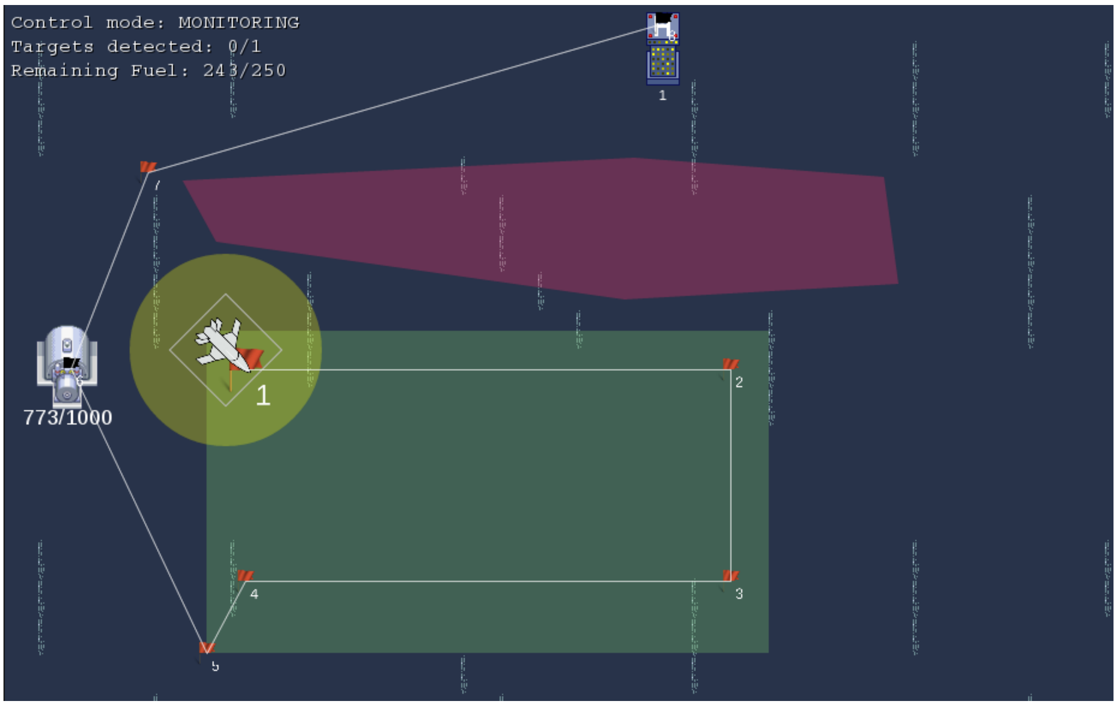

```
Control mode: MONITORING
Targets detected: 0/1
Remaining Fuel: 243/250
```

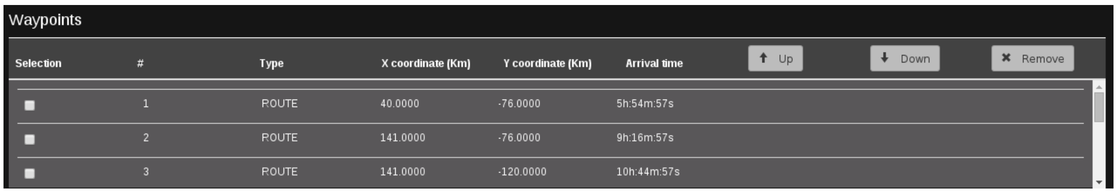
Figure 3.10: Simulator GUI screenshot - Main screen

- *UAVs*: Represented by a UAV icon, wrapped by a yellow circle marking its radar range. When a UAV is selected, the simulator displays its current flying path, as a sequence of flags (waypoints) connected by straight lines. The color of a flag indicates the waypoint type:
  - Red Flags: Route waypoints
  - Black Flags: Action waypoints (Refueling, Landing).
  - Task waypoints (Surveillance).
- *Task Zones*: Represented by a green polygonal area.
- *No Flight Zones*: Represented by a red polygonal area.
- *Airports*: Represented by a H heliport icon, together with a label indicating the name of the airport.

- *Refueling Station*: Represented by a station icon, together with a label indicating the duple *(remaining fuel/fuel capacity)*.

In addition, in the upper-left of the screen, some additional simulation information is displayed so that the operator can have a quick look at them: Current control mode (See controls in Section 3.4), targets detected (if any), and remaining fuel for the selected UAV.

3.3.2 Waypoints Panel



Selection	#	Type	X coordinate (Km)	Y coordinate (Km)	Arrival time
☐	1	ROUTE	40.0000	-76.0000	5h:54m:57s
☐	2	ROUTE	141.0000	-76.0000	9h:16m:57s
☐	3	ROUTE	141.0000	-120.0000	10h:44m:57s

FIGURE 3.11: Simulator GUI screenshot - Waypoints Panel

Displays a table with detailed information about flying path of the selected UAV. For each waypoint belonging to the UAV trajectory, the following attributes are showed:

- *Selection*: Shows whether this waypoint is currently selected or not (See controls in Section 3.4).
- *Order (#)*: Shows the order in which this waypoint will be reached.
- *Position*: Given in geodesic coordinates ($Latitude, Longitude$) or Cartesian coordinates (x, y).
- *Arrival time*: See 3.2.1.1.

3.3.3 Simulation Time Panel

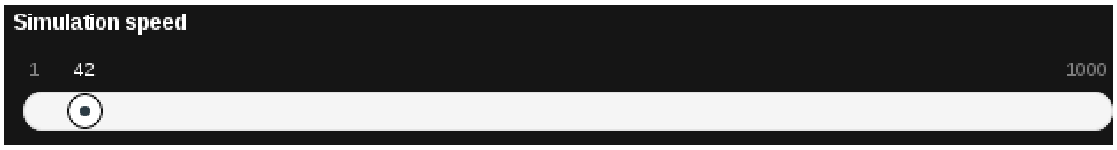
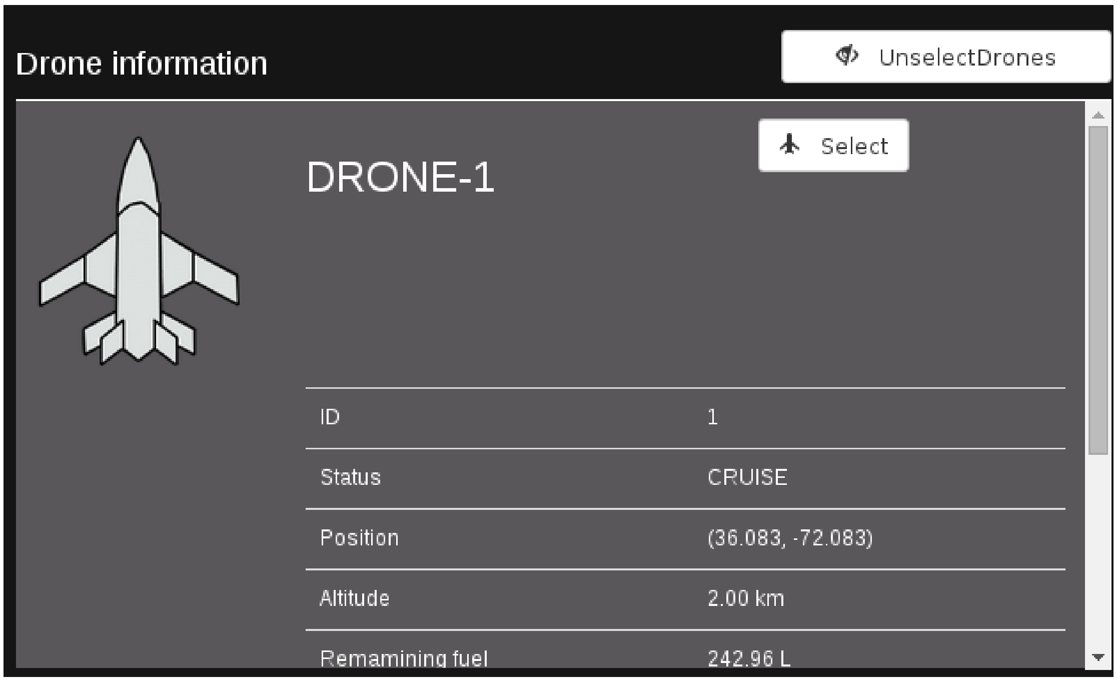


FIGURE 3.12: Simulator GUI screenshot - Simulation Time Panel

Displays a slide with information about the current simulation speed. The minimum assignable value is 1 ($SimulationTime = Realtime$) and the maximum is 1000 ($SimulationTime = 1000 * Realtime$).

3.3.4 UAV Information Panel



ID	1
Status	CRUISE
Position	(36.083, -72.083)
Altitude	2.00 km
Remamining fuel	242.96 L

FIGURE 3.13: Simulator GUI screenshot - UAV Information Panel

Displays a table with information about all UAVs participating in the mission. This information is updated continuously during the simulation execution. Below are detailed the attributes shown in this table:

- *Name* (DRONE-1 in Figure 3.13).
- *ID* (Alphanumeric).
- *Position*: A duple $(Latitude, Longitude)$ or (x, y).
- *Altitude*: Invariable during the simulation.
- *Remaining Fuel* (L): When it is less than 50 L, this label turns red to alert about it.
- *Fuel Capacity* (L)
- *Speed* (Km/h)

As detailed in the controls section (See 3.4), this panel also offers interactive controls to select/unselect a UAV.

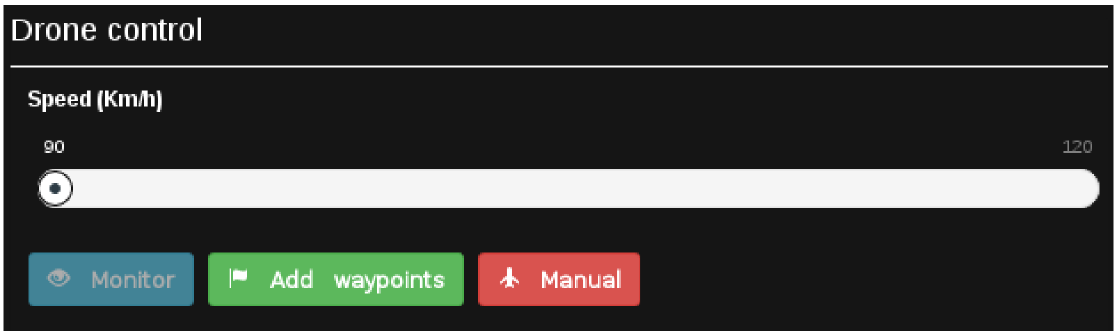
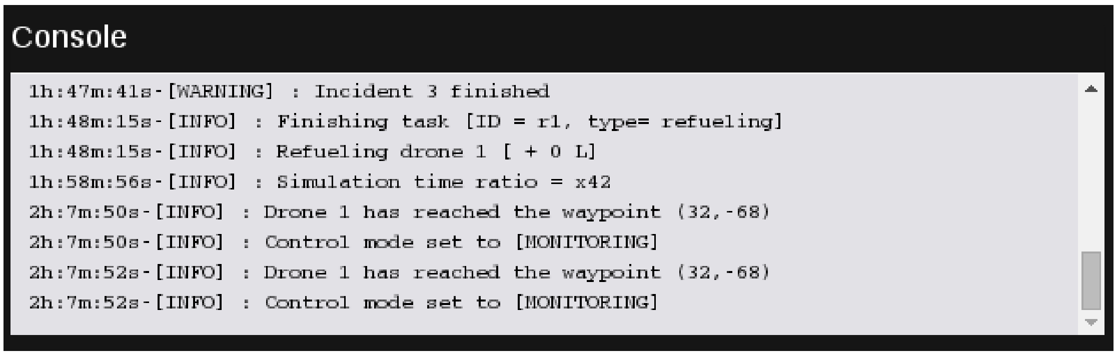


FIGURE 3.14: Simulator GUI screenshot - UAV Control Panel

3.3.5 UAV Control Panel

Displays the interactive widgets to control the selected UAV (if any selected). It features a slide to adjust the UAVs speed, bounded by the UAVs min/max speed, and three buttons to change the control mode, as detailed in the controls section (See 3.4).

3.3.6 Console Panel



```
1h:47m:41s-[WARNING] : Incident 3 finished
1h:48m:15s-[INFO] : Finishing task [ID = r1, type= refueling]
1h:48m:15s-[INFO] : Refueling drone 1 [ + 0 L]
1h:58m:56s-[INFO] : Simulation time ratio = x42
2h:7m:50s-[INFO] : Drone 1 has reached the waypoint (32,-68)
2h:7m:50s-[INFO] : Control mode set to [MONITORING]
2h:7m:52s-[INFO] : Drone 1 has reached the waypoint (32,-68)
2h:7m:52s-[INFO] : Control mode set to [MONITORING]
```

FIGURE 3.15: Simulator GUI screenshot - Console Panel

Logs every event that happens during the simulation (events and operator interactions). Each message is displayed together with the time it has been generated (measured from the beginning of the mission). Depending of the level of the message, it is colored differently. There are 4 types of message levels: Information (uncolored), Warning (Orange-colored), Error (Red-colored) and Success (Green-colored).

3.4 Controls

To achieve an intuitive and quick understanding of the different controls available in the simulator, almost all of them have been designed to be activated by doing mouse clicks on the *Main screen.* The interactions with the simulator can be divided into several groups.

3.4.1 Basic controls

These controls do not alter any aspect of the simulation. They are useful for the operator in order to monitor the simulation status easily. Table 3.1 lists all basic controls, and how to perform them.

3.4.2 Control Modes

In order to interact with the Main Screen of the GUI in multiple ways, 3 *control modes* are defined. Depending on the control mode chosen, the operator would be able to perform different interactions in that screen. Table 3.2 lists all possible control modes interactions, and how to activate them.

3.4.3 Waypoint controls

In terms of replanning a Mission Plan, the most important interactions to consider during a simulation are those that create or modify the flying path of a UAV. Table 3.3 details all possible interactions with waypoints offered by the simulator.

3.4.4 UAV controls

It is necessary to have control over the basic features of a UAV. Table 3.4 lists all possible interactions of this type offered by the simulator.

Table 3.1: Basic controls in DWR

Name	**Description**	**How to perform it**
Move Camera	Moves the camera along the Main Screen. This allows the operator to see every point in the mission map.	Arrow keys (Up, Down, Right, Down)
Select a UAV	At the beginning of the simulation run there is no UAV selected. Selecting a UAV allows the operator to monitor and control the UAV status and waypoints.	2 modes: • Left-click over a UAV in the main screen. • Left click on the Select button of a specific UAV in the UAV information panel.
Unselect UAVs	Unselect UAVs in order to clear the Main Screen vision and have a general overview of the simulation.	Left-click over the Unselect UAVs button in the UAV Information Panel.
Unselect UAVs	Unselect UAVs in order to clear the Main Screen vision and have a general overview of the simulation.	Left-click over the Unselect UAVs button in the UAV Information Panel.
Set simulation speed	Increase or decrease the simulation time speed. Normally, UAV missions last many hours, thus sometimes it is desirable to accelerate the process. The minimum simulation time speed is 1, what means that it is equal to the real time. The maximum value is 1000, which means that it is 1000 times higher.	Left-click over the slider on the Simulation Time Panel.

3.5 Architecture

DWR has been developed using modern **web development technologies** from the field of video games. The main advantages of these environments include:

- *Portability* of the developed application between both desktop and mobile systems.

TABLE 3.2: Control Mode interactions in DWR

Name	Description	How to perform it
Set control mode: Monitor	Sets the control mode of the selected UAV to Monitor. This is the default control mode and allows the operator to see and edit the UAVs waypoints, but not to add new waypoints. When a new UAV is selected or when a UAV reaches a waypoint, this control mode is set automatically.	Press the Monitor button in the UAV control panel. It is necessary to select a UAV before performing this interaction.
Set control mode: Add waypoints	Sets the control mode of the selected UAV to "Add waypoints". This control mode allows the operator to view and edit the UAVs waypoints, and also to add new waypoints **at the beginning** of the UAVs flying path, maintaining **the rest of the waypoints unchanged**.	Press the "Add waypoints" button in the UAV Control Panel. It is necessary to select a UAV before performing this interaction.
Set control mode: Manual	Sets the control mode of the selected UAV to Manual. This control mode allows the operator to define a new path, **deleting the previous one**.	Press the "Manual" button in the UAV control panel. It is necessary to select a UAV before performing this interaction.

- *High Accessibility*: Using any web browser with HTML5 capabilities, a user can access the URL where the simulator is hosted and use it without installing any additional software.

However, it is important to note the limitations of this type of technologies. The system requirements on a UAV simulator are much higher than those of a common web application, and current Javascript engines, despite being more and more powerful, yet have notorious performance troubles when running compute-intensive jobs. This simulator is an example of this kind of jobs in which the system usage must be taken into account.

Because of this, the simulator has been designed with a **2-level architecture (server-client)**, based on the design patterns used in the development of multi-user real time applications and video games. These design patterns divide the different components conforming the application engine (or videogame engine) between the two levels of the architecture [78].

Table 3.3: Waypoint interactions in DWR

Name	**Description**	**How to perform it**
Add a waypoint	Adds a new waypoint to the current UAVs path. It is necessary to select a UAV to perform this action. Depending on the control mode selected, this interaction change its behavior: • Control mode "Monitor": It is not possible to add waypoints. • Control mode "Add waypoints": The waypoints added do not change the previous UAVs flying path. • Control mode "Manual": The waypoints added define a new path.	With a UAV selected, and the appropriate control mode set, click any point in the Main Screen to add a waypoint in that position. Click anywhere over the sea to create *Route* waypoints. Click over a refueling station to create *Refueling* waypoints. Click over an airport to create *Land* (or *return*) waypoints.
Set waypoint position	Set a new position for an existing waypoint. It is necessary to select a UAV to perform this interaction. All the control modes are valid for this interaction.	With a UAV selected, **drag any waypoint** across the Main Screen.
Increase waypoint order	Increase by one the order of a waypoint. This interaction is not available for the first waypoint.	Select a specific waypoint by clicking its associated row in the *Waypoints Panel* and pressing the button Up.
Decrease waypoint order	Decrease by one the order of a waypoint. This interaction is not available for the last waypoint.	Select a specific waypoint by clicking its associated row in the *Waypoints Panel* and press the Down button of the same table.
Remove waypoint	Remove a waypoint from the current UAVs path.	Select a specific waypoint by clicking its associated row in the *Waypoints Panel* and by pressing the Remove button of the same table.

Table 3.4: UAV interactions in DWR

Name	**Description**	**How to perform it**
Set UAVs speed	Set the UAV speed. The new speed value must be between the UAV minimum and maximum speed values. It is necessary to select a UAV to perform this action.	Mouse click over the slider on the UAV control panel.

The **server** contains the logical core of the simulator. It is responsible for initializing a new simulation every time a client asks for it, retrieve and process the mission data loaded by the Data Entry Module (See Section 3.5.2), maintain and update the simulation status, send it to the client periodically, process the user commands and store the relevant information and events happened during the simulation time. The framework used to implement the server is *NodeJS* [79], written in Javascript and highly supported by the open source community.

The **client** (web browser) receives the simulation status sent by the server and is responsible for showing it on screen in real time. It is completely unconscious of the core logic running in the server, and only knows how to display, graphically or textually, each of the elements that compose the simulation status, according to the GUI described in 3.3. Moreover, the client is also responsible for catching the user interactions (keyboard and mouse inputs), and sending the corresponding control commands to the server, that will process them and change the simulation status appropriately. The client has been implemented using multiple web-development frameworks: *Phaser* [80], to build the main screen, and *AngularJS* [81], to manage the information and control panels (See GUI description in 3.3).

Client-Server communication is achieved by the use of the *Websockets* communication protocol [82], which offers lower latency than HTTP, and is specially suitable for real time data streams [83]. From a functional point of view, the architecture of this simulator can be divided into 5 distinct modules, as shown in figure 3.16. Below are described each of the 5 modules comprising DWR architecture.

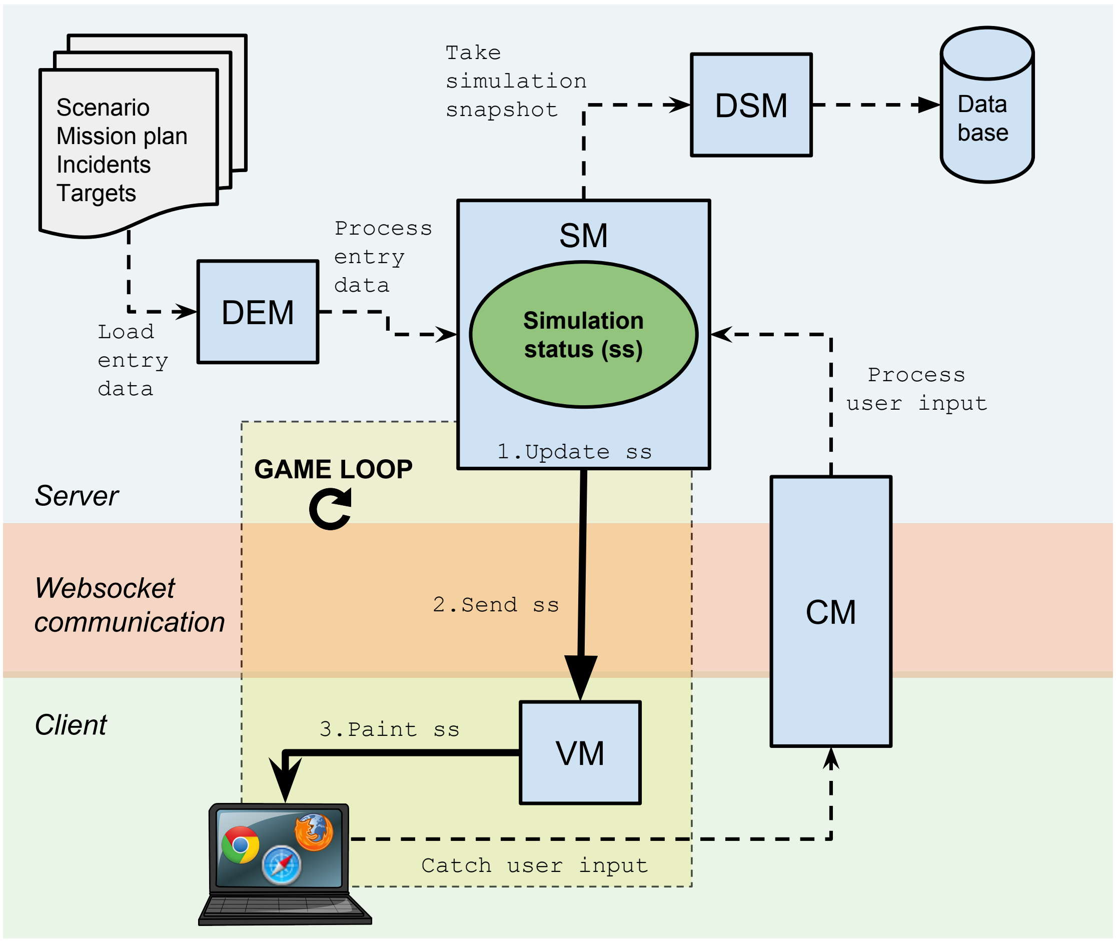


FIGURE 3.16: Architecture diagram of the designed simulator. Each of the five distinct modules (boxes) is located in its corresponding architecture level (client-server).

3.5.1 Simulation Module (SM)

The Simulation Module (SM) represents the functional core of the simulator and all the elements that compose it. It runs entirely on the server, and it is responsible for managing the Simulation Status and sending it to the client periodically. All simulation elements described in 3.2 are managed by this module.

3.5.2 Data Entry Module (DEM)

The *Data Entry Module* is responsible for loading into the simulator all data necessary to start the simulation of a mission (See data input in 3.2.2). It runs entirely on the server side, called by the *Simulation Module* whenever an user connects to the simulator web page.

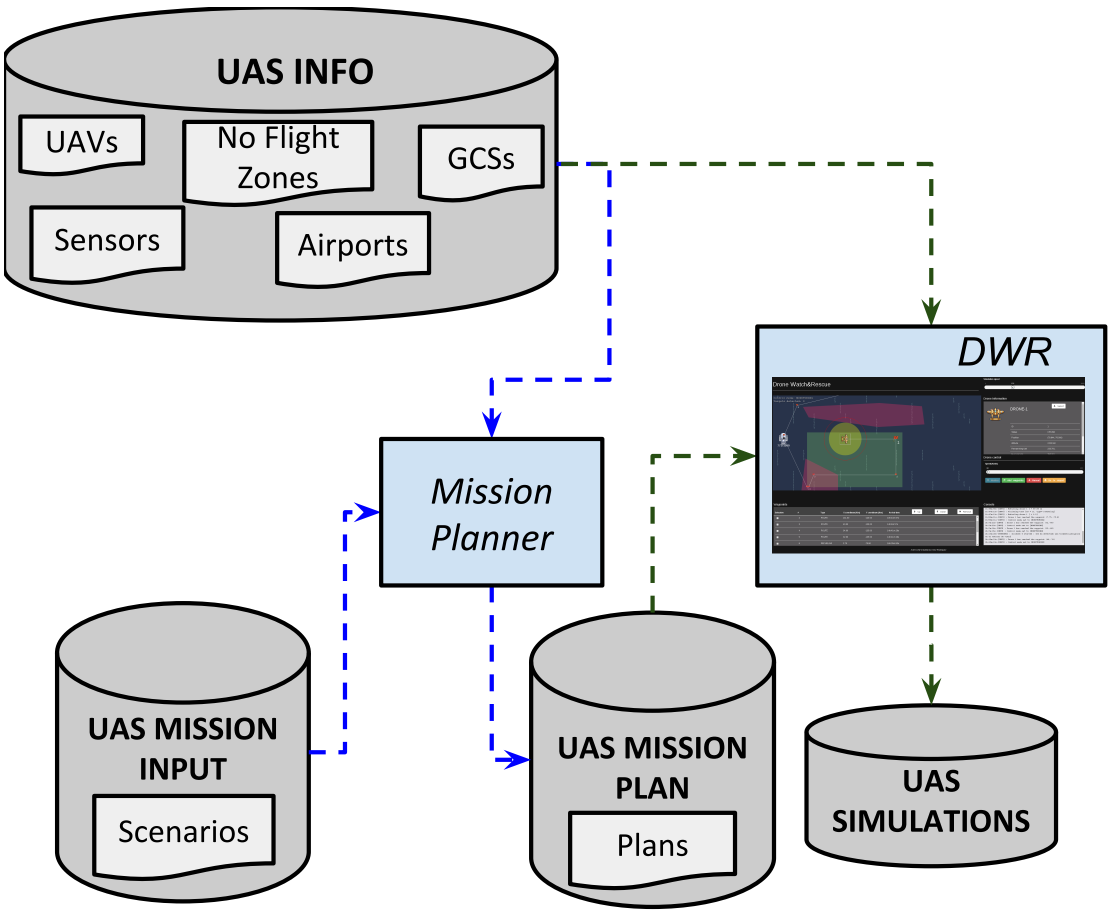


FIGURE 3.17: General data flow (input/output) among the developed simulator (DWR), and the Mission Planner utilized.

Although this module can load some *test missions* directly from Javascript Object Notation (JSON) files, normally the data source will be a set of databases implemented using the Database Management System (DBMS) MongoDB [84]. These databases are the same that uses the Mission Planner developed by Ramirez-Atencia et al. in [72–74], and contain data concerned to the elements that compound a typical UAS. As shown in figure 3.17, there are many databases from which the Data Entry Module loads the input data:

- *UAS Info*: Contains the static information about the environment, which is shared shared by all missions (No flight Zones, UAVs main features, Airports...).
- *UAS Mission Input*: Contains Mission Scenarios (See 3.2.2.1).
- *UAS Mission Plan*: Contains Mission Plans obtained by the Mission Planner (See 3.2.2.2).

3.5.3 Visualization Module (VM)

The *Visualization Module* is responsible for displaying, both graphically and textually, the Simulation Status in real time. It runs entirely on the client side of the architecture (i.e, the web browser). None of the simulation logic elements are contained in this module, its behavior only consists in receiving the Simulation Status periodically (sent by the Simulation Module) and displaying each of its elements in a proper screen or panel. The layout designed for visualizing the simulation data is detailed in the GUI section (See 3.3).

3.5.4 Control Module (CM)

The *Control Module* catches, sends and processes the different interactions that an operator performs during a simulation run. In accordance with the developed two-level game architecture, this module establish a link between the two levels of the architecture, since it catches the operator inputs (keyboard and mouse events) on the client side, transmits them (via *Websockets*), and finally processes them on the server side, changing the simulation status appropriately. Section 3.4 details the different interactions recognized by this module.

3.5.5 Data Storage Module (DSM)

The *Data Storage Module* (DSM) is responsible for storing all the necessary data from the execution of a simulation. This module is essential for the purposes of this work, since the data sets retrieved here serve as a starting point for the experimentation made in the next chapter. To store the data, a MongoDB database is used (See Figure 3.17, *UAS SIMULATIONS*). The data stored is event-organized as described in 3.2.3. Due to this, this module works asynchronously as an **event listener**. Whenever an event occurs during the simulation run, the DSM stores the simulation status in that moment, taking a *Simulation Snapshot*. As was discussed before, the data obtained this way represents a simulation robustly.

Chapter 4

Experimentation and Analysis of Simulation Data

The purpose of this chapter is to show the experiments carried out using the simulator DWR, in order to prove that this platform, and specially the data extracted from it, is suitable to analyze the behavior and performance of UAV operators during a training session.

First of all, the simulator will pass a load test to ensure that it is able to maintain multiple users simultaneously. Then, the real experiments made for extracting data from the simulator will be introduced, detailing the data source and the processes followed to obtain it. Once we have a robust dataset, we will explain how the performance of a user is evaluated and, based on this evaluation, how we create and group user profiles in order to create **clusters** that indicate similar user behaviors. Finally, those clusters will be analyzed and interpreted in the context of this experiment.

4.1 Software load test

The **web architecture** described in the previous chapter (See 3.5), and used to implement DWR, is the basis for allowing a *low-cost multi-UAV simulation environment*, where operators, and more generally, users, are able to train their monitoring, planning and replanning skills without the need for a powerful machine or a specific software installation (just a web-browser).

To ensure the proper functionality and scalability of the designed architecture, it is necessary to carry out some load tests in each of the architecture levels (server, client, communication). To test the client, some basic experiments have been made in order to show that client requirements are very low, hence any modern computer can run and visualize a simulation correctly. The simulator has been executed (i.e, accessed via web browser) using different platforms (desktop and laptops), operative systems (Windows, Linux, MacOS...) and HTML5-browsers (Chrome, Safari, Firefox, Internet Explorer). All of them have run all the test missions designed for this experimentation (See 4.2), performing a constant 30 Frames Per Second (FPS) rate. Smart-phones, tablets and other mobile devices cannot execute the developed simulator.

In this section we focus on testing the **server** side of the architecture, evaluating its capacity to host multiple simulations simultaneously. The server host machine, whose specifications are showed in Table 4.1, is responsible for both serving the simulator web GUI to the client (doing the works of a *web server*), and for maintaining a real-time communication with it, sending the simulation status periodically. It also contains the database that will store the simulation data retrieved by the Data Storage Module (See 3.5.5).

Table 4.1: Specifications of the server host machine used for the load test.

Parameter	**Description**
CPU	AMD64 - 4 cores (1.6 GHz)
RAM	8 GB
OS	Linux (Debian)

Instead of using real users for analyzing the server load, an automatic test-bed has been created. This test-bed connects to the server, every 2 seconds, a new *virtual player* to a total of 2000 players. Every time a new connection is received, the server begins to load, process, and send data of a new simulation. Virtual players behave *passively*: They keep the connection with the simulator opened, but they do not send control commands to interact with it. This is the behavior that an operator would have if he only monitors and observes a mission, without performing any interaction.

The mission loaded as input data for every simulation in this load test is the same. A brief summary of the mission parameters is shown in Table 4.2. As it can be appreciated, the complexity and load of this mission is low, since it only features one UAV performing a unique Surveillance task to watch one target.

Table 4.2: Summary of the mission parameters used in the load test.

Parameter	**Value**
Map dimensions	$440x160Km$
# UAVs	1
# Targets	1
# Incidents	2
# Surveillance Tasks	1
# Planned waypoints	11
# No flight Zones	1
# Refueling Stations	1
# Planned Refueling Actions	2

In order to analyze the server, a **performance profiler** is used. This profiler measures the status of several server components at one minute intervals. In this test, the following metrics are taken into account:

1. *Number of players*: This metric is taken from the amount of simultaneous connections kept by the server at a specific time (See Figure 4.1, players).
2. *CPU Time*: It measures, in milliseconds, how much CPU time has been consumed by the server during the minute in which this metric was taken. The maximum possible value is 60000 ms, which means that the server has consumed the whole CPU time during a specific minute (See Figure 4.1, CPU time).
3. *Use of Memory*: It measures, in MB, the amount of RAM used by the server during the minute in which the metric was taken (See Figure 4.1, memory).

The results of this test are shown graphically in Figure 4.1. From these results, three **server states** can be distinguished:

1. *Light Load State*: In this state, the server *use of memory* keeps **constantly low**, because the amount of data processed in each of the connections (simulations) is not too high. However, increasing the number of players affects directly to the

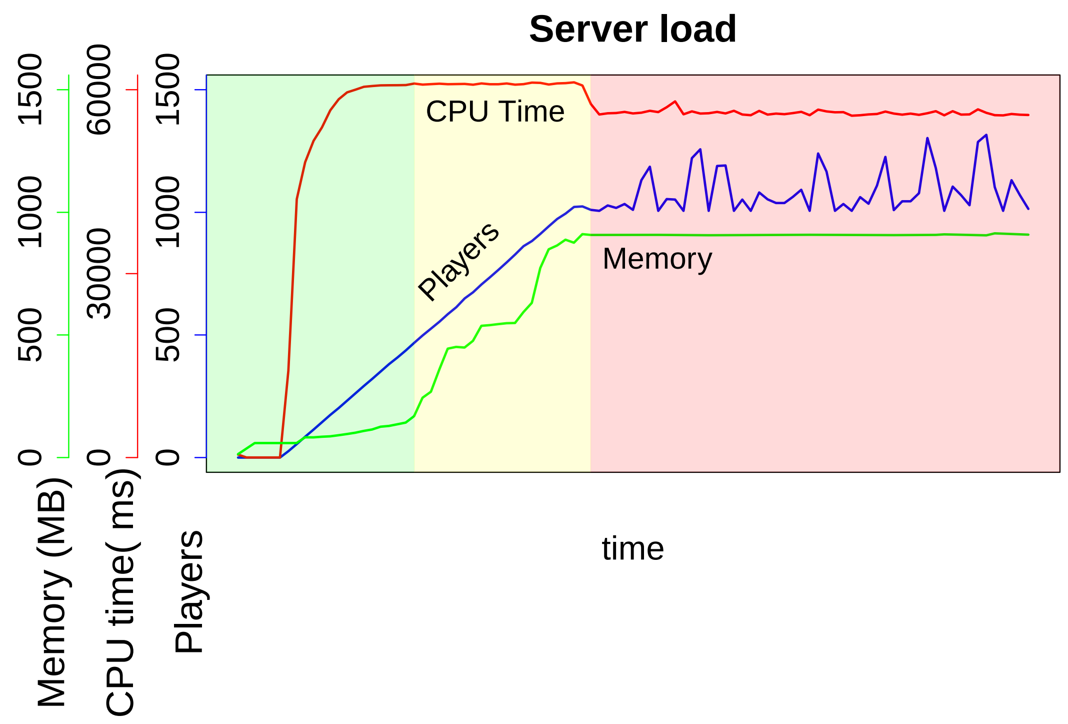


FIGURE 4.1: Metric comparison for analyzing the server load versus the number of users connected to the simulator DWR

CPU time metric, which reaches its maximum value (60000 *ms*) shortly before connecting 500 players (See Figure 4.1, green area).

2. *Heavy Load State*: When the CPU time metric reaches its maximum value, the arrival of new players (connections) **requires an increasing in the use of memory** in order to host, load an process the simulation of each connection. During this state, the process of updating the simulation status begins to suffer some delays, causing an increase in the response time to the client (See Figure 4.1, yellow area).

3. *Saturation State*: When the server hosts around 1000 connected virtual players, the CPU collapses and the large increase on the use of memory result in an excessive delay in the computing of each simulation, and an inability to maintain some player connections. Therefore, as it can be appreciated, some connections are closed, and the server response time is too high to offer a real-time communication (See Figure 4.1, red area).

The three server states obtained by this load test suggest that it would be necessary to enable a **second server** to host simulations when the amount of users connected simultaneously gets close to 1000 users. Thanks to the high portability of the designed architecture, this could be done quite easily. However, in the experiments carried out in this work, and described in the following sections, a unique server was enough to host all users correctly.

4.2 Experimental dataset

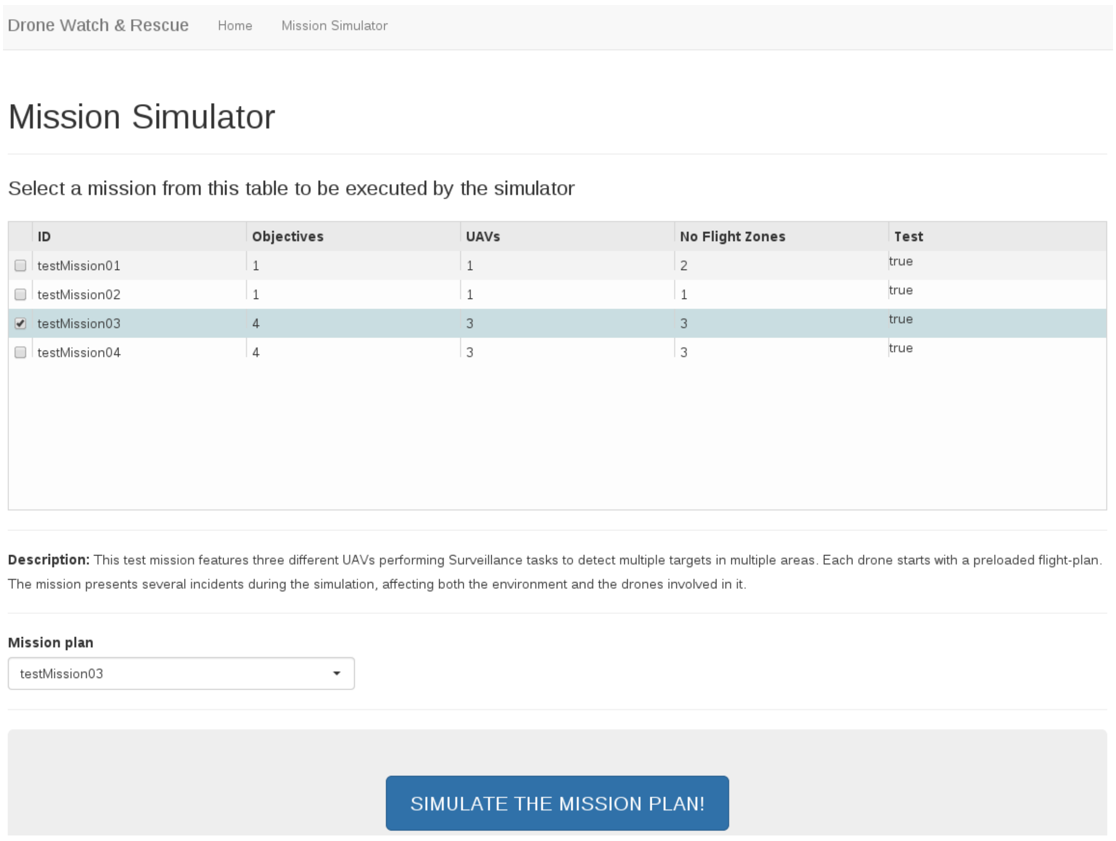
Once we have ensured that the designed simulator architecture is valid for being used by hundreds of users simultaneously, the next step is to use the simulator with real users, in order to extract knowledge about their behavior in a multi-UAV simulation environment.

For this purpose, the simulator was deployed into a server located at Autonomous University of Madrid (UAM), with the system specifications detailed in Table 4.1. The *testers* of the simulator were Computer Engineering students of the same university (UAM), all of them coursing the last year of the degree. Although the experiment was conducted in two different days, all users received the same tutorial before using the simulator, so, a priori, it makes no sense to distinguish the students by the day when they were tested. All data extracted during these two days is therefore treated uniformly.

The experiment was conducted as follows: for each of the days, the students (testers) were given a brief explanation about: the basic concepts of multi-UAV missions, the goal they had to reach using the simulator, and the list of all possible interactions they were able to perform during a simulation. At the same time, since each student had a personal computer, they were asked to access the simulator web page and follow the tutorial by their own, to get a better understanding of the GUI and the controls. When the tutorial ended, they were asked to use the simulator freely during 30 minutes approximately.

When a user entered the simulator web page, he did not start a simulation immediately but was prompted with a *mission selection screen*, as shown in Figure 4.2.

This screen shows, for each available mission, a brief summary of the mission content (Number of targets, UAVs, no flight zones...) and a little description explaining the goal of the mission. Table 4.3 summarizes the main features for each of the **4 test missions**


Drone Watch & Rescue Home Mission Simulator

Mission Simulator

Select a mission from this table to be executed by the simulator

	ID	Objectives	UAVs	No Flight Zones	Test
☐	testMission01	1	1	2	true
☐	testMission02	1	1	1	true
✓	testMission03	4	3	3	true
☐	testMission04	4	3	3	true

Description: This test mission features three different UAVs performing Surveillance tasks to detect multiple targets in multiple areas. Each drone starts with a preloaded flight-plan. The mission presents several incidents during the simulation, affecting both the environment and the drones involved in it.

Mission plan

testMission03

FIGURE 4.2: Mission Selection Screen prompted to the students taking part in the experiment conducted for this work

designed for this experiment. As it can be appreciated, there is an increasing order in terms of the challenge that suppose a mission:

1. *TestMission01*: This test mission features one UAV performing one Surveillance task. It starts with a pre-loaded Mission Plan and presents several incidents during the simulation (See Figure 4.3).

2. *TestMission02*: This mission is similar to *TestMission01* except for the Mission Scenario (See Figure 4.4).

3. *TestMission03*: This test mission features three different UAVs performing Surveillance tasks to detect multiple targets in multiple areas. Each UAV begins with a preloaded flight-plan. The mission presents several incidents during the simulation, affecting both the environment and the UAVs involved in it (See Figure 4.5).

4. *TestMission04*: This test mission loads exactly the same Mission Scenario, Incidents Plan and Targets Definition as *TestMission03*. The main difference lies

Table 4.3: Specification summary for the test missions (T.M) designed in the experiment.

	T.M.01	**T.M.02**	**T.M.03**	**T.M.04**
ID	0	1	3	4
Map extension (Km)	440x160	430x500	800x500	800x500
UAVs	1	1	3	3
Tasks	1	1	4	0
Targets	1	1	4	4
Incidents	2	2	4	4
No Flight Zones	1	2	4	4
Refueling Stations	1	3	4	4

in the fact that **there is no pre-loaded Mission Plan, hence the operator must plan each UAV manually before starting the simulation** (See Figure 4.5).

The dataset resulted of extracting data from this experiments is composed of **127 distinct simulations**, played by a total of **27 users**. This is a great amount of data if we take into account that the conducted experiments were simple and relatively short in terms of duration. This fact proves the potential of the developed simulator when trying to collect data massively.

Before starting the user performance analysis, it is interesting to analyze if the users have generally used the simulator progressively. To do this, we must check the order in which the students have run the four available test missions. Figure 4.6 shows the evolution of the 27 users in the dataset. Each user is identified by the day when he participated in the experiment (05 or 07, corresponding to the fifth and seventh of November) plus the three last digits of the *IP address* from the computer he/she was using.

As can be appreciated, the general trend for the users that have run more than one simulation is to go from the first two missions (TestMission 01,02), which served as a tutorial during the experiment, to the last ones (TestMission 03-04), which were played when the students used the simulator freely.

In order to achieve a robust analysis of the data extracted, we must clean the dataset by removing those simulations which can be considered as **useless**. Since the simulator

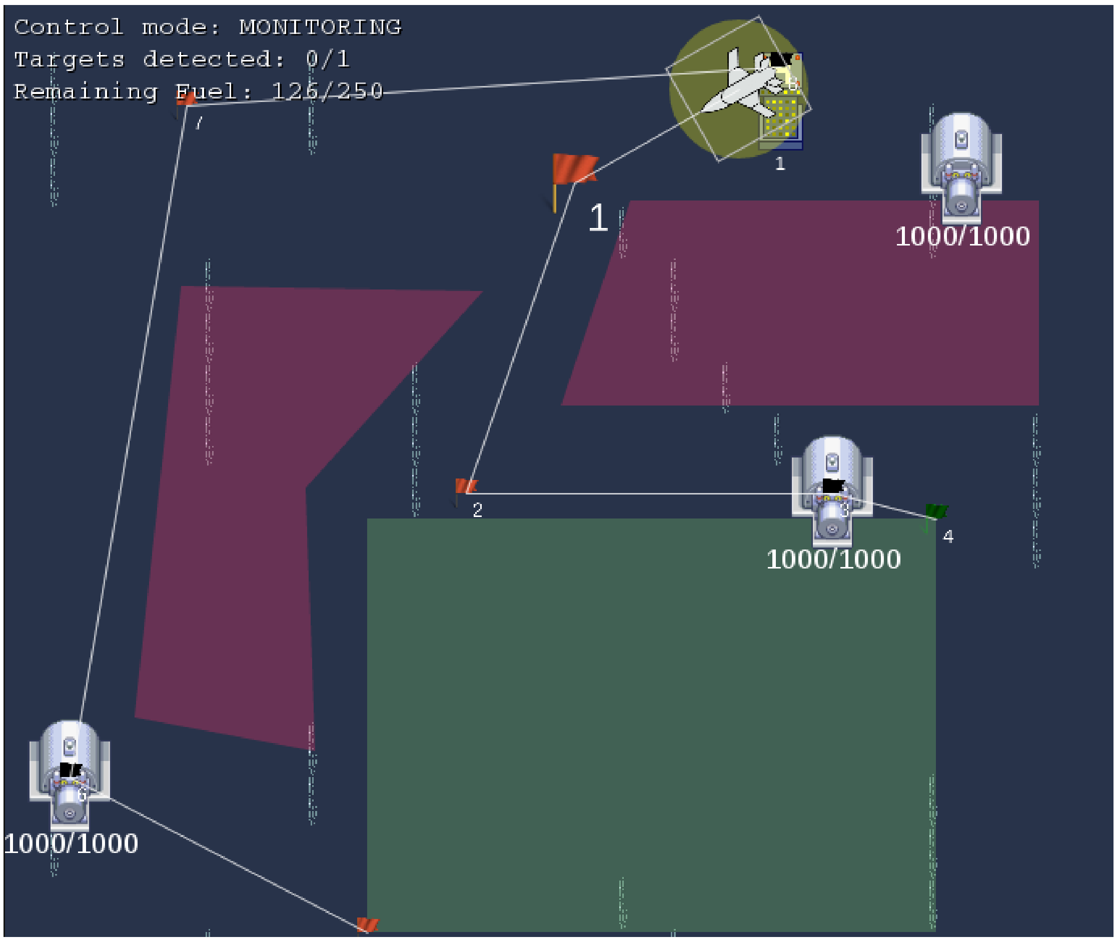


FIGURE 4.3: Test Mission 01 screenshot.

is running in a *web-environment*, a user can "restart" (or abort) a mission simulation by doing a *page refresh* in his web browser. Due to that, we must remove from the dataset those simulation which have been **aborted prematurely**. In this work, we consider that a simulation is useless if it has been aborted before **20 seconds**. From the 127 simulations composing our students dataset, only 102 of them are considered useful simulations, and will be used in the data analysis process.

4.3 User performance metrics

The main goal of this experimentation is to analyze the user performance when running missions in the designed simulator. That lead us to the necessity of defining a way to measure the performance of a user in a specific simulation.

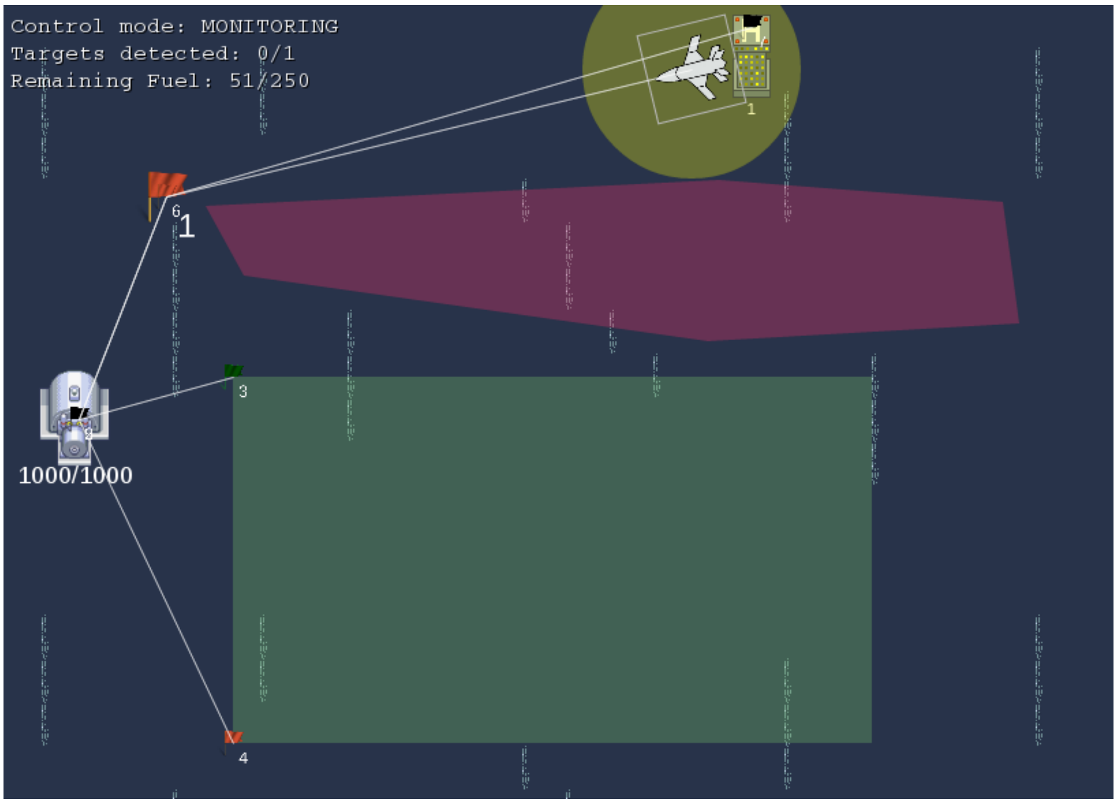


FIGURE 4.4: Test Mission 02 screenshot.

To achieve this, five performance metrics have been defined: Agility (A), Consumption (C), Score (S), Attention (At) and Precision (P). All of them are numeric values in the range $[0, 1]$, where 0 represents the worst performance for that metric, and 1 represents the best. It is remarkable that the metrics have been designed so that none of them is dependent of some other. Below are detailed the implementation of each of the metrics.

4.3.1 Agility

Agility (A) measures the average speed with which the user has interacted with the simulator. All interactions defined in Section 3.4 are taken into account. Let $I(s)$ the set of interactions performed during a given simulation s, the Agility metric is computed as:

$$A(s) = \frac{\sum_{i \in I(s)} \frac{simulationSpeed(i)}{MAX\_SPEED}}{|I(s)|} \tag{4.1}$$

where $MAX\_SPEED = 1000$ and $simulationSpeed(i)$ gives the speed in which the simulation was running in the moment when the interaction i was made. As was explained in Section 3.4, the user can manipulate the simulation speed using a slider, giving values

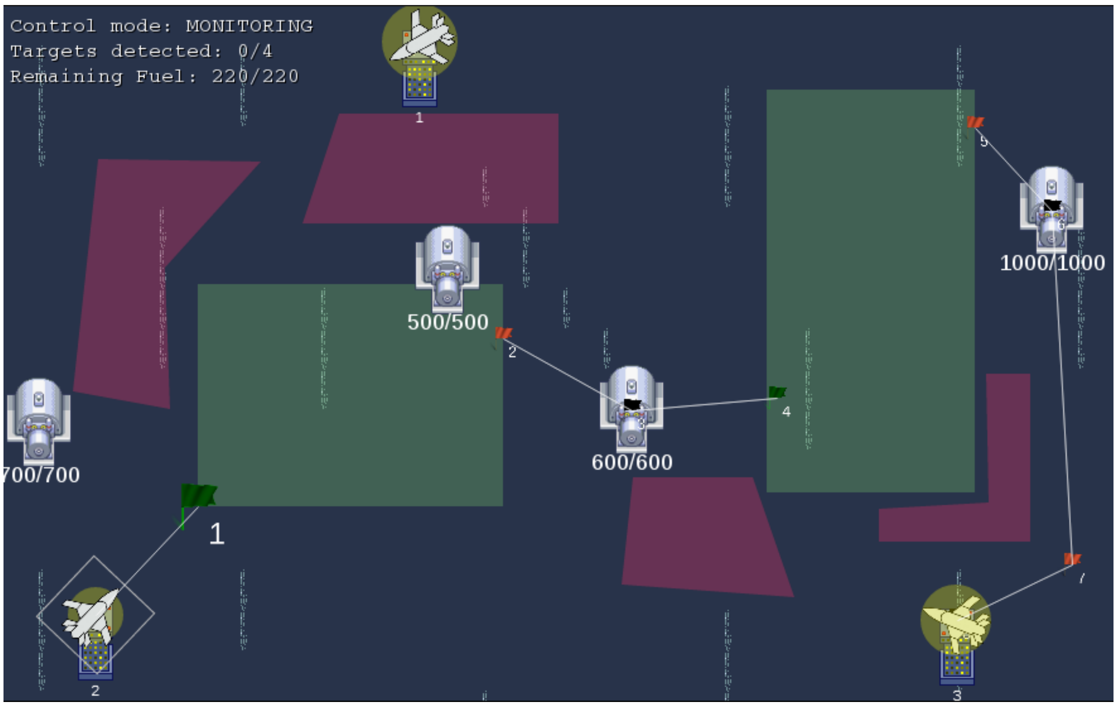


FIGURE 4.5: Test Mission 03-04 screenshot. Note that these two missions share the same Mission Scenario

from 1 to 1000. A user is considered agile if he can interact when things are happening fast.

Figure 4.7 shows the distribution of this metric on the students dataset (removing useless simulations). As can be seen, **the metric tends to obtain low values in this dataset**, which means that the students have usually run the simulator slowly and carefully. This makes sense when taking into account the fact that the students are novice users, using the simulator for the first time. Therefore, when analyzing user performances, a high agility value (higher than the mean, which is 0.29) should be considered as discriminating.

4.3.2 Consumption

The *Consumption* metric measures the fuel consumed throughout the simulation time. Given a specific instant (also called *snapshot*) sh of a simulation s, we can compute the global remaining fuel (rf) at that instant as

$$rf(s, sh) = \sum_{u \in U(s)} rf(u, sh) + \sum_{r \in R(s)} rf(r, sh) \quad , \tag{4.2}$$

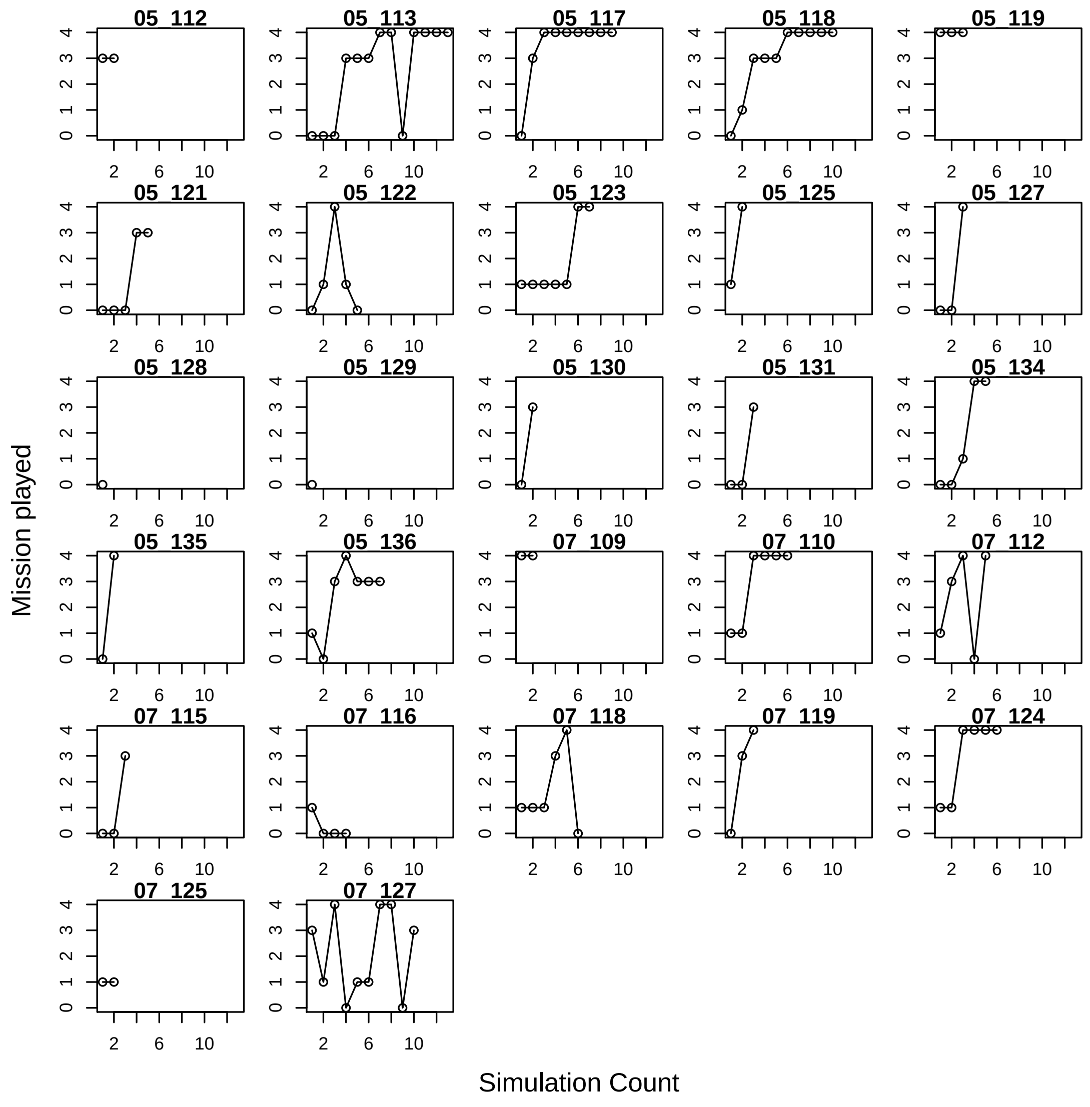


FIGURE 4.6: Evolution of the test missions played per user. *ID(TM01)=0, ID(TM02)=1, ID(TM03)=3, ID(TM04)=4*

where $U(s)$ is the set of UAVs participating in the simulation s and $R(s)$ is the set of refueling stations taking part in the Mission Scenario of simulation s. The remaining fuel value for both UAVs and refueling stations can be retrieved from the information contained in the simulation snapshots taken during a simulation. When a UAV u is destroyed during the simulation, it is considered that $rf(u, sh) = 0$ for every instant sh after the UAV destruction.

To calculate the consumption over a simulation s, we compare the remaining fuel at the end of the simulation (last snapshot, or *lSh*) with that at the beginning (first snapshot, or *fSh*):

$$C(s) = \frac{rf(s, lSh(s))}{rf(s, fSh(s))} \tag{4.3}$$

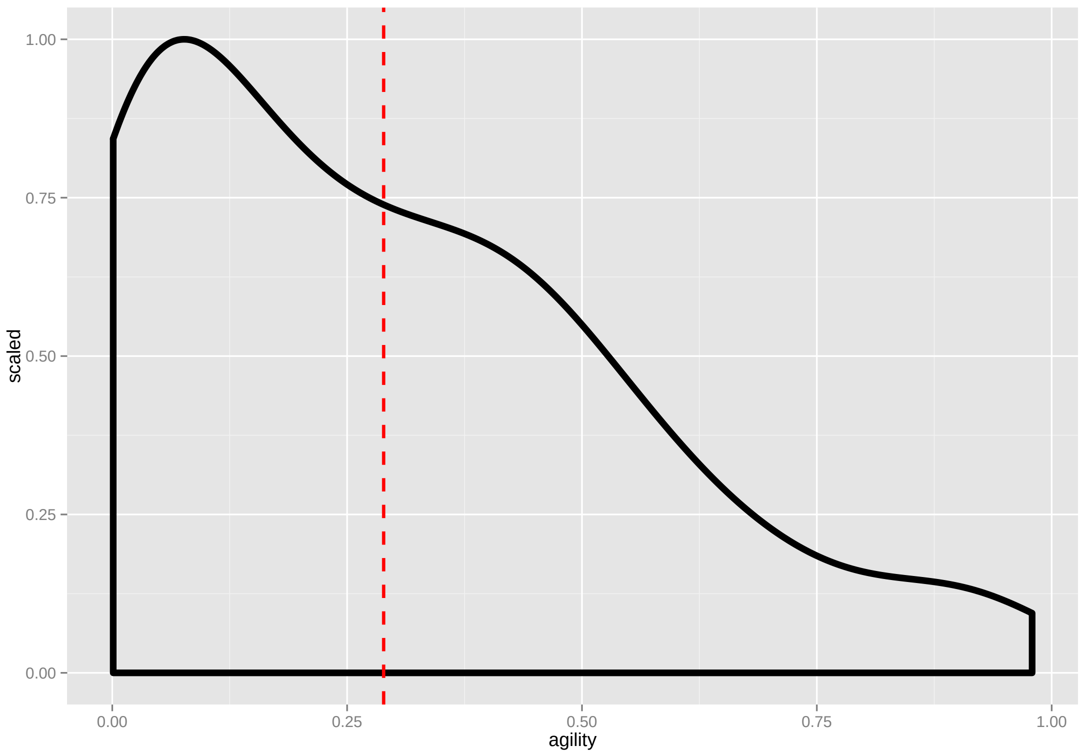


FIGURE 4.7: Density distribution for the metric *Agility*. The dashed line represents the mean value.

High values of this metric indicate that the remaining fuel at the end of the mission is high, so the consumption is considered low. On the other hand, low values mean high consumption rate.

Unlike the case of the Agility metric, Figure 4.8 shows how the distribution of the Consumption metric for the dataset used in this experiment tends to concentrate on very high values, obtaining a mean value of 0.848. This could lead us to conclude that the users usually make a good use of the resources available in a mission, but this seems confusing providing that the dataset is entirely composed of novice users in the field.

If we analyze deeper the consumption equation defined in Equation 4.3, we see that the value obtained is **inversely proportional to the duration of the simulation**. Short missions will likely obtain high consumption rates, and viceversa. Since a simulation can be aborted at any time by the user, we can conclude that this metric distribution is really indicating that **users in this dataset have generally run short simulations**, probably because they were inside a *trial and error* process in which the mission objectives were not taken into account.

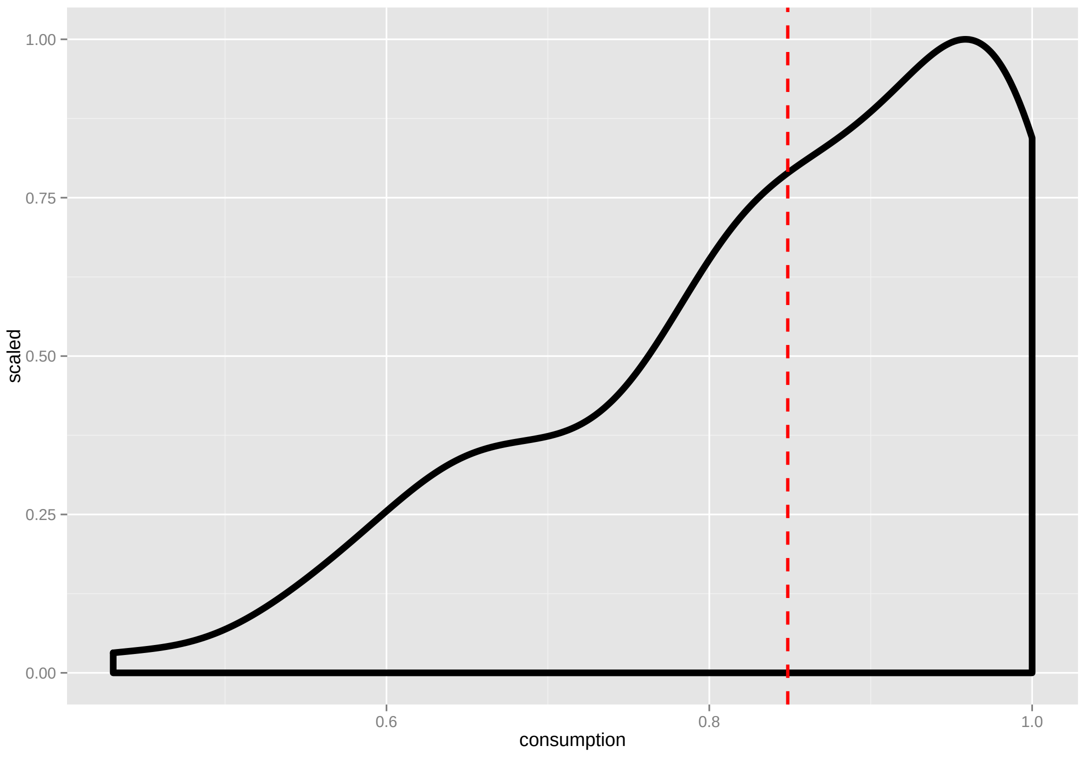


FIGURE 4.8: Density distribution for the *Consumption* metric. The dashed line represents the mean value.

However, in terms of the general information that a variable is potentially able to offer for ranking and grouping users, this metrics performs the worst values of all, since it features an extremely low variance (0.01812) and a short range of values ($[0.43, 1.00]$).

4.3.3 Score

The *Score* (S) metric gives a global success/failure rate of a simulation. As was discussed in Section 3.2, the main goal for a user (operator) monitoring a simulation in DWR is to capture the maximum number of targets, minimizing the resources consumed and returning all UAVs to an airport at the end of the mission. Given that, we can divide the general goal into 3 sub-goals:

1. Detecting targets.
2. Minimizing resource loss.
3. Returning the UAVs to an airport to finish the mission.

Based on this description, we define the score of a simulation s as:

$$S(s) = \frac{1}{3}\left[\frac{|targetsDetected(s)|}{|T(s)|} + \left(1 - \frac{|destroyedUAVs(s)|}{|U(s)|}\right) + \frac{UAVsInBase(s, lSh(s))}{|U(s)|}\right], \tag{4.4}$$

where $U(s)$ is the set of UAVs participating in the mission and $T(s)$ the set of mission targets. Note that $UAVsInBase(s, lSh(s))$ queries how many UAVs were positioned on an airport at the last instant (last snapshot *lSh*) of the simulation.

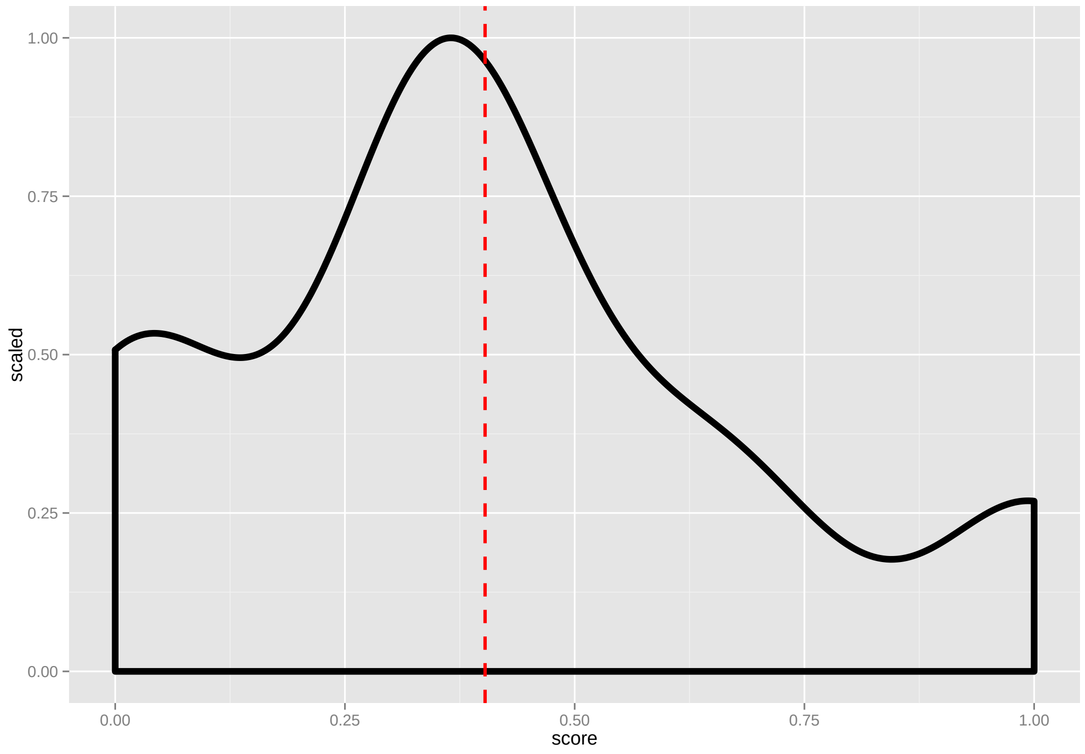


FIGURE 4.9: Density distribution for the *Score* metric. The dashed line represents the mean value.

Figure 4.9 proves that the Score metric obtains a well-balanced density distribution for our dataset. This means that there is not a general tendency for this metric, and therefore it can be used robustly to compare the user performance in a general way.

TABLE 4.4: Statistics summary for the metrics defined in the analysis.

Metric	**Min.**	**1st Qu.**	**Median**	**Mean**	**3rd Qu.**	**Max.**	**Var.**
Agility	0.00100	0.06777	0.24560	0.28890	0.44170	0.97920	0.06359
Consumption	0.4307	0.7847	0.8728	0.8486	0.9620	1.0000	0.01812
Score	0.0000	0.1806	0.3333	0.4025	0.5556	1.0000	0.08684
Attention	0.2929	0.7226	0.8143	0.7657	0.8737	0.9219	0.02157
Precision (Missions 1-2-3)	0.02083	0.04762	0.18330	0.36620	0.50000	1.00000	0.13691
Precision (Mission 4)	0.01351	0.01887	0.03333	0.16670	0.29170	1.00000	0.07215

4.3.4 Attention

The *Attention* (At) metric rates globally the user intensity in terms of the interactions he has performed during a simulation. Given a simulation s, the Attention is defined as:

$$At(s) = 1 - \frac{1}{1 + \sqrt{|I(s)|}} \tag{4.5}$$

where $I(s)$ is the set of all interactions performed during simulation s. A square root is introduced in the equation in order to avoid a fast convergence to 1.

The density distribution of the Attention metric, showed in Figure 4.10, shows a general tendency very close the Consumption distribution. The mean value is close to 1, and the minimum is located at 0.29. This means that this metric will result useless in terms of ranking and grouping users, since the range of values it takes ($[0.29, 0.92]$) is short and its variance (0.02157) is too low.

4.3.5 Precision

The *Precision* (P) metric measures the replanning skills of a user on a simulation, rating how he has reacted to the mission incidents. The design of this metric is based in the following assumption: A precise operator should only perform replanning interactions (add/edit/remove waypoints) when an incident occurs. Therefore, the waypoints added when no incident has happened should penalize the precision rate. Based on this, we

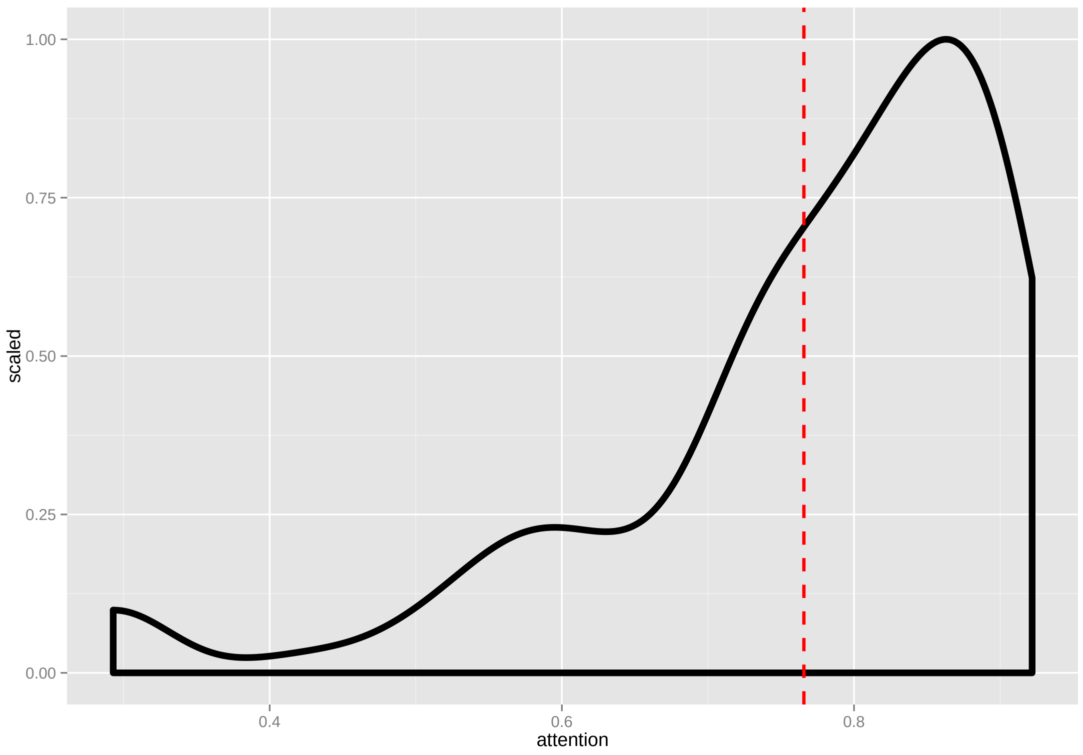


FIGURE 4.10: Density distribution for the *Attention* metric. The dashed line represents the mean value.

can divide the precision computation into two parts: The precision in times of incidents (*Incident Precision*, P_I) and the precision when nothing is altering the simulation, i.e, the operator must only monitor the simulation status (*Monitoring Precision*, P_M).

$$P(s) = \frac{P_I + P_M}{2} \tag{4.6}$$

The *Incident Precision*, P_I, supposes that every waypoint added/edited/removed during a specific interval time (10 seconds for this experiment) since the beginning of an incident is placed in order to avoid that incident, so it is considered as a precise interaction. Let $In(s)$ be the set of incidents happened during the simulation s, we can compute a *incident precision* average as follows:

$$P_I(s) = \frac{\sum_{i \in In(s)} p_I(i, s)}{|In(s)|} \tag{4.7}$$

, where $p_I(i, s)$ gives the precision for an specific incident i, computed as:

$$p_I(i, s) = 1 - \frac{1}{1 \quad + \quad |W_i(s)|} \tag{4.8}$$

In this last equation, $W_i(s)$ is the set of all *waypoint interactions* (add/edit/remove) performed since the incident i started until 10 seconds after (i.e, interactions within the interval $[startTime(i), startTime(i) + 10]$). The more waypoints are changed during that interval, the more the precision increases for that incident.

The *Monitoring Precision*, P_M, is conceptually contrary to the *Incident Precision*, in the sense that it penalizes the waypoint interactions performed during *monitoring time*, so the less interactions here, the more precision obtained. It is computed as

$$P_M(s) = \frac{1}{1 + |W_M(s)|} \quad , \tag{4.9}$$

where $W_M(s)$ is the set of all waypoint interactions performed during monitoring time. This can be seen as the complementary of all waypoint interactions made to avoid incidents, i.e:

$$W_M(s) = \overline{\bigcup_{i \in In(s)} W_i(s)} \tag{4.10}$$

To analyze the distribution of this metric, we must note that in our dataset, detailed in Section 4.2, there are two type of missions: *Replanning missions* (TestMission01,TestMission02,TestMission03), in which the UAVs started with a pre-loaded Mission Plan, and *Planning Missions* (TestMission04), in which the user had to build a Mission Plan from scratch at the beginning of the simulation. The reason why we must divide the precision metric depending on the type of mission is that, in the case of planning missions, the user is forced to add/edit/remove waypoints in order to design a valid Mission Plan, so the incident/monitoring precision cannot be evaluated in the same way we evaluate a replanning mission, where, a priori, an operator is asked to act only when it is needed.

Figure 4.11 compares the distribution of the Precision metric for these two type of missions in the dataset. Obviously, the values for the planning mission are lower than the same for replanning missions, but still, the last are not too high (The mean value is 0.3662). This indicates clearly that the users in this experiment are not expert in the field, since they have made lot of replanning interactions when nothing happened, which can be very costly in the real world.

Although the Precision metric has been designed focused on replanning missions, it can also give secondary information about skills in planning missions. On the one hand,

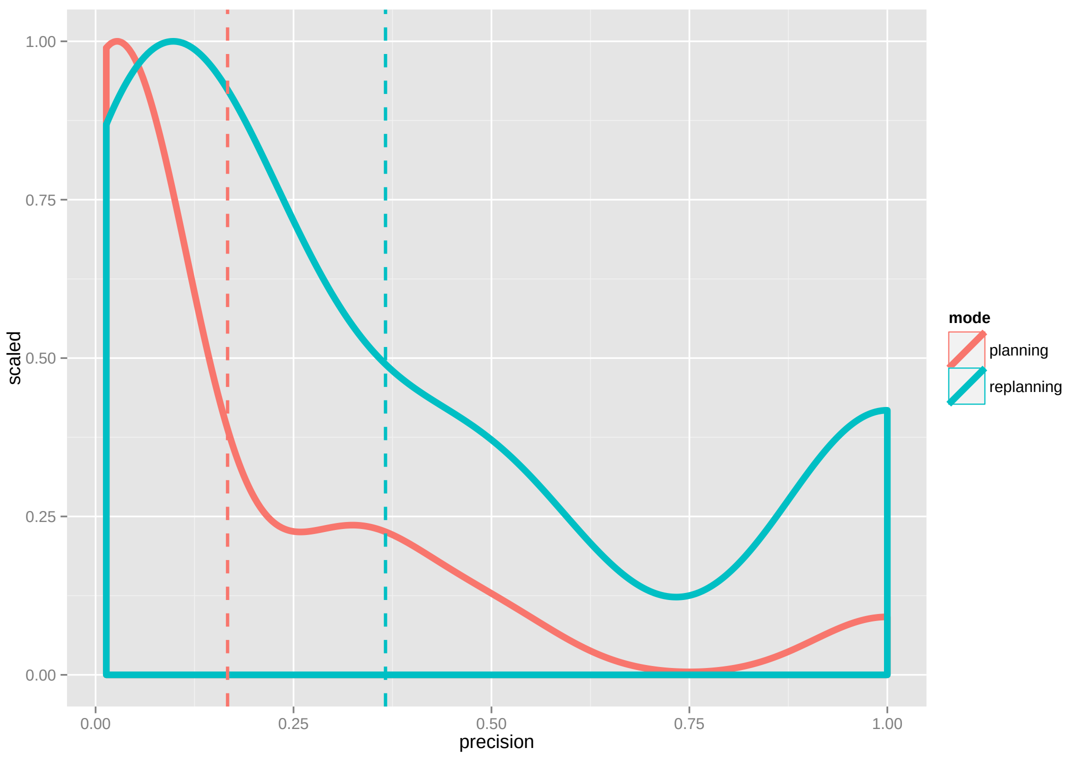


FIGURE 4.11: Density distribution for the *Precision* metric, comparing the results in replanning missions against the results in planning missions. The dashed lines represent mean values.

lower values of precision indicate that the operator (user) has designed a complex and efficient Mission Plan, good enough to avoid some incidents only by the way it was built. On the other hand, medium/high values of precision here may tell us that the user has designed a poor initial plan and he has needed to make lot of changes to it during the simulation time.

4.4 Experimental Setup

As was introduced at the beginning of this chapter, the final goal of the experimentation made in this work is to detect and analyze performance patterns among operators using the developed simulator, DWR.

The performance profile of a simulation s is defined by the tuple:

$$PPr(s) = (A(s), C(s), S(s), At(s), P(s)),$$

and based on that, we can define, for a given user u, the user performance profile, $UPP(u)$, as tuple obtain averaging all its performance profiles, i.e:

$$UPP(u) = \frac{\sum_{s \in S(u)} PPr(s)}{|S(u)|} \tag{4.11}$$

where $S(u)$ is the set of simulations executed by user u.

Computing $UPP(u)$ for all users in our dataset results in a 5-dimensional metric space, on which we can apply **clustering** methods to group together users which have similar performance profiles. Since the test missions designed for this experiment are divided in two types (replanning, planning), we must compute two different performance profiles for each user u, defined as $UPP_R(u)$, in the case of replanning, and $UPP_P(u)$, in the case of planning:

$$UPP_R(u) = \frac{\sum_{s \in S_R(u)} PPr(s)}{|S_R(u)|}, \quad UPP_P(u) = \frac{\sum_{s \in S_P(u)} PPr(s)}{|S_P(u)|}, \quad S(u) = S_R(u) \cup S_P(u) \tag{4.12}$$

To extract the similar User Performance Profiles (UPP), we make use of five clustering algorithms from the state of the art: *Hierarchical, K-means, DIANA, Model-based clustering and PAM.* All of them are tested against both the set of observations $UPP_R(U)$ and $UPP_P(U)$, where U is the set of distinct users in our dataset.

For internal validation of the User Performance Profile groups, we selected three validation measures from the state of the art that reflect the compactness, connectedness, and separation of the cluster partitions:

- *Connectivity*: Returns a value between zero and ∞ and should be minimized.
- *Dunn index*: Lies in the interval $[-1, 1]$ and should be maximized.
- *Silhouette width*: Returns a value between zero and ∞ and should be maximized.

All these clustering algorithms are tested using *Cluster k* values from 2 to 8. To perform this iterative execution and validation process, we make use of the R library *clValid* [48].

4.5 Experimental Results

As was explained in the previous sections, although the main goal of every simulation is always the same, in this experiment two type of test missions have been designed: Replanning missions (default type), in which the UAVs start with a pre-loaded Mission Plan, and Planning Missions, in which the operator has the extra task of constructing an initial plan for each UAV. Thus, the results obtained in the clustering experiment must be analyzed separately.

4.5.1 Results for Replanning Missions (TestMission 01-02-03)

Figure 4.12 shows the plots of the connectivity, Dunn index, and Silhouette width resulted from executing and validating the five clustering algorithms with the replanning dataset of UPPs. Recall that the connectivity should be minimized, while both the Dunn index and the silhouette width should be maximized. Apparently, the only metric that follows a tendency, regardless of the clustering algorithm, is the connectivity, which seems to be directly proportional to the number of clusters evaluated, reaching the minimum in 2. Here it is also appreciable that hierarchical clustering outperforms the other clustering algorithms. It is remarkable that Model based Clustering does not perform well on any of the measures. In fact, if we remove it from the plot, it seems clearer that the silhouette width decreases over the number of clusters, reaching the maximum in 2. For the Dunn index the best choice for the number of cluster is less clear.

Table 4.5: Optimal results for the clustering of User Performance Profiles on replanning missions (TestMission01,TestMission02,TestMission03).

Metric	**Score**	**Method**	**N. Clusters**
Connectivity	8.7254	Hierarchical	2
Dunn	0.5050	Hierarchical	7
Silhouette	0.3816	Hierarchical	2

The numerical results for the clustering algorithms validation are given in Table 4.6. If two or more algorithms achieve the best score for a specific measure, we choose the most popular one as optimal. Thus, from this table we can extract the optimal scores for each validation measure. The results, shown in Table 4.5, prove that **Hierarchical**

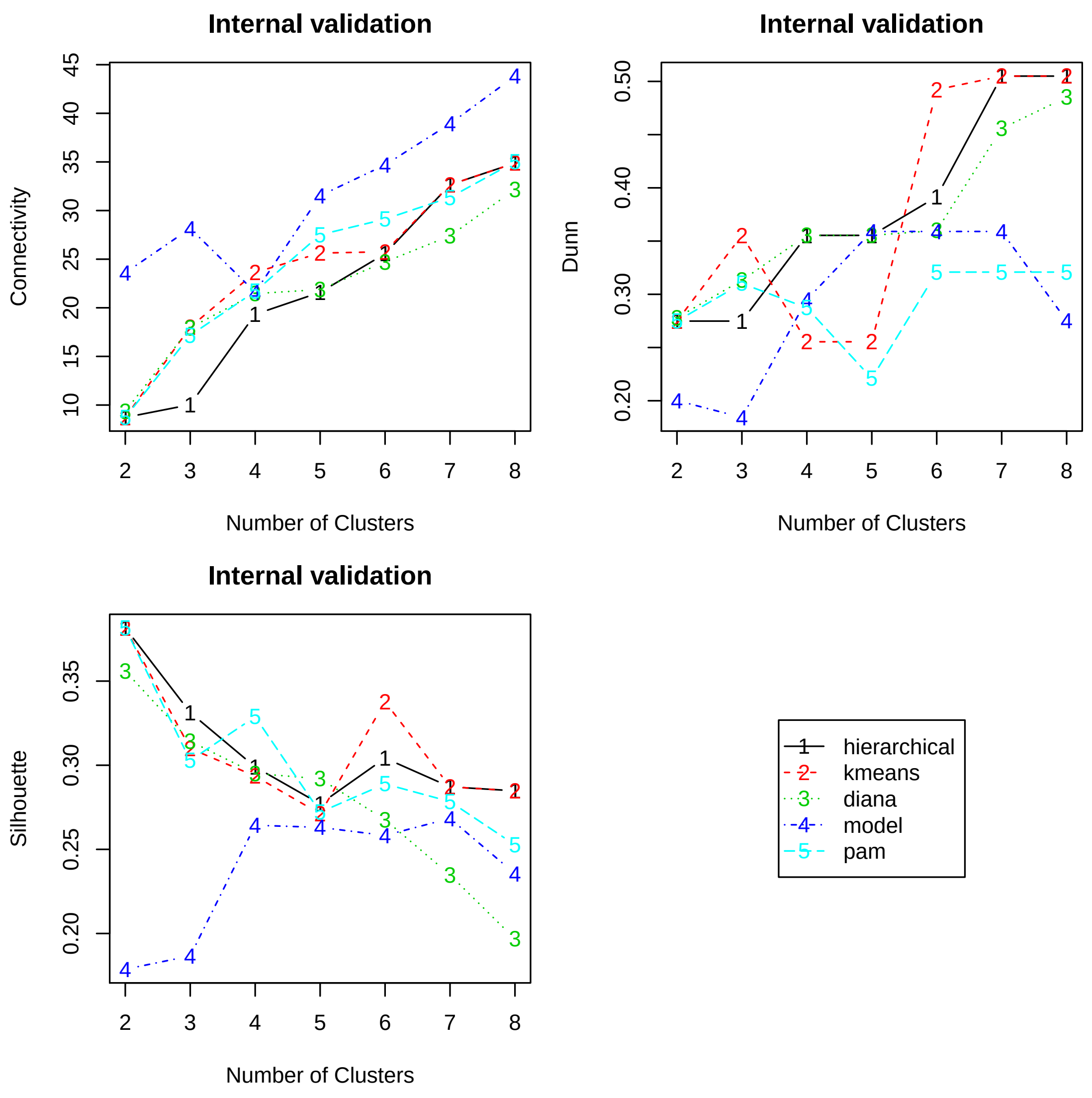


FIGURE 4.12: Plots of the connectivity measure, the Dunn index, and the Silhouette width against the fixed number of clusters when running each of the five clustering algorithms to the UPP_R dataset.

Clustering stands as the best algorithm for all validation measures, with 2 and 7 as chosen number of clusters.

Thus, we must check how the clusters are arranged for the Hierarchical Clustering execution with $k = 2$ and $k = 7$. A common dendrogram for both number of clusters is given in Figure 4.13. In the next section we will inspect and give sense to these clusters.

4.5.2 Results for Planning Missions (TestMission04)

Analogously to the analysis made above, Figure 4.14 shows the evolution of the clustering validation metrics against the number of clusters, for each of the algorithms tested on

Table 4.6: Clustering results for the User Performance Profiles on replanning missions (TestMission01,TestMission02,TestMission03). Bolded cells represent the best results obtained for each cluster validation metric

Clustering Method	Validation Metric	k=2	k=3	k=4	k=5	k=6	k=7	k=8
Hierarchical	*Connectivity*	**8.725**	10.037	19.326	21.593	25.561	32.650	34.883
	Dunn	0.275	0.275	0.355	0.355	0.392	**0.505**	0.505
	Silhouette	**0.382**	0.331	0.299	0.277	0.304	0.287	0.285
K-Means	*Connectivity*	8.725	18.015	23.638	25.630	25.760	32.650	34.883
	Dunn	0.275	0.355	0.255	0.255	0.492	0.505	0.505
	Silhouette	0.382	0.310	0.294	0.271	0.338	0.287	0.285
DIANA	*Connectivity*	9.353	17.955	21.459	21.876	24.719	27.395	32.178
	Dunn	0.278	0.313	0.355	0.355	0.360	0.456	0.485
	Silhouette	0.356	0.314	0.295	0.292	0.268	0.235	0.197
M-Clustering	*Connectivity*	23.600	28.164	21.605	31.528	34.689	38.922	43.826
	Dunn	0.200	0.184	0.295	0.359	0.359	0.359	0.275
	Silhouette	0.179	0.187	0.264	0.263	0.258	0.268	0.236
PAM	*Connectivity*	8.725	17.170	21.683	27.508	29.119	31.352	35.012
	Dunn	0.275	0.310	0.287	0.221	0.321	0.321	0.321
	Silhouette	0.382	0.303	0.329	0.272	0.289	0.278	0.253

the UPP_P dataset. Connectivity and silhouette width, as it happened previously, seems to be dependent on the number of clusters whatever the algorithm used, while Dunn index is apparently free of a clear general tendency. On the one hand, Model base clustering does not perform well on any of the measures, so we can conclude that it is not appropriate for this type of problem. On the other hand, Hierarchical clustering and DIANA achieve good results, and it is remarkable that they get similar results almost for every case. K-means obtain good results too, specially for the Dunn index, where it gets the maximum score at $k = 7$.

Comparing this plot with the same of the above section (results for replanning missions), we can see that, regardless of the validation measure, the best score achieved when clustering the UPP_P dataset is better than the one achieved in the UPP_R dataset. Table 4.7 proves this by showing the $[min, max]$ ranges obtained in each measure and dataset. This result may be caused simply because the size of the UPP_P data set (16

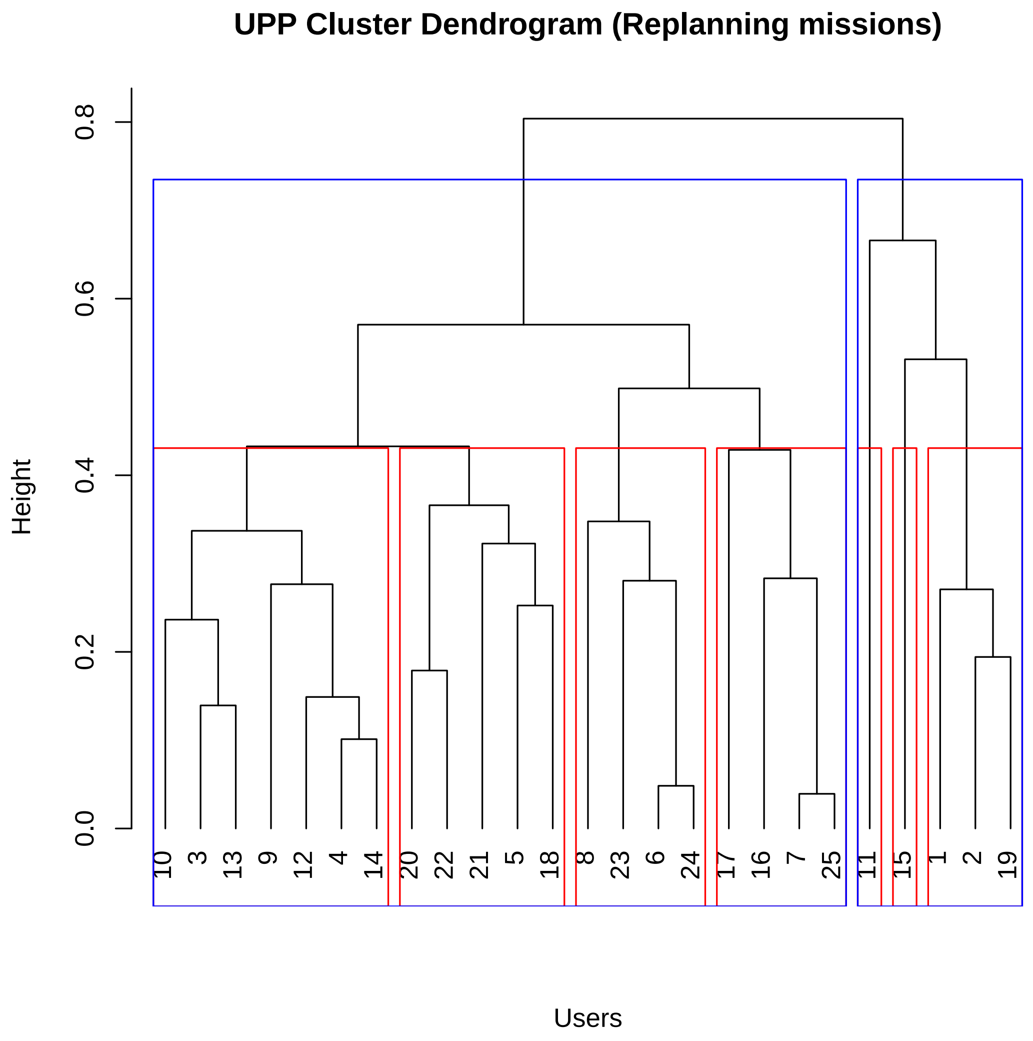


FIGURE 4.13: Plot of the dendrogram for hierarchical clustering with k=2, k=7 on the UPP_R dataset.

user profiles) is less than the one in UPP_R (25 user profiles), and hence the clustering process is likely to perform better.

TABLE 4.7: Comparison of $[min, max]$ ranges for the clustering validation on UPP_R and UPP_P datasets.

Metric	**Range in UPP_R dataset**	**Range in UPP_P dataset**
Connectivity	$[8.725, 43.826]$	$[\mathbf{2.929}, 33.817]$
Dunn	$[0.184, 0.505]$	$[0.182, \mathbf{0.610}]$
Silhouette	$[0.162, 0.382]$	$[0.113, \mathbf{0.482}]$

As done in the previous case, we show the complete set of validation results in Table 4.9, and extract from it the optimal scores for each metric, shown in Table 4.8. The results

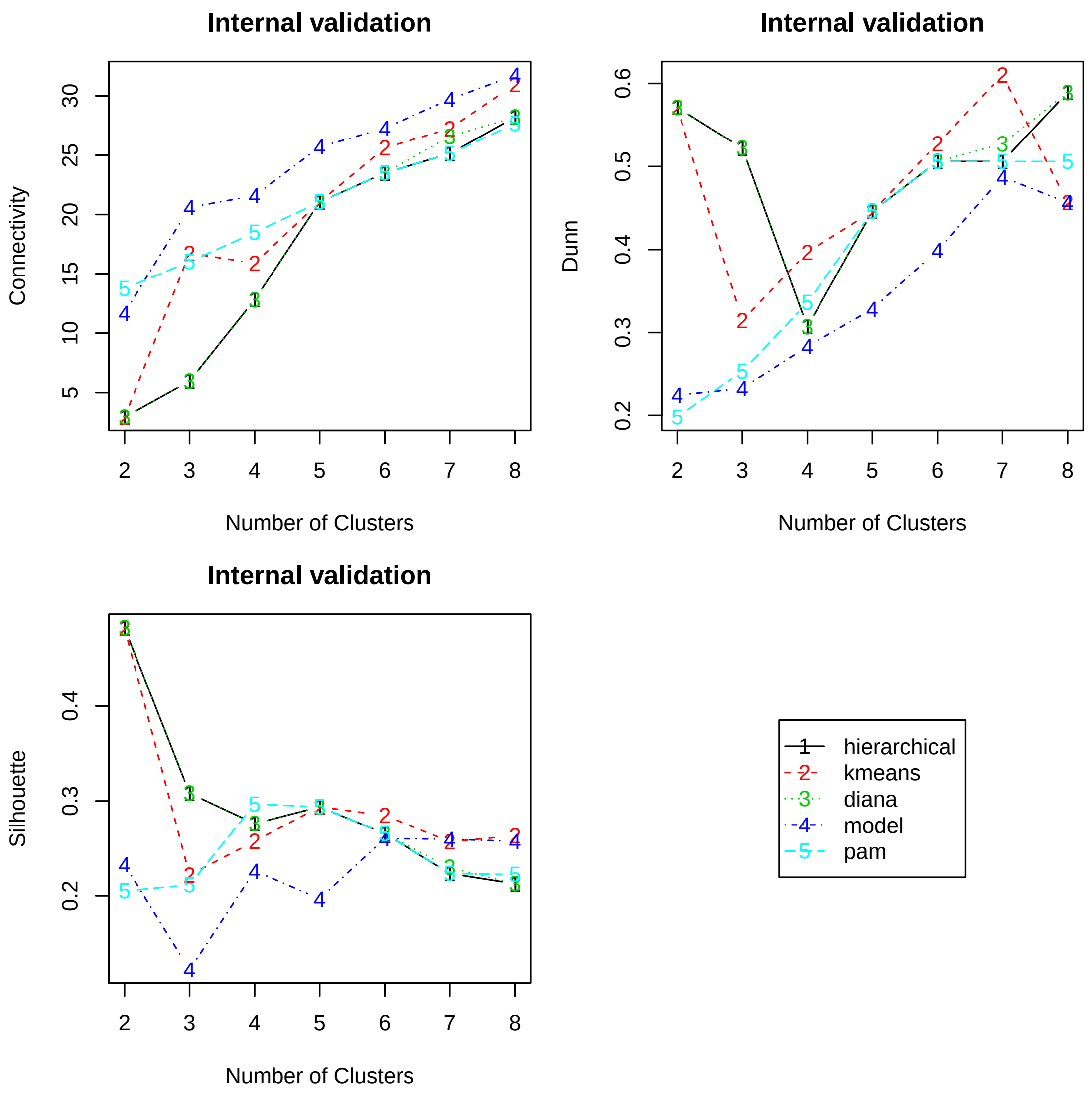


FIGURE 4.14: Plots of the connectivity measure, the Dunn index, and the silhouette width against the number of clusters when running each of the five clustering algorithms to the UPP_P dataset.

in the number of clusters to use are the same (2 and 7), but this time Hierarchical clustering does not impose on the rest of the algorithms completely, since **K-means outperforms it for the Dunn Index, with k=7**.

In order to check which are the clusters resulted from the execution of the optimal algorithms, we plot them: Figure 4.15 shows a dendrogram with the results of Hierarchical Clustering with k=2, and Figure 4.16 represents graphically the clusters, projected into a 2-dimensional composed of the two principal components from our 5-dimensional space (named as Component1 and Component2 in the plot). To do this projection, a Principal

Table 4.8: Optimal results for the clustering of User Performance Profiles on planning missions (TestMission04).

Metric	**Score**	**Method**	**N. Clusters**
Connectivity	2.9290	Hierarchical	2
Dunn	0.6099	K-means	7
Silhouette	0.4825	Hierarchical	2

Table 4.9: Clustering results for the User Performance Profiles on planning missions (TestMission04). Bolded cells represent the best results obtained for each cluster validation metric

Clustering Method	**Validation Metric**	**k=2**	**k=3**	**k=4**	**k=5**	**k=6**	**k=7**	**k=8**
Hierarchical	*Connectivity*	**2.929**	5.969	12.811	21.051	23.504	25.129	28.204
	Dunn	0.571	0.522	0.307	0.446	0.506	0.506	0.589
	Silhouette	**0.483**	0.308	0.276	0.294	0.266	0.224	0.213
K-means	*Connectivity*	2.929	16.743	15.893	21.051	25.618	27.243	30.969
	Dunn	0.571	0.314	0.397	0.446	0.527	**0.610**	0.456
	Silhouette	0.483	0.222	0.258	0.294	0.285	0.257	0.263
DIANA	*Connectivity*	2.929	5.969	12.812	21.051	23.504	26.579	28.204
	Dunn	0.571	0.522	0.307	0.446	0.506	0.527	0.589
	Silhouette	0.483	0.308	0.276	0.294	0.266	0.230	0.213
M-Clustering	*Connectivity*	11.702	20.604	21.587	25.693	27.281	29.715	31.743
	Dunn	0.225	0.233	0.283	0.328	0.399	0.487	0.4563
	Silhouette	0.233	0.122	0.226	0.197	0.260	0.260	0.257
PAM	*Connectivity*	13.744	16.028	18.490	21.051	23.504	25.129	27.654
	Dunn	0.198	0.253	0.336	0.446	0.506	0.506	0.506
	Silhouette	0.205	0.212	0.297	0.294	0.266	0.224	0.222

Component Analysis (PCA) algorithm is used. Next section will discuss the results obtained by this optimal clustering algorithms, and how we can extract some meaningful patterns from them.

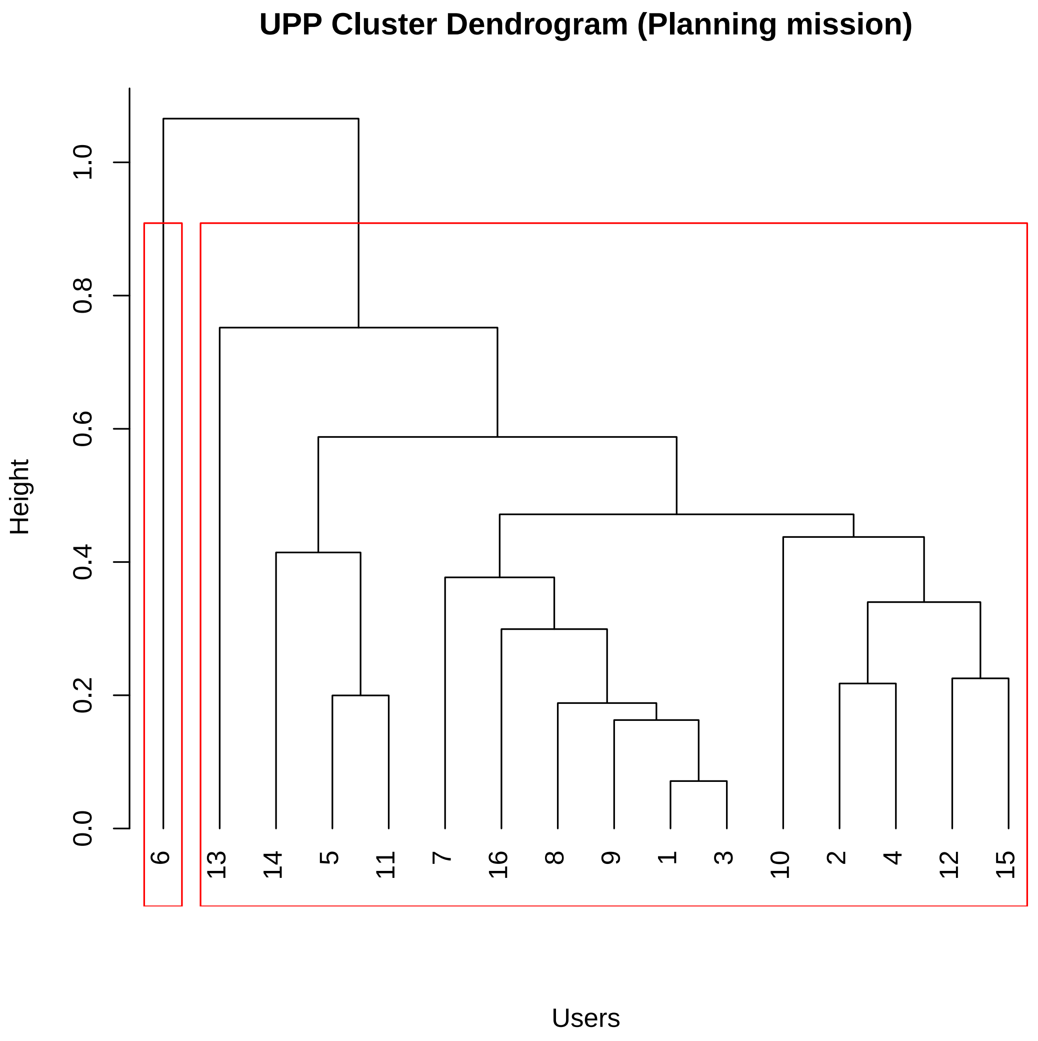


FIGURE 4.15: Plot of the dendrogram for hierarchical clustering with k=2, using the UPP_P dataset.

4.6 Discussion

In this last section, once we have validated and extracted the optimal algorithms for both UPP_R and UPP_P datasets, we must analyze the clusters obtained in order to give a sense to them, and to define which behavior patterns have been found among the users participating in the experiment. Due to the small number of users in this experiment, we can do this manually by analyzing one by one the user profiles in every cluster.

As was done in the previous section, the cluster analysis will be separated into 2 parts, depending of the dataset being analyzed (UPP_R or UPP_P).

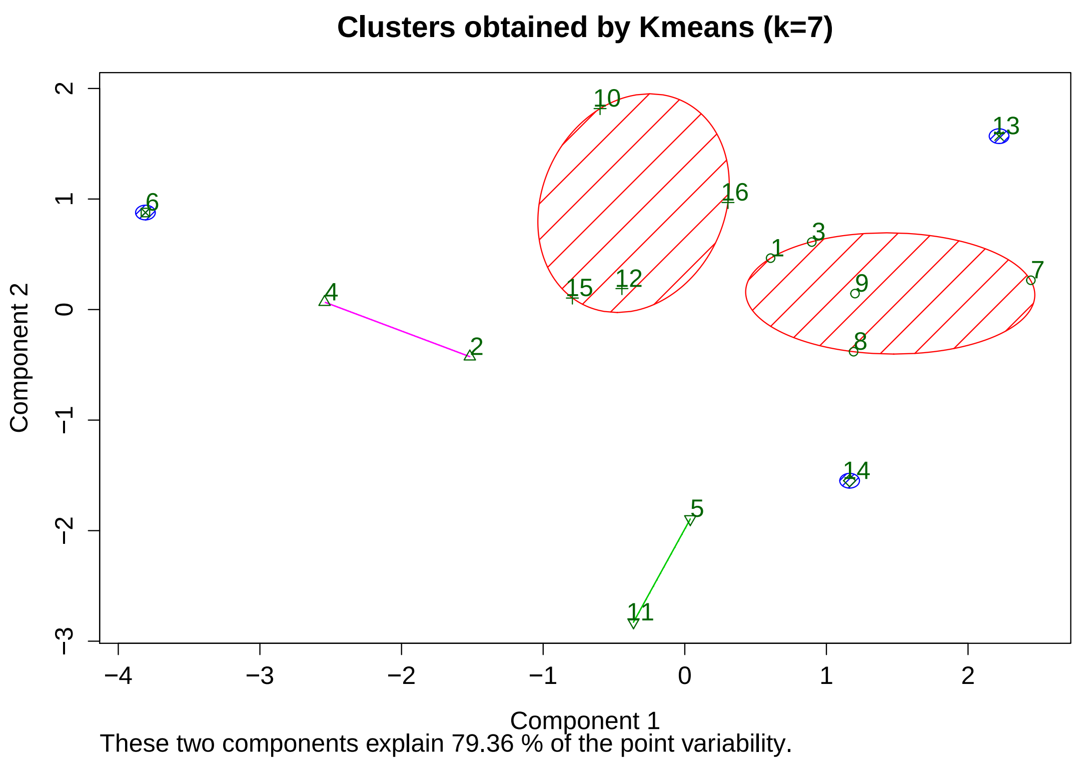


FIGURE 4.16: K-means clusters for k=7, using the UPP_P dataset.

4.6.1 Replanning Profiles Patterns

In the previous section we proved that the dataset composed of User Performance Profiles (UPPs) in replanning missions (TestMission01,02,03), also called $UPP_R(U)$, was optimally clustered by a Hierarchical algorithm with 2 and 7 clusters. That means that we can distinguish two general behavioral patterns, and those can be broken down into seven more specific patterns.

4.6.1.1 UPP_R Clusters for Hierarchical Clustering, k=2

Below are detailed the explanation and behavioral patterns extracted from each of the clusters obtained by the *Hierarchical Clustering* algorithm with 2 clusters. Figure 4.17 show these results graphically.

1. *UPP_R elements: (1,2,11,15,19); Users: (05_112, 05_113, 05_129, 05_135 y 07_115)*: This cluster is specially representative for having **high levels of precision**. Somewhat surprisingly, the score level does not perform well on any user of the cluster despite the high precision level. If we check the secondary features (Consumption,

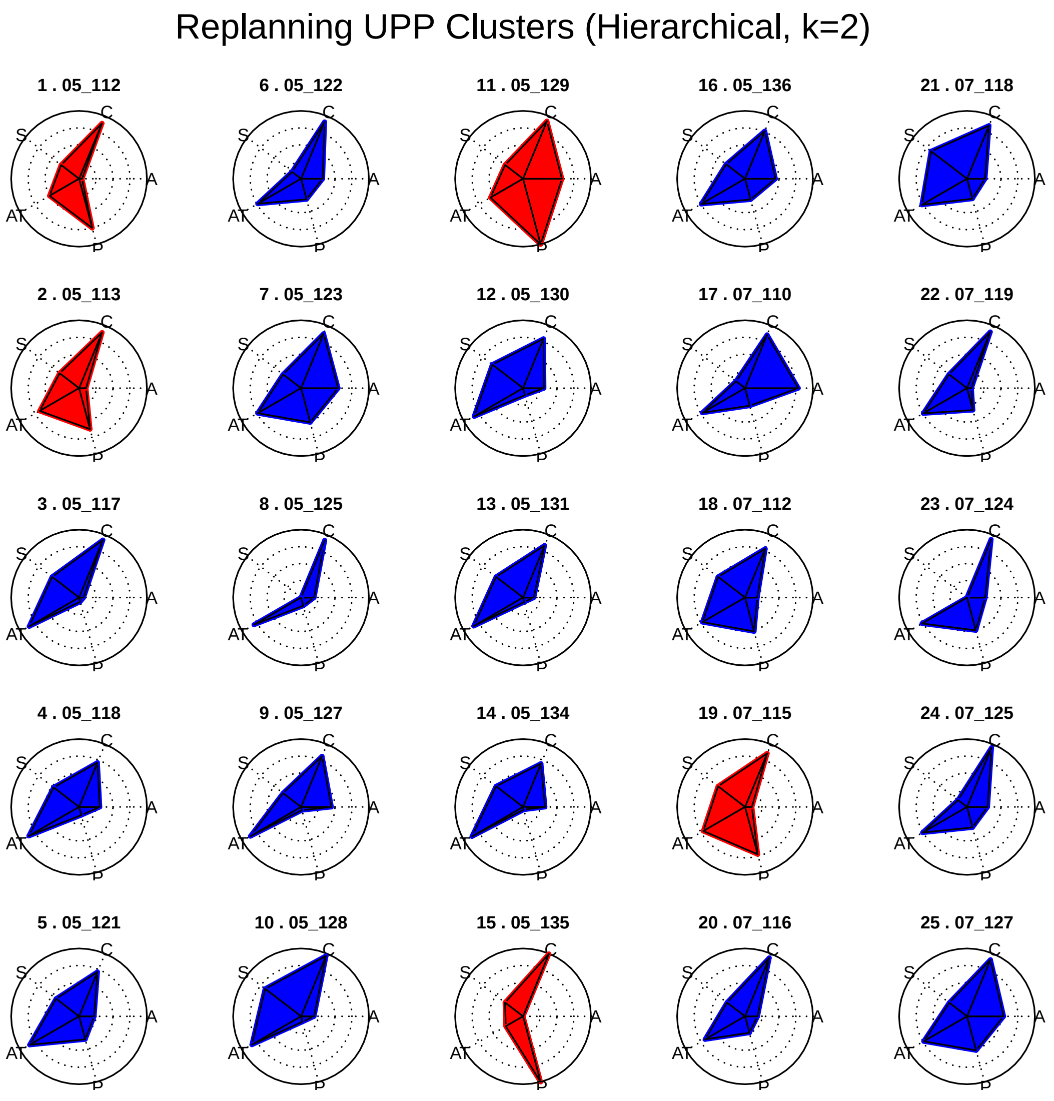


FIGURE 4.17: Replanning User Performance Profile Clusters, resulted from the execution of Hierarchical Clustering with k=2.

Agility, Attention), we can conclude that these are slow users who have played short simulations, making not too much interactions, but precise ones. They will be labeled as **Impatient with potential** (See Figure 4.17, red UPPs).

2. *Rest of the users*: The variety obtained in this cluster makes difficult to extract a pattern from it, so we must analyze the subgroups obtained by a deeper cut in the hierarchical tree (See Figure 4.17, blue UPPs).

4.6.1.2 UPP_R Clusters for Hierarchical Clustering, k=7

Below are detailed the explanation and behavioral patterns extracted from each of the clusters obtained by the *Hierarchical Clustering* algorithm with 7 clusters. Figure 4.18 show these results graphically.

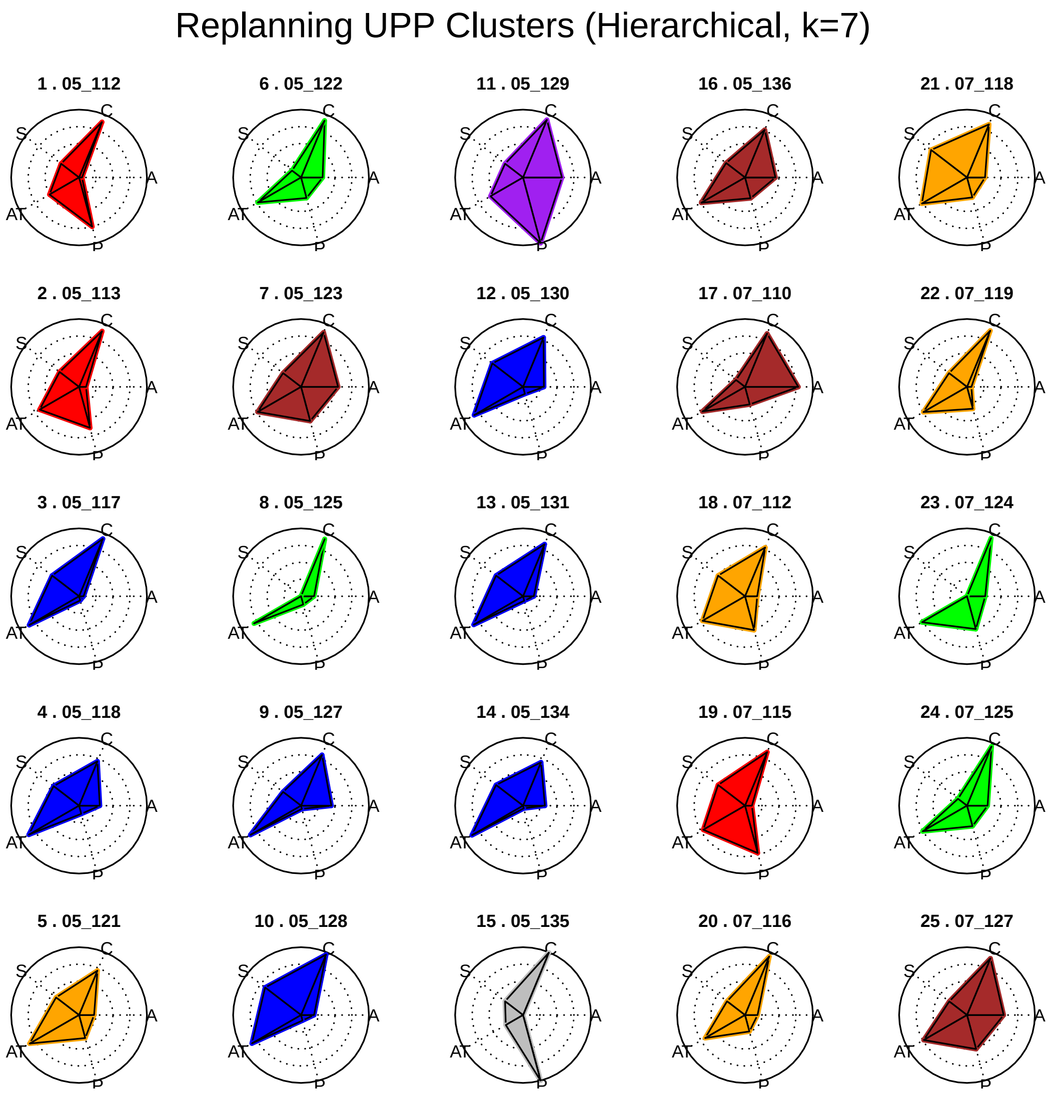


FIGURE 4.18: Replanning User Performance Profile Clusters, resulted from the execution of Hierarchical Clustering with k=7.

1. UPP_R *elements: (11), Users: (05_129)*: This cluster comprises only one user. The user stands out because he obtains the biggest *profile area* of all users, reaching maximum levels of precision and consumption, but low score levels. He represent an impatient user which performed very fast and precise interactions. Their

behavior can be labeled as **Fast Impatient with replanning potential** (See Figure 4.18, purple profiles).

2. *UPP_R elements: (15), Users: (05_135)*: This cluster comprises only one user. Its performance profile can be labeled as **Slow impatient with replanning potential**, due to the low agility level it features and the shortness of his missions. Probably he played missions until the first incidence appeared, then slowed down the simulation speed and tried to avoid them, with bad results, which caused him to abort the mission (See Figure 4.18, gray profiles).

3. *UPP_R elements: (1,2,19), Users: (05_112, 05_113, 07_115)*: This cluster represents a soft version of the previous one (Element 15). Users here are cautious due to its slow level of agility, and its precision/score/attention balance says that they tried to avoid the incidents making a complex *incident replanning* (which gives them high precision), but lost the focus on the mission targets (which gives them low scores). Its behavior could be labeled as **focused on avoiding incidents** (See Figure 4.18, red profile).

4. *UPP_R elements: (5,18,20,21,22), Users: (05_121, 07_112, 07_116, 07_118, 07_119)*: If we look at the distribution of each UPP metric in table 4.4, we can conclude that this cluster represents an average UPP_R profile. This means that users in this cluster are a representative sample of the general behavior of the students in our dataset. Precision is low, as well as agility, and the consumption is not excessively high, which indicates that the missions have not been prematurely aborted. This, together with the high rate of attention and the variable score values lead us to conclude that users in this cluster tend to feature **restless behaviors, aggressive, more focused on detecting targets than avoiding incidents efficiently**. This makes sense given that students are novice users all of them (See Figure 4.18, yellow profiles).

5. *UPP_R elements: (3,4,9,10,12,13,14), Users: (05_117, 05_118, 05_127, 05_128, 05_130, 05_131, 05_134)*: This cluster stands out for its precision-attention balance. The rest of the metrics maintain expected values with respect to their distributions, but in all cases, the precision is very low and the attention exceeds

the average. This is clearly associated to a **reckless and aggressive behavior**, focused on detecting targets as soon as possible, ignoring the effects of the incidents (See Figure 4.18, blue profiles).

6. *UPP_R elements: (6,8,23,24), Users: (05_122, 05_125, 07_124, 07_125)*: This cluster clearly **represents a bad UPP_R profile**, undesirable for anyone looking for good operators. This is because the score metric performs extremely low values, and recalling the score equation (See Section 4.3.3), this means that not only none of the targets have been detected, but also the UAVs have been destroyed (See Figure 4.18, green profiles).

7. *UPP_R elements: (7,16,17,25), Users: (05_123, 05_136, 07_110, 07_127)*: Users in this cluster share a high level of agility, above the average. The rest of the metrics revolve around the average values, which lead us to conclude that these users can be trained to respond quickly to unknown situations. They could be labeled as **agile users** (See Figure 4.18, brown profiles).

4.6.2 Planning Profile Patterns

As was said above, the main difference when analyzing planning profiles (UPP_P) with respect to the analysis of replanning profiles (UPP_R) lies in **the meaning of the precision metric**. As was defined in Section 4.3.5, this metric focuses on rating how the users change a plan only when it is needed, i.e, only when incidents happen. Thus, in planning missions this metric is expected to perform very low values, because users are forced to build a mission plan at the beginning of the simulation. However, in this cases the precision can give us additional information: Low values will be associated to complex and wise initial plans, and high values will be related to an impatient behavior.

In the previous section we proved that the dataset composed of User Performance Profiles (UPPs) in planning missions (TestMission04), also called $UPP_R(U)$, was optimally clustered by a Hierarchical algorithm with 2 clusters and a K-means algorithm with 7 clusters. Thus, we shall analyze those clusterizations separately.

4.6.2.1 UPP_P Clusters for Hierarchical Clustering, k=2

Below are detailed the explanation and behavior patterns extracted from each of the clusters obtained by the *Hierarchical Clustering* algorithm with 2 clusters. Figure 4.19 shows these results graphically.

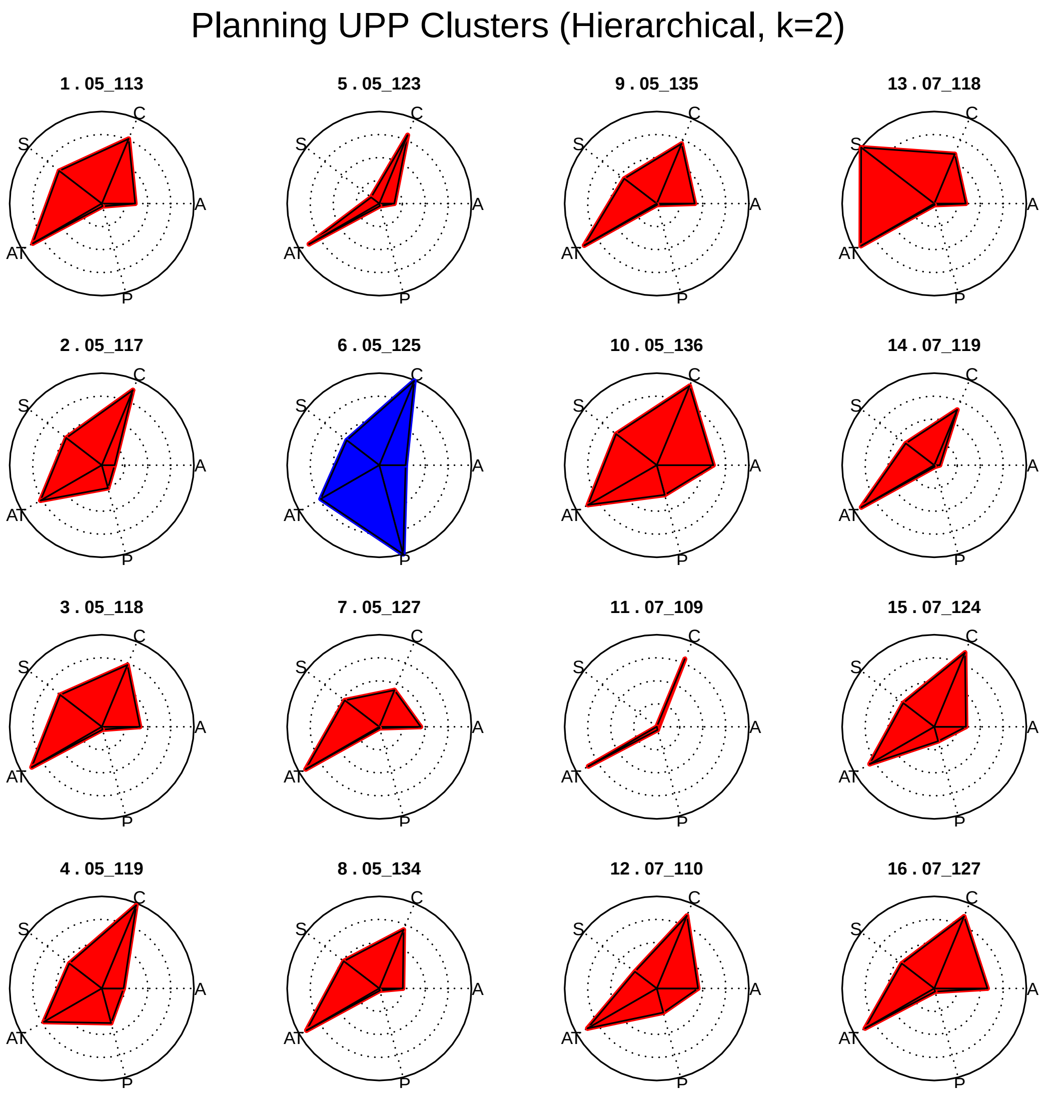


FIGURE 4.19: Planning User Performance Profile Clusters, resulted from the execution of Hierarchical Clustering with k=2.

1. *UPP_P elements: (6), Users: (05_125)*: This cluster comprises only one user, clearly distinguished by having the maximum level of precision ($P = 1$), in a mission in which it is expected to have low precision values (The average precision value is around 0.16, as shown in Table 4.4). This means that this user **has not created a beginning plan**, but has run the simulation until the incidents

appeared, and based on those incidents, he made a precise plan to avoid them. This is actually a wrong way of playing this mission, and thus will be labeled as **tricky planner** (See Figure 4.19, blue profiles).

2. *Other profiles*: The rest of the profiles in this clusterization share all of them low levels of the precision metric, which mean that they have played the planning mission correctly. However, it is needed a deeper sub-clusterization in order to group these profiles more specifically (See Figure 4.19, red profiles).

4.6.2.2 UPP_P Clusters for K-means, k=7

Below are detailed the explanation and behavior patterns extracted from each of the clusters obtained by the *K-means* algorithm with 7 clusters. Figure 4.20 show these results graphically.

1. *UPP_P elements: (6), Users: (05_125)*: This is the **tricky planner** cluster explained above. It was also obtained by the Hierarchical Clustering algorithm with k=2 (See Figure 4.20, brown profiles).

2. *UPP_P elements: (2,4), Users: (05_117, 05_119)*: This cluster features profiles close to the average values for all the metrics. The precision performs above the mean value for this type of mission, which indicates, together with the high consumption and the low attention, that **the initial plans created by these users were simple**, and aborted prematurely. It could be labeled as **Simple and unfinished plans** (See Figure 4.20, gray profiles).

3. *UPP_P elements: (5, 11), Users: (05_123, 07_109)*: This cluster stands out for having a really low profile balance, specially in terms of score. The precision and attention rates may tell that the created plans here are complex, and thus efficient, but it seems that nothing was done to avoid the incidents, and hence all UAVs were destroyed. This plans could be labeled as **destructive** (See Figure 4.20, green profiles).

4. *UPP_P elements: (13), Users: (07_118)*: Undoubtedly, this cluster, which comprises only a user, represents the best performance of all analyzed profile. Not only it gets the maximum score, but also its precision value is minimum, which

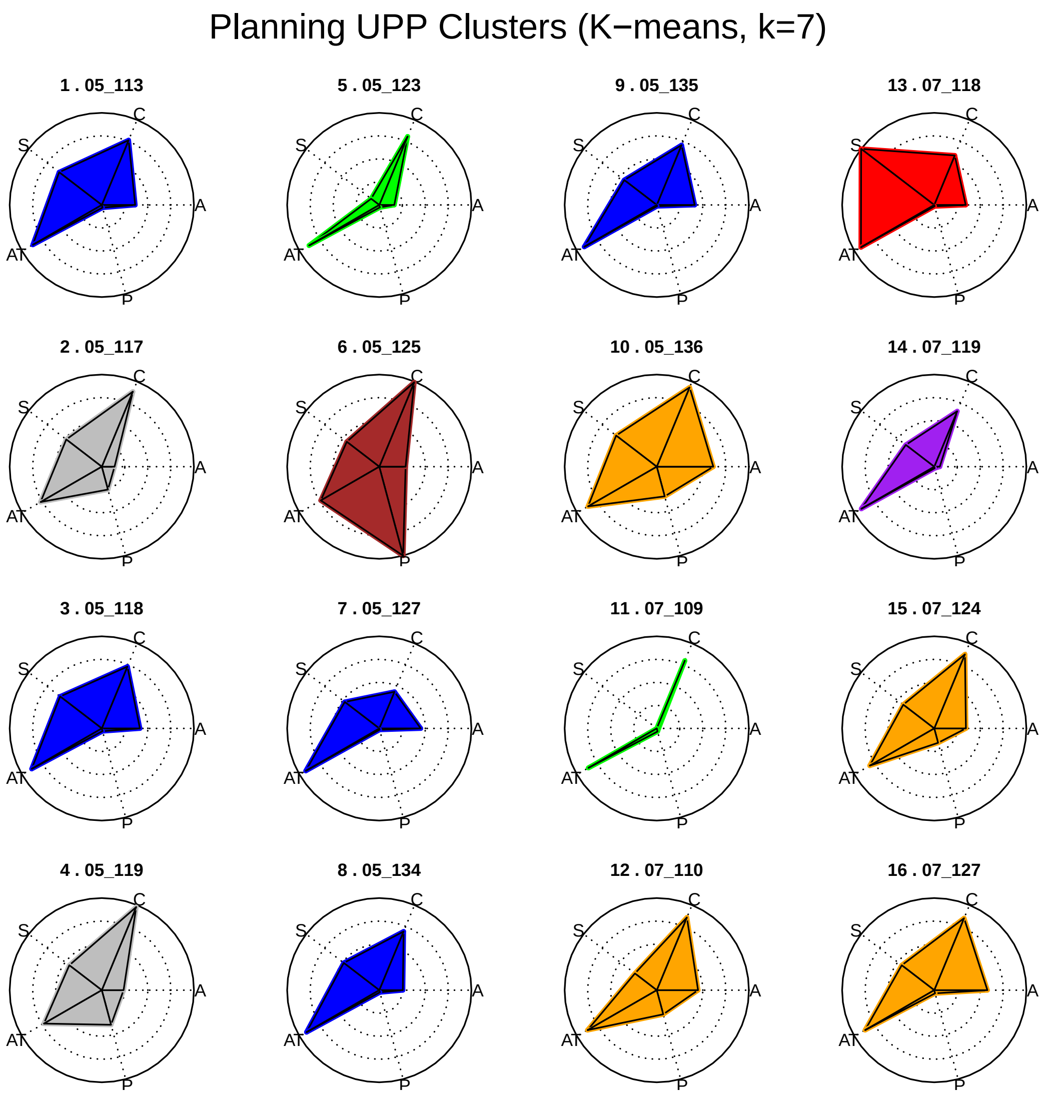


FIGURE 4.20: Planning User Performance Profile Clusters, resulted from the execution of K-means with k=7.

indicates that the initial plan designed was so good that it scarcely needed to be replanned. Thus, this user is labeled as **best performance** (See Figure 4.20, red profiles).

5. *UPP_P elements: (14), Users: (07_119)*: This cluster comprises only a user, and it features, as explained above, a precision-attention balance proving a wise and complex initial planning. Also, the low levels of agility and consumption make clear that the user executed its initial plan and trusted it to avoid the incidents and detect the targets. Unfortunately, the plan did not perform well score rate, hence probably this user should have made a small replanning during the simulation. It

could be labeled as **complex and unsuccessful** plan (See Figure 4.20, purple profiles).

6. *UPP_P elements: (10, 12, 15, 16), Users: (05_136, 07_110, 07_124, 07_127)*: This cluster stands out for its high values on the agility metric. Plans created by these users were initially simple and grew during the simulation time. Thus, this cluster can be labeled as **intense replanning** (See Figure 4.20, orange profiles).

7. *UPP_P elements: (1, 3, 7, 8, 9), Users: (05_113, 05_118, 05_127, 05_134, 05_135)*: Users in this cluster feature a consumption rate lower than the rest of clusters in this clusterization, as well as minimum values of precision. This means that they have build wise, complex and long plans from the beginning of the mission, and the have executed them without aborting the mission nor making many changes in them. They also feature medium agility values, which lead us to conclude that they were actively monitoring their mission plan execution. Scores in this cluster do not get significant values, but still, these users could be labeled as **complex planners** (See Figure 4.20, blue profiles).

Chapter 5

Conclusions and Future Work

5.1 Conclusions

This work has presented the design and development of a lightweight multi-UAV simulator, focused on retrieving data from the interactions of UAV operators in a simple and accessible way. The high degree of expansion of UAV operations has resulted in an increasing requirement of qualified UAV operators to supervise the missions. This new demand of operators has created the need for new, open and simple simulation and training platforms where a inexperienced user is able, on the one hand, to familiarize with the world of Human-Supervisory Control in unmanned systems, and on the other hand, to train its skills in order to become a potential UAV expert operator.

Due to that, the simulator, which has been named as *Drone Watch And Rescue*, has been created following the criteria of simplicity and accesibility. The main features in the development of this simulator include: *gamification elements*, as dynamic incidents, mobile targets and scores, introduced to challenge the operator skills during a simulation; *simple interface and controls*, to allow an easy learning curve of the simulator mechanisms; a *2-level web architecture*, which allows any user in the world to access and use the simulator using just a modern web browser, and a *complete and robust data extraction*, which catches all events and interactions happened during a simulation and permit complex and effective analysis of the resulting data.

Once the simulator development is finished, we have created a load test to ensure that the designed architecture is able to host simulations from hundreds of users at the same time. Then, the simulator has been deployed and used to carry out an experiment with inexperienced students at the Autonomous University of Madrid.

The goal of the experiment is to extract and discover behavioral patterns from the performance of each of the users participating in the experiment, and detect the type and expertise of the user using the simulator. To achieve this, three main steps have been followed: First, five performance metrics have been designed in order to measure the quality of the user interactions during a simulation. Then, based on those metrics, each user is associated to a *user performance profile*, calculated by averaging the performance profile of all his/her simulations. Finally, the user profiles are introduced into several *clustering algorithms*, to discover some groups or patterns in the user performance. The clusterizations obtained by the algorithms are validated, optimized and analyzed to extract the behavioral patterns hidden among the user profiles. The results show that the metrics and profiles created for this experiment characterize well the low expertise and novice behavior of the users in the experiment.

5.2 Future Work

Since this work is clearly divided into two parts (Development & Analysis), there are different lines of future work depending on where we focus.

On the one hand, regarding to the development of *Drone Watch And Rescue*, we can set multiple developing lines to improve the platform:

- Although the simulator is not focused on achieving high fidelity in terms of physics and UAV technical issues, it is needed to improve the simulation mechanisms and interface of DWR, to bring it closer to the functionality of a real GCS. Achieving this without losing the essence of simplicity and accessibility is a big challenge, but it would reduce the gap between expert operators and normal users.
- In order to make DWR become a complete UAV mission training platform, it is necessary to extent the tool so that not only it is able to simulate specific missions, but also offer complete *training exercises*, with different difficulty levels. This would allow a deeper analysis of the user evolution through the successive

exercises, assessing their learning curve and their potential in a more objective manner.

On the other hand, in terms of the performance analysis made in this work, there are also many improvements and extensions that can be applied:

- It is desirable to improve the designed performance metrics in many ways. First, we must ensure that all of them offer valuable information. In the experimentation made in this work, we saw that some metrics, as *consumption* and *attention*, resulted in density distributions featuring a very-low variance value. This is usually undesirable for the purposes of data analysis, thus we must improve the metrics to perform better variance values. Besides, in order to bring this simulation environment closer to a real scenario, the performance metrics should be adapted to become more sophisticated and appropriate to the related state of the art in the field [36].

- Apart from the *internal validation measures* used to validate the clustering algorithms applied in the experimentation of this work, it would be interesting to add some *stability measures* to the analysis, as the *Average proportion of non-overlap (APN)*, the *Average distance (AD)* or the *Figure of Merite (FOM)* [48]. These metrics compare the results from clustering based on the full data to clustering based on removing each column (feature), one at a time [56]. These measures work especially well if the data are highly correlated, and help us to decide if all the features that we are choosing for clustering are relevant.

- The cluster analysis made in the experimentation of this work consisted of manually assigning behavioral "tags" to each group, based on the shape of the user profiles in the group cluster. A big improvement to this experimentation would be to do this analysis in an automatic way.

- As was described in the state of the art, some of the most relevant works in the field of analyzing behavioral patterns in UAV operators use unsupervised Hidden Markov Models as a way of modeling the different cognitive states of the operator [25, 26]. An interesting future research line would include using that type of modeling technique, and comparing it with the clustering methods used in this work.

Appendix A

DWR Dataset

This appendix shows the entire dataset used to save the data from a simulation executed by the simulator *Drone Watch And Rescue*. The Data Base Management System used to store this dataset is *MongoDB*, and hence the data is organized into collections.

Below are detailed the collections of each database created for DWR, as well as the Enumeration Data Types needed to save some data.

A.1 UAS SIMULATION Database

The *UAS_Simulations* database contains a set of collections storing the results of every simulation executed in DWR. The relation between each collection in this database is detailed in Section 3.2.3, and displayed graphically into an ER-Diagram in Figure 3.8 of that section. Tables A.1,A.2,A.3,A.4 detail the attributes of each of these collections.

A.2 UAS MISSION Database

The *UAS_Mission* database contains the set of collections defining the essential components of any input mission in DWR, i.e, the Mission Scenario and the Mission Plan. The data from these collections is shared with the works of Ramirez-Atencia et al. in [72]. See this work for details of the attributes contained in this database.

Table A.1: Simulations collection.

Name	**Format**	**Description**
_id	ObjectId	MongoDB unique identifier
name	String	*Optional.* Identifies the user running the simulation.
clientIP	String	IP address of the device running the simulation.
createdAt	Date	Indicates the exact moment when the simulation started.
missionPlanId	ObjectId*-Reference to *(UAS_MissionPlans, Plans)*	Foreign key identifying the mission plan loaded in this simulation.

A.3 UAS SCENARIO Database

The *UAS_Scenario* database contains a set of collections comprising the gamification elements of a mission in DWR, as detailed in Section 3.2.2. Tables A.5,A.6,A.7,A.8

Table A.2: Simulation snapshots collection.

Name	**Format**	**Description**
_id	ObjectId	MongoDB unique identifier
simulation	ObjectId*-Reference to *(UAS_Simulations, Simulations)*	Foreign key identifying the simulation instance to which this snapshot belongs to.
simulationElapsedTime	Number	Time when this snapshot was taken, in milliseconds, measured in the simulation time line since the beginning of the mission.
realElapsedTime	Number	Time when this snapshot was taken, in milliseconds, measured in the real time line since the beginning of the mission.
simulationSpeed	Number	Simulation Speed at the time when the snapshot was taken. This value lies in the interval $[1, 1000]$
cause	EventType (Enum)	Identification of the *Event Type* causing this snapshot. See A.9,A.10

TABLE A.3: Simulation snapshots collection.

Name	**Format**	**Description**
_id	ObjectId	MongoDB unique identifier
simulationSnapshot	ObjectId*-Reference to *(UAS_Simulations, SimulationSnapshots)*	Foreign key identifying the simulation snapshot instance to which this snapshot belongs to.
UAVId	String	Identifier of the UAV associated to this snapshot.
status	UAVStatus(Enum)	Current status of the UAV at the time when the snapshot was taken. See Table A.11.
remainingFuel	Number	Current remaining fuel (L) at the time when the snapshot was taken.
speed	Number	Current UAV speed (Km/h) at the time when the snapshot was taken.
position	(Number,Number)	Current UAV position ($Latitude, Longitude$) at the time when the snapshot was taken.

detail the attributes of each collection comprising this database.

A.4 Enumerations

The term *Enumeration* in this work refers to a closed data structure, which only can have a finite set of values. However, some of these enumerations admit the use of parameters. Tables A.9,A.10,A.11,A.12,A.13,A.14 detail all possible values for each enumeration in DWR.

Table A.4: Waypoints collection.

Name	**Format**	**Description**
_id	ObjectId	MongoDB unique identifier
droneSnapshot	ObjectId*-Reference to *(UAS_Simulations, DroneSnapthots)*	Foreign key identifying the drone snapshot instance to which this waypoint belongs to.
position	(Number,Number)	Position in which the waypoint is located, in geodesic coordinates $(Latitude, Longitude)$.
type	WAypointType(Enum)	Waypoint Type. See Table A.14.
plannedTime	Number	Time in milliseconds, measured from the beginning of the simulation timeline, in which it is expected to reach this waypoint.

Table A.5: Incidents Plan collection.

Name	**Format**	**Description**
_id	ObjectId	MongoDB unique identifier
mission	ObjectId*-Reference to *(UAS_MissionInput, MissionInput)*	Foreign key identifying the Mission Scenario associated to this Incidents Plan.
incidents	Array	Lists the incidents comprising this incidents plan. See Table A.6

Table A.6: Incidents collection.

Name	**Format**	**Description**
_id	ObjectId	MongoDB unique identifier
type	IncidentType(Enum)	Identifier of the incident type. See Table A.12.
level	IncidentLevel(Enum)	Identifier of the incident level. See Table A.13.
message	String	*Optional.* Text describing the incident cause.
startTime	Number	Time in milliseconds, measured from the beginning of the mission (time 0), when the incident will be triggered.
endTime	Number	*Optional.* Time in milliseconds, measured from the beginning of the mission (time 0), when the incident will finish.

Table A.7: Targets definition collection.

Name	**Format**	**Description**
_id	ObjectId	MongoDB unique identifier
mission	ObjectId*-Reference to *(UAS_Mission, MissionInput)*	Foreign key identifying the Mission Scenario associated to this Targets Definition.
targets	Array	Lists the targets comprising this targets definition. See Table A.8

Table A.8: Targets definition collection.

Name	**Format**	**Description**
_id	ObjectId	MongoDB unique identifier
type	TargetType(Enum)	Identifier of the Target Type. Currently there is only one type of target in DWR ("Enemy" targets). Thus, this attribute is useless.
position	(Number,Number)	Initial position of the target.
speed	Number	Target speed (constant).

Table A.9: Event Types Enumeration.

ID	Key	Description	Params
0	USER INPUT	This Event Type includes all possible operator interactions in DWR.	See Table A.10.
1	DRONE REACH WAYPOINT	A UAV has reached a waypoint.	• UAVId • Waypoint Identification
2	INCIDENT STARTED	A new incident starts	• IncidentID • IncidentType
3	INCIDENT ENDED	An incident ends (either because the operator opposed it or because it had an scheduled end time).	• IncidentID • IncidentType
4	TARGET DETECTED	A UAV detects a target.	• detectorId • targetPosition
5	DRONE DESTROYED	A UAV is destroyed	UAVId
6	ACTION STARTED	An action or task begins.	• UAVId • ActionId • ActionType
7	ACTION ENDED	An action or tasks ends. This event is not always linked to the event ACTION STARTED, since there are times that a UAV starts an action but is not able to finish.	• UAVId • ActionId • ActionType
8	CONTROL MODE CHANGED	The control mode was automatically changed by the simulator.	ControlModeId

Table A.10: User inputs enumeration.

ID	Key	Description	Params
0	SELECT DRONE	The operator selects a UAV	UAVId
1	SET DRONE SPEED	The speed of a specific UAV is changed	UAVId
2	SET SIMULATION TIME RATIO	The simulation speed is changed by the operator. The value set can be seen in the *Simulation Snapshot* associated to this event.	
3	ADD WAYPOINT	Not used. Included in the input CHANGE DRONE PATH	
4	CHANGE DRONE PATH	The flying path of a UAV is changed by an operator, either because a new waypoint has been added, edited or removed.	UAVId
5	SET WAYPOINT SELECTION	Indicates if a waypoint has been selected/unselected in the *Waypoints panel* of the GUI	*Selected*(True if it is a selection, False if it is an unselection)
6	SET CONTROL MODE	The control mode is changed by an operator.	ControlModeId

Table A.11: UAV Status enumeration.

ID	Key	Description
0	LOITER	The UAV has no waypoints in his flying path.
1	CRUISE	The UAV is flying without performing any task.
2	REFUELING	The UAV is charging its fuel in a fuel station.
3	DEAD	The UAV has been destroyed during the simulation.
4	TASK	The UAV is performing a task.

Table A.12: Incident Type enumeration.

ID (String)	**Description**	**Params**
"DangerArea"	A new No Flight Zone appears during the execution of a mission. If a UAV overflies one of these areas, it will be destroyed.	• Area object, containing an array of (lat, lon) *vertices* describing a polygon. • End Time (required)
"PayloadIncident"	The UAV sensors are broken for a limited/unlimited interval of time	• Identifier of the affected UAV • End Time (*optional*)

Table A.13: Incident Level enumeration.

ID (String)	**Key**	**Description**
"ADVISORY"	ADVISORY	The incident will not cause any UAV to be destroyed, and the mission success will not be threatened.
"CAUTION"	CAUTION	The incident will not cause any UAV to be destroyed, but the mission success is threatened.
"WARNING"	WARNING	The incident may cause the destruction of one or many UAVs

Table A.14: Waypoint Type enumeration.

ID	**Key**	**Description**
0	ROUTE	Common waypoint used to fly between tasks.
1	ACTION	Located at the entry point of the mission tasks. Marks the start of a new task, whatever its type (Currently only *Surveillance* tasks are supported).
2	REFUELING	Located exclusively on fuel stations. Always have a "Refueling" action associated.
3	LAND	Located exclusively on airports. Always have a "Landing" action associated.

Appendix B

Publications

1. Rodriguez-Fernandez, Victor; D.Menendez, Hector; Camacho, David. Diseño de un Simulador de Bajo Coste para Vehículos Aéreos no Tripulados. Actas del X Congreso español sobre metaheursticas, algoritmos evolutivos y bioinspirados: MAEB 2015: Merida-Almendralejo, 4, 5 y 6 de febrero de 2015, pp. 447-454. Feb 2015.

2. Rodriguez-Fernandez, Victor; Ramirez-Atencia, Cristian; Camacho, David. A Multi-UAV Mission Planning Videogame-based Framework for Player Analysis. In Evolutionary Computation (CEC), 2015 IEEE Congress on. IEEE. Accepted. Sendai International Center, Sendai, Japan. May 25-28, 2015.

3. Rodriguez-Fernandez, Victor; D.Menendez, Hector; Camacho, David. Design and Development of a Lightweight Multi-UAV Simulator. Proceedings of 2015 IEEE International Conference on Cybernetics (CYBCONF 2015). Accepted. Gdynia Poland, June 24-26, 2015

4. Rodriguez-Fernandez, Victor; D.Menendez, Hector; Camacho, David. User Profile Analysis for UAV operators in a Simulation Environment. In proceedings of the 7th International Conference on Computational Collective Intelligence Technologies and Applications (ICCCI 2014). Accepted. Madrid, Spain. September 21-23, 2015.

Bibliography

[1] E Pereira, R Bencatel, J Correia, L Félix, G Gonçalves, J Morgado, and J Sousa. Unmanned air vehicles for coastal and environmental research. *Journal of Coastal Research*, pages 1557–1561, 2009.

[2] Farid Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *Journal of Field Robotics*, 29(2):315–378, 2012.

[3] P Fabiani, V Fuertes, A Piquereau, R Mampey, and F Teichteil-Königsbuch. Autonomous flight and navigation of vtol uavs: from autonomy demonstrations to out-of-sight flights. *Aerospace Science and Technology*, 11(2):183–193, 2007.

[4] J Everaerts et al. The use of unmanned aerial vehicles (uavs) for remote sensing and mapping. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 37:1187–1192, 2008.

[5] Jason S McCarley and Christopher D Wickens. Human factors concerns in uav flight. *University of Illinois at Urbana-Champaign Institute of Aviation, Aviation Human Factors Division*, 2004.

[6] Jeff Craighead, Robin Murphy, Jenny Burke, and Brian Goldiez. A survey of commercial & open source unmanned vehicle simulators. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 852–857. IEEE, 2007.

[7] R Andy McKinley, Lindsey K McIntire, and Margaret A Funke. Operator selection for unmanned aerial systems: comparing video game players and pilots. *Aviation, space, and environmental medicine*, 82(6):635–642, 2011.

[8] Amy L Alexander, Tad Brunyé, Jason Sidman, and Shawn A Weil. From gaming to training: A review of studies on fidelity, immersion, presence, and buy-in and

their effects on transfer in pc-based simulations and games. *DARWARS Training Impact Group*, 5:1–14, 2005.

[9] Flightgear. URL `http://www.flightgear.org`. [Online].

[10] Phil Summers, Dave Barnes, and Andy Shaw. Determination of planetary meteorology from aerobot flight sensors. In *Proceedings of 7th ESA Workshop on Advanced Space Technologies for Robotics and Automation*, 2002.

[11] B Cervin, C Mills, and BC Wünsche. A 3d interface for an unmanned aerial vehicle. *SE Stage*, 4, 2004.

[12] Simbad, . URL `http://simbad.sourceforge.net/`. [Online].

[13] Simrobot, . URL `http://www.informatik.uni-bremen.de/simrobot/index_e.htm`. [Online].

[14] Tim Laue, Kai Spiess, and Thomas Röfer. Simrobot–a general physical robot simulator and its application in robocup. In *RoboCup 2005: Robot Soccer World Cup IX*, pages 173–183. Springer, 2006.

[15] Unmanned Aerial Vehicles Roadmap. Roadmap 2002-2027. *US DoD, December*, 2002.

[16] Huaiyu Wu, Dong Sun, and Zhaoying Zhou. Micro air vehicle: Configuration, analysis, fabrication, and test. *Mechatronics, IEEE/ASME Transactions on*, 9(1): 108–117, 2004.

[17] HaiYang Chao, YongCan Cao, and YangQuan Chen. Autopilots for small unmanned aerial vehicles: a survey. *International Journal of Control, Automation and Systems*, 8(1):36–44, 2010.

[18] Pascal Brisset, Antoine Drouin, Michel Gorraz, Pierre-Selim Huard, and Jeremy Tyler. The paparazzi solution. In *MAV 2006, 2nd US-European Competition and Workshop on Micro Air Vehicles*, pages pp–xxxx, 2006.

[19] Luca F Bertuccelli, Han-Lim Choi, Peter Cho, and Jonathan P How. Real-time multi-uav task assignment in dynamic and uncertain environments. In *presentado al AIAA Guidance, Navigation, and Control Conference, Chicago, Illinois*, 2009.

[20] Tomonari Furukawa, Frédéric Bourgault, Hugh F Durrant-Whyte, and Gamini Dissanayake. Dynamic allocation and control of coordinated uavs to engage multiple targets in a time-optimal manner. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, volume 3, pages 2353–2358. IEEE, 2004.

[21] Brett Bethke, Mario Valenti, and Jonathan How. Cooperative vision based estimation and tracking using multiple uavs. In *Advances in Cooperative Control and Optimization*, pages 179–189. Springer, 2007.

[22] Benjamin Lavis, Yasuyoshi Yokokohji, and Tomonari Furukawa. Estimation and control for cooperative autonomous searching in crowded urban emergencies. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 2831–2836. IEEE, 2008.

[23] Richard Garcia and Laura Barnes. Multi-uav simulator utilizing x-plane. In *Selected papers from the 2nd International Symposium on UAVs, Reno, Nevada, USA June 8–10, 2009*, pages 393–406. Springer, 2010.

[24] Marc Pujol-Gonzalez, Jesus Cerquides, and Pedro Meseguer. Mas-planes: a multiagent simulation environment to investigate decentralised coordination for teams of uavs. In *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*, pages 1695–1696. International Foundation for Autonomous Agents and Multiagent Systems, 2014.

[25] Yves Boussemart and ML Cummings. Behavioral recognition and prediction of an operator supervising multiple heterogeneous unmanned vehicles. *Humans operating unmanned systems*, 2008.

[26] Yves Boussemart, Jonathan Las Fargeas, Mary L Cummings, and Nicholas Roy. Comparing learning techniques for hidden markov models of human supervisory control behavior. In *AIAA Infotech@ Aerospace'09 Conference, Seattle, Washington*, 2009.

[27] Yves Boussemart and Mary L Cummings. Predictive models of human supervisory control behavioral patterns using hidden semi-markov models. *Engineering Applications of Artificial Intelligence*, 24(7):1252–1262, 2011.

[28] Yves Boussemart, Mary L Cummings, Jonathan Las Fargeas, and Nicholas Roy. Supervised vs. unsupervised learning for operator state modeling in unmanned vehicle settings. *Journal of Aerospace Computing, Information, and Communication*, 8(3):71–85, 2011.

[29] Ji Hyun Yang, Marek Kapolka, and Timothy H Chung. Autonomy balancing in a manned-unmanned teaming (mut) swarm attack. In *Robot Intelligence Technology and Applications 2012*, pages 561–569. Springer, 2013.

[30] Musim - multiple uav simulation. URL http://www.siliconvalleysimulation.com/muDoxyDocs/main.html. [Online].

[31] Dreamhammer. URL http://www.dreamhammer.com/. [Online].

[32] Ballista. URL http://www.dreamhammer.com/ballista/#ourstory/uc-1. [Online].

[33] AJ Chaput. Conceptual design of uav systems. *Aircraft Design and Laboratory, Spring Semester*, 2004.

[34] Jerry L Franke, Vera Zaychik, Thomas M Spura, and Erin E Alves. Inverting the operator/vehicle ratio: Approaches to next generation uav command and control. *Proceedings of AUVSI Unmanned Systems North America*, 2005, 2005.

[35] Jacob W. Crandall and M. L. Cummings. Developing performance metrics for the supervisory control of multiple robots. In *Proceedings of the ACM/IEEE International Conference on Human-robot Interaction*, HRI '07, pages 33–40, New York, NY, USA, 2007. ACM. ISBN 978-1-59593-617-2. doi: 10.1145/1228716.1228722. URL http://doi.acm.org/10.1145/1228716.1228722.

[36] Jacob W Crandall and Mary L Cummings. Identifying predictive metrics for supervisory control of multiple robots. *Robotics, IEEE Transactions on*, 23(5):942–951, 2007.

[37] Christopher D Wickens. *Engineering psychology and human performance* . HarperCollins Publishers, 1992.

[38] ML Cummings and Stephanie Guerlain. Using a chat interface as an embedded secondary tasking tool. *Human performance, situation awareness and automation: Current research and trends. HPSAA II*, 1:240–248, 2004.

[39] Thomas C Hankins and Glenn F Wilson. A comparison of heart rate, eye activity, eeg and subjective measures of pilot mental workload during flight. *Aviation, space, and environmental medicine*, 69(4):360–367, 1998.

[40] Mica R Endsley. Automation and situation awareness. *Automation and human performance: Theory and applications*, pages 163–181, 1996.

[41] Mica R Endsley. Design and evaluation for situation awareness enhancement. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, volume 32, pages 97–101. SAGE Publications, 1988.

[42] Jill L Drury, Jean Scholtz, and Holly A Yanco. Awareness in human-robot interactions. In *Systems, Man and Cybernetics, 2003. IEEE International Conference on*, volume 1, pages 912–918. IEEE, 2003.

[43] Jill L Drury, Laurel Riek, and Nathan Rackliffe. A decomposition of uav-related situation awareness. In *Proceedings of the 1st ACM SIGCHI/SIGART conference on Human-robot interaction*, pages 88–94. ACM, 2006.

[44] Glenn Begis. Adaptive gaming behavior based on player profiling, August 22 2000. US Patent 6,106,395.

[45] Pang-Ning Tan, Michael Steinbach, Vipin Kumar, et al. *Introduction to data mining*, volume 1. Pearson Addison Wesley Boston, 2006.

[46] Pramila Rani, Changchun Liu, Nilanjan Sarkar, and Eric Vanman. An empirical study of machine learning techniques for affect recognition in human–robot interaction. *Pattern Analysis and Applications*, 9(1):58–69, 2006.

[47] Lawrence Rabiner and Biing-Hwang Juang. An introduction to hidden markov models. *ASSP Magazine, IEEE*, 3(1):4–16, 1986.

[48] Guy Brock, Vasyl Pihur, Susmita Datta, and Somnath Datta. clvalid, an r package for cluster validation. *Journal of Statistical Software (Brock et al., March 2008)*, 2011.

[49] Joseph L DeRisi, Vishwanath R Iyer, and Patrick O Brown. Exploring the metabolic and genetic control of gene expression on a genomic scale. *Science*, 278(5338):680–686, 1997.

[50] Vasker Bhattacherjee, Partha Mukhopadhyay, Saurabh Singh, Charles Johnson, John T Philipose, Courtney P Warner, Robert M Greene, and M Michele Pisano. Neural crest and mesoderm lineage-dependent gene expression in orofacial development. *Differentiation*, 75(5):463–477, 2007.

[51] Ameer Ahmed Abbasi and Mohamed Younis. A survey on clustering algorithms for wireless sensor networks. *Computer communications*, 30(14):2826–2841, 2007.

[52] T Warren Liao. Clustering of time series dataa survey. *Pattern recognition*, 38(11): 1857–1874, 2005.

[53] Michael W Berry and Malu Castellanos. Survey of text mining. *Computing Reviews*, 45(9):548, 2004.

[54] Jaideep Srivastava, Robert Cooley, Mukund Deshpande, and Pang-Ning Tan. Web usage mining: Discovery and applications of usage patterns from web data. *ACM SIGKDD Explorations Newsletter*, 1(2):12–23, 2000.

[55] Leonard Kaufman and Peter J Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 344. John Wiley & Sons, 2009.

[56] Susmita Datta and Somnath Datta. Comparisons and validation of statistical clustering techniques for microarray gene expression data. *Bioinformatics*, 19(4):459–466, 2003.

[57] John A Hartigan and Manchek A Wong. Algorithm as 136: A k-means clustering algorithm. *Applied statistics*, pages 100–108, 1979.

[58] Tapas Kanungo, David M Mount, Nathan S Netanyahu, Christine D Piatko, Ruth Silverman, and Angela Y Wu. An efficient k-means clustering algorithm: Analysis and implementation. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 24(7):881–892, 2002.

[59] José Manuel Pena, Jose Antonio Lozano, and Pedro Larranaga. An empirical comparison of four initialization methods for the k-means algorithm. *Pattern recognition letters*, 20(10):1027–1040, 1999.

[60] Leonard Kaufman and Peter Rousseeuw. *Clustering by means of medoids*. North-Holland, 1987.

[61] Chris Fraley and Adrian E Raftery. Model-based clustering, discriminant analysis, and density estimation. *Journal of the American statistical Association*, 97(458): 611–631, 2002.

[62] Arthur P Dempster, Nan M Laird, and Donald B Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the royal statistical society. Series B (methodological)*, pages 1–38, 1977.

[63] Maria Halkidi, Yannis Batistakis, and Michalis Vazirgiannis. On clustering validation techniques. *Journal of Intelligent Information Systems*, 17(2-3):107–145, 2001.

[64] Anastasios Tombros, Robert Villa, and Cornelis J Van Rijsbergen. The effectiveness of query-specific hierarchic clustering in information retrieval. *Information processing & management*, 38(4):559–582, 2002.

[65] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of classification*, 2(1):193–218, 1985.

[66] Paul Jaccard. *Nouvelles recherches sur la distribution florale.* 1908.

[67] Julia Handl, Joshua Knowles, and Douglas B Kell. Computational cluster validation in post-genomic data analysis. *Bioinformatics*, 21(15):3201–3212, 2005.

[68] Joseph C Dunn. Well-separated clusters and optimal fuzzy partitions. *Journal of cybernetics*, 4(1):95–104, 1974.

[69] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[70] J Kvarnstrom and Patrick Doherty. Automated planning for collaborative uav systems. In *Control Automation Robotics & Vision (ICARCV), 2010 11th International Conference on*, pages 1078–1085. IEEE, 2010.

[71] Kanna Rajan, Frederic Py, Conor McGann, John Ryan, Tom OReilly, Thom Maughan, and Brent Roman. Onboard adaptive control of auvs using automated planning and execution. In *International Symposium on Unmanned Untethered Submersible Technology (UUST)*, pages 1–13, 2009.

[72] Cristian Oliver Ramírez Atencia et al. Modelling unmanned vehicles mission planning problems as constraint satisfaction problems. 2014.

[73] Cristian Ramirez-Atencia, Gema Bello-Orgaz, David Camacho, et al. A simple csp-based model for unmanned air vehicle mission planning. In *Innovations in Intelligent Systems and Applications (INISTA) Proceedings, 2014 IEEE International Symposium on*, pages 146–153. IEEE, 2014.

[74] Cristian Ramírez-Atencia, Gema Bello-Orgaz, Maria D R-Moreno, and David Camacho. Branching to find feasible solutions in unmanned air vehicle mission planning. In *Intelligent Data Engineering and Automated Learning–IDEAL 2014*, pages 286–294. Springer, 2014.

[75] Sebastian Deterding, Miguel Sicart, Lennart Nacke, Kenton O'Hara, and Dan Dixon. Gamification. using game-design elements in non-gaming contexts. In *CHI'11 Extended Abstracts on Human Factors in Computing Systems*, pages 2425–2428. ACM, 2011.

[76] Svyatoslav Guznov, Gerald Matthews, Gregory Funke, and Allen Dukes. Use of the roboflag synthetic task environment to investigate workload and stress responses in uav operation. *Behavior research methods*, 43(3):771–780, 2011.

[77] Mary L Cummings, Sylvain Bruni, S Mercier, and PJ Mitchell. Automation architecture for single operator, multiple uav command and control. Technical report, DTIC Document, 2007.

[78] Michael Lewis and Jeffrey Jacobson. Game engines. *Communications of the ACM*, 45(1):27, 2002.

[79] Nodejs: open source, cross-platform runtime environment for server-side and networking applications. URL `http://nodejs.org/`. [Online].

[80] Phaser: Html5/javascript videogame development framework. URL `http://phaser.io/`. [Online].

[81] Angularjs: Client-side web development based on the model-view-controller model. URL `https://angularjs.org/`. [Online].

[82] Ian Fette and Alexey Melnikov. The websocket protocol. 2011.

[83] Victoria Pimentel and Bradford G Nickerson. Communicating and displaying real-time data with websocket. *Internet Computing, IEEE*, 16(4):45–53, 2012.

[84] Mongodb: Nosql, cross-platform, document oriented database. URL http://www.mongodb.org/. [Online].

[85] Alexander Lewis Aitkin et al. Playing at reality: Exploring the potential of the digital game as a medium for science communication. *Ph.D.dissertation, The Australian National University, October 2004*, 2004.

[86] C Shawn Green and Daphne Bavelier. Action video game modifies visual selective attention. *Nature*, 423(6939):534–537, 2003.