 Faculty of Computer Science
 Technical University of Madrid

A Human-Robot Cooperation System For Surface Inspection Aerial Missions

Master Thesis
Master in Artificial Intelligence

AUTHOR: Pedro Frau Amar
DIRECTORS: Martín Molina, Darío Maravall
Dedication

Every challenge needs self-effort. Every project needs compromise. Every goal is reached with perseverance and hard work. No matter what you do, take the tools and knowledge your environment gives you.

I want to dedicate my humble effort to my mother, who has always been looking for my future and my well-being; to my girlfriend Rocíó, who has shown her interest in my trajectory this year always pushing me forward and giving me support; but I specially want to dedicate this project to my deceased father, rest in peace. He taught me how to be a real engineer with concerns about the future, and gave me knowledge in several fields. He shared with me a lot of hobbies and helped me so many times with my projects.

Regardless of the final mark, this project is a success for me and has somehow a little bit of my father on it.

We will never forget you and will always love you.
Special thanks to:

First of all, I would like to thank my two supervisors, Prof. Darío Maravall and Prof. Martín Molina. They helped me to achieve the project and gave me advice on how to turn it into a successful work. They have always been very attentive and have always held out their hands to me.

Secondly, I would like to thank my laboratory teammates as they have also been very serviceable when I asked for help, and we shared a good working environment.

In the third place, I would also like to thank my Girlfriend Rocío, as she always showed her concern on my day to day and has given me support to accomplish all my goals.

Finally, I would like to thank my mother, my father, my brother and sister and the rest of my family and friends. They have always been interested about my trajectory and have given me support on all my decisions.
Publications:

The present document describes a project that has been carried out in collaboration with the group Computer Vision and Aerial Robotics (CVAR) of the Technical University of Madrid and has allowed to publish two papers listed in the references.

The first one [Molina et al., 2017] was published and presented in the International Micro-Air Vehicle Congress (IMAV) held in Toulouse (France) in September 2017. A link to the presentation video can be found in the annex section.

The second one [Molina et al., 2018] was published in the journal Sensors (Impact factor 2677) in March 2018.
Abstract

The development of functionalities for drones is constantly increasing. Nowadays, most enterprises of the drone industry are proposing new uses for those unmanned aerial vehicles, uses which mainly involve inspection or vigilance tasks. The goal of this work is to make a proposal on a human-robot cooperative system, hence considered semi-autonomous, to perform structural inspection tasks on walls, in order to find imperfections. This “semi-autonomy” is based on a mixed-interaction human-drone where the human asks the robot to perform specific tasks during an autonomous inspection mission in which the drone has to be able to find and classify several pre-trained imperfections and ask the operator for help if needed. For this purpose, an algorithm holding the concept of Frequency Histogram of Connected Elements will be introduced in the computer vision part of the project for imperfections recognition. The findings show that it is feasible to extract general patterns from such application to give certain robustness to the project, and the experiments can be used as a proof of concept for more general surface inspection missions. The whole project is framed on an increasing evolution of intelligent systems and robotics and freely available as a part of the open source framework Aerostack.

Resumen

El desarrollo de funcionalidades para drones crece constantemente. Hoy en día, la mayoría de las empresas de la industria de los drones proponen nuevas utilidades para estos vehículos no tripulados, utilidades que implican principalmente tareas de inspección y vigilancia. El propósito de este trabajo es proponer un sistema cooperativo humano-robot, considerado tanto semi-autónomo, capaz de realizar tareas de inspección estructural en muros con el objetivo de hallar desperfectos. Esta “semi-autonomía” está basada en una interacción mutua entre humano y drone en la que el humano pide al robot que realice ciertas tareas durante el transcurso de una misión autónoma de inspección en la que el drone debe ser capaz de hallar y clasificar diferentes desperfectos previamente entrenados y pedir ayuda al operario de tierra si es necesario. Con este objetivo, se desarrolla un algoritmo en base al concepto de Histograma de Frecuencia de Elementos Conexos que se incluye en la parte de visión por computador para el reconocimiento de desperfectos del proyecto. Los hallazgos muestran que es posible extraer ciertos patrones de la aplicación propuesta de manera a otorgar cierta robustez al proyecto, y los experimentos pueden ser usados como prueba de concepto para misiones más genéricas de inspección de superficies. El proyecto pretende contribuir a la evolución actual de sistemas inteligentes y robótica y está disponible de forma gratuita como parte del proyecto de código abierto de Aerostack.
Chapter 1. INTRODUCTION, MOTIVATION AND CONTEXT

1. INTRODUCTION

1.1. Context
1.2. Personal Motivation
1.3. Goals of the project
1.4. Structure of the report
1.5. Suitability for the master’s degree
1.6. Challenges

Chapter 2. HUMAN-ROBOT COOPERATIVE SYSTEM AND SURFACE AERIAL INSPECTION MISSIONS

1. INTRODUCTION

1.1. General description of aerial missions
1.2. Why a semi-autonomous system?
1.3. Description of surface aerial inspection problem
1.4. Scenarios

3. USER COMMAND LINE INTERPRETER

3.1. The idea
3.2. Speech acts theory
3.3. Robot assisted by operator
3.4. Operator assisted by robot
3.5. Robot assisted by other robots
3.6. Implementation

Chapter 3.
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>INTRODUCTION</td>
<td>36</td>
</tr>
<tr>
<td>1.1.</td>
<td>Needs</td>
<td>38</td>
</tr>
<tr>
<td>1.2.</td>
<td>Imperfections definition</td>
<td>38</td>
</tr>
<tr>
<td>2.</td>
<td>FREQUENCY HISTOGRAM OF CONNECTED ELEMENTS</td>
<td>39</td>
</tr>
<tr>
<td>2.1.</td>
<td>Introduction</td>
<td>39</td>
</tr>
<tr>
<td>2.2.</td>
<td>The algorithm</td>
<td>40</td>
</tr>
<tr>
<td>2.3.</td>
<td>Special cases</td>
<td>40</td>
</tr>
<tr>
<td>2.4.</td>
<td>Imperfections highlighting and output</td>
<td>41</td>
</tr>
<tr>
<td>2.5.</td>
<td>Denoising and recognition</td>
<td>43</td>
</tr>
<tr>
<td>2.6.</td>
<td>Collaboration Human-robot on flaw detection</td>
<td>45</td>
</tr>
<tr>
<td>Chapter 4.</td>
<td></td>
<td>48</td>
</tr>
<tr>
<td>1.</td>
<td>THE COMPLETE SYSTEM</td>
<td>50</td>
</tr>
<tr>
<td>1.1</td>
<td>Final structure</td>
<td>50</td>
</tr>
<tr>
<td>1.2</td>
<td>Main difficulties and solution</td>
<td>50</td>
</tr>
<tr>
<td>2.</td>
<td>EXPERIMENTATION AND RESULTS</td>
<td>51</td>
</tr>
<tr>
<td>2.1.</td>
<td>Contemplated scenarios</td>
<td>51</td>
</tr>
<tr>
<td>2.2.</td>
<td>Recognition tests and results</td>
<td>52</td>
</tr>
<tr>
<td>2.3.</td>
<td>Simulated flight tests and results</td>
<td>56</td>
</tr>
<tr>
<td>2.4.</td>
<td>Real Flight scenarios, tests and results</td>
<td>60</td>
</tr>
<tr>
<td>3.</td>
<td>CONCLUSIONS OF THE PROJECT AND FUTURE RESEARCH</td>
<td>62</td>
</tr>
<tr>
<td>References</td>
<td></td>
<td>64</td>
</tr>
<tr>
<td>ANNEX</td>
<td></td>
<td>66</td>
</tr>
</tbody>
</table>
Figures list

Figure 1: Sketch of the Scenario 22
Figure 2: Sketch of the scenario viewed from the top 23
Figure 3. Collaborative control for surface inspection [Molina et al., 2018]. 24
Figure 4: Block diagram with the main software components [Molina et al., 2017]. 27
Figure 5: Mission structure 28
Figure 6: Finite-state machine structure 29
Figure 7: Communication architecture. 32
Figure 8: Robot notification architecture. 34
Figure 9. Shape of the selected kernel. 40
Figure 10. top left-hand corner, top and top right-hand corner pixels. 40
Figure 11. Right-hand side, bottom right-hand corner and bottom pixels 41
Figure 12. Bottom left-hand corner and left hand side pixels 41
Figure 13. Image result after selecting the connected elements 41
Figure 14. FHCE for a hole 42
Figure 15. Original image vs Image with flaw delimited 43
Figure 16. Applying high contrast to images 44
Figure 17. Recognition results before denoising 44
Figure 18. Recognition results after denoising 45
Figure 19. Assistance request windows shown to the operator 45
Figure 20. Recognizing several flaws at once 46
Figure 21. Final structure of the system 50
Figure 22. Configured map for tests 56
Figure 23. Mission execution (HMI vision) 58
Figure 24. Mission execution (UCLI vision) 59
Figure 25. Simulated recognition using “flaw-shaped” elements 59

Tables list

Table 1: Representation of different inspection types 25
Table 2: Machine needs 26
Table 3: Definition of scenarios 27
Table 4: Control commands 33
Table 5: Flaws lexicon with visual correspondence 39
Table 6: Results of the recognition on evaluation dataset 54
Table 7: Results of the recognition on real flaw sequences 55
Table 8: Results of the recognition on fake flaws sequences to test false positives
Table 9: Command sequence for simulation
Chapter 1.

INTRODUCTION,
MOTIVATION AND CONTEXT
1. INTRODUCTION

1.1. Context

In the present, drone technology can be found anywhere: in the public commercial domain, in the professional domain and even in the military domain. A drone is an Unmanned Aerial Vehicle, which means that there is not a pilot aboard; hence it may have different levels of autonomy going from a remote control operator to a fully autonomous computer-driven system. There are a high number of different applications already developed for this technology and their versatility has no potential limits which make them have an increasing number of uses.

We can find several types of drones: Military airplane-shaped drones, quadcopters, hexacopters, octocopters, decacopters and even dodecacopters. Along this project, the word “drone” stands for the commercial AR.drone\(^1\) quadcopter. Nowadays, almost anyone can acquire a drone at any price and in the range of micro drone (approximately 40€) to drone (between 150€ and 4000€ approximately, it depends on the brand, specifications, equipment, etc.).

An important point is that the drone has some advantages in front of other aerial devices. Good drones are easily maneuverable and very stable and they normally come with a camera which is used as a First Person View that sends the image to another device. However, there are several fields in which drone technology can be improved. For instance, we may be particularly interested in creating behaviors to enhance the autonomy of those robots. Given that quadcopters may have a built-in camera, their use on applications involving image recognition is unlimited which is why we propound the following case of study in the form of a hypothesis:

- **Drones with a built-in camera can be used as a device for structural semi-supervised imperfections recognition in inspection missions.**

On the first place, a work with the AR.drone will be done regarding all the autonomy related implementation. The Aerostack framework developed by [Jose Luis Sanchez-Lopez et al. 2016](https://www.parrot.com/es/drones/parrot-ardrone-20-elite-edition#parrotardrone-20-elite-edition) will serve as a basis to build the inspection missions. Then, it will be enhanced with a two-directional communication system based on orders from the operator and notifications from the robot.

Regarding the image processing part, there are many algorithms which have already been implemented as libraries of the well-known open source platform *OpenCV*\(^2\).

---

2 - [https://opencv.org/](https://opencv.org/)
Nevertheless, a state of the art object recognition algorithm known as Frequency Histogram of Connected Elements (FHCE) proposed in its first state by [Maravall, Patricio, 2003] is the one we will be using as a start point for our project due to its efficiency for image segmentation as it has been proved in practical problems such as wooden pallets [Maravall, Patricio, 2007] and road segmentation for autonomous car driving [Maravall, Patricio, 2004].

1.2. Personal Motivation

In the final phase of my bachelor’s degree, I developed a Speech Controlled quadcopter which gave me very good results. Mainly, I built a quadcopter on my own which I programmed to follow simple voice orders and answer either with movements or with spoken synthesized notifications.

During the last three years I have been captivated by Robotics and Artificial Intelligence, so I decided to complement my academic formation with the Master’s degree in Artificial Intelligence in the Technical University of Madrid.

Searching for ideas for the Master’s final project, I realized the field of study of drones applications was something that I was really into, so my Bachelor’s final project became the starting point for this new horizon even if some approaches were changed. The idea became then “drone technology on image recognition tasks”.

1.3. Goals of the project

The main goals of this project are finding a good way to implement a human-robot cooperative system focused on mission achievement based on the Aerostack framework, and getting to characterize wall imperfections in order to find the best implementation of a recognizer for those flaws.

Most probably, what might give us more headaches will be the abstraction of the existent Aerostack framework to generalize mission tasks into more complex behaviors for wall inspection. Then the efforts have to be focused on the extraction of the most accurate characteristics of the wall flaws in order to get the best recognition results.

1.4. Structure of the report

This document follows a logical structure based on divider parts of the project each one with its sub-parts.

The first chapter provides an introduction to the context and motivation of the project. We present the technological background and a link with the Master’s degree.
Chapter 2 focuses on the drone missions, i.e. the Human-Robot Cooperative System used in Surface Aerial Inspection Missions. We provide a general description of the idea proposed in this project for quadcopter control and autonomy along certain laws given by the mission itself. Then we combine it with the Aerostack framework and the concept of User Command Line Interpreter.

Chapter 3 sets the bases of the surface imperfections recognition establishing the needs and the definition of such flaws. Then we present the Frequency Histogram of Connected Elements showing the concept itself, its adaption to our project and the tests and improvements made on this part.

Finally, chapter 4 provides an overview of the integrated system and a discussion on the main conclusions and future challenges of this project. It also includes a part talking about testing all the components of the whole system.

1.5. Suitability for the master’s degree

During the Master’s degree, I studied how multi-agent systems work; which are the main tools related to common sense reasoning; learning to find paths for obstacle avoidance on robots; self-localization on topological maps; how to do heuristics; how information is reclaimed, treated and classified and how to simulate decision making processes.

What about this project? It is mainly, the design of a semi-autonomous multi-agent system using drones to recognize objects on walls for structural inspection missions. As we can see from the title, the project relates almost every field of study seen in the master: it is a multi-agent system relating a drone which has to find its path to a certain goal and it involves image recognition in order to classify the flaws by means of certain given characteristics.

All in all, every field of the master is taken into account except for linguistics engineering. The project could be enriched by the use of natural user interfaces such as speech control or hand gestures, but this part is indeed semi-contemplated as we use visual markers for self-localization.

1.6. Challenges

The idea is to program both a system capable of performing simple inspection missions interacting with an operator in a semi-autonomous environment, and a flaws recognizer which has to be able to separate the imperfection from the background and classify it.

Our two main concerns will be then the abstraction of Aerostack to fit our inspection needs, and specially, the extraction of easily usable patterns for the recognition of the imperfections. Then, integrate all the parts to build a complete all-in-one drone wall-inspector.
The output of the work will be composed by the following parts:

- **The mission specification:** Here what we need is to take into account several inspection scenarios, and represent them as a single mission file. The idea is to allow the robot to move in the 3D space and search for imperfections.

- **The User Command Line Interpreter (UCLI):** The main idea of this part is to implement a command line interpreter in which we will be able to send commands to the drone and make him act as a consequence.

- **The Notification system:** This system aims to send all potentially important information to the user to facilitate the interaction by means of an information exchange.

- **The flaws recognition algorithm:** The algorithm will be developed based on the concept of Frequency Histogram of Connected Elements and will return as a result, the label for each flaw and its position in the image.

To integrate all those parts we will be creating processes for the Aerostack framework, processes which will be interacting among themselves.
Chapter 2.

HUMAN-ROBOT COOPERATIVE SYSTEM AND SURFACE AERIAL INSPECTION MISSIONS
1. INTRODUCTION

1.1. General description of aerial missions

In the context of this work, aerial missions are procedures realized by flying agents capable of performing some actions that a human wouldn’t be able to realize in a normal situation. In our concrete case, we are talking about the capability of performing an inspection task on physical structures that can be at different altitudes. For instance, along this project we aim to design a system to perform the inspection of imperfections on walls. For that reason it is mandatory for the drones to be able to localize themselves in front of the wall. To achieve that goal the wall needs to be mapped using visual Aruco markers delimiting the borders of the inspection area. Hence, the wall will be treated as a three-dimensional matrix which components will be represented by a position X on the horizontal axis (from side to side of the wall), a position Y on the depth axis (for zoom in and zoom out) and a position Z on the vertical axis (from bottom to top of the wall). The initial point of the drone will then be (0, -2, 0), on the left side bottom corner of the wall at a two meter distance from it. Even so, the Aerostack framework represents the scenario in the (x,y) plane and with respect to a point placed on the (0,0). This means that we have to imagine the wall placed (for example) at a 3 meters distance from the reference point and all drone positions will be taken also from the reference point. The wall will be then at (x, 3, z) and the drone at (1, 1, 0).

Below, a sketch of the scenario:

![Figure 1: Sketch of the Scenario](image-url)
Once initialized the scenario, it is mandatory to specify the robot’s behavior for the inspection. Hence, the drone will start its mission by doing a scanning of the wall depending on the specified movement strategy. For instance, in a normal zigzag strategy, the quadcopter would iterate over the rows of the Z axis of the wall and the columns of the X axis of the wall. In case the robot finds a normal imperfection, it will classify it by its label. Conversely, in case it finds a large imperfection, the drone will ask for help to the operator which will tell the drone to perform a zoom out (for example).

In a scenario needing more than one robot, the drones different from the main one will be located at the initial point to be able to self-localize themselves and approach the main drone if needed.

1.2. Why a semi-autonomous system?

Surface inspection missions in aerial robotics may require special human-robot interaction with intermediate degrees of robot autonomy between manual teleoperation and complete autonomy. In this type of scenario, the aerial robot may behave as an assistant for the human operator who delegates in the vehicle inspection tasks. The robot might have certain inspection abilities (e.g., path planning, defect recognition, etc.). These abilities reduce
significantly the workload of the operator and increase the safety of the system, compared to simple manual teleoperation. However, in this type of mission, it is difficult to have robots that operate fully autonomously because they don’t have a complete understanding of the environment. Robots may have recognition abilities for certain defects but, sometimes, certain defects are difficult to classify automatically. In this case, the robot can ask for assistance to the operator.

Figure 1 summarizes this type of human-robot interaction. On the one hand, the operator can play the role of supervisor. This form of automation is related to the notion of supervisory control [Sheridan, 1992] in which a human operator is intermittently acting on the robot to delegate tasks.

![Figure 1. Human-robot interaction diagram](image)

But, on the other hand, the operator can also play the role of assistant. The human works as a resource for the robot, providing additional information. The robot may ask the human questions as it works, to obtain assistance with perception and cognition. This allows the human to compensate for limitations of autonomy. This is related to the idea of collaborative control in which human and robot work together [Fong et al., 2003]. The human and the robot dialogue to exchange information, to ask questions, and to resolve differences.

**1.3. Description of surface aerial inspection problem**

Surface aerial inspection problems are not difficult to characterize. Mainly, the idea is to perform a good scanning of the surface so that all the existing flaws on it can be found. To do that, we just have to think of possible drone paths that would allow the robot to scan the whole wall.

![Figure 3. Collaborative control for surface inspection](image)
Several inspection types have been defined in the first project specification of characteristics as shown in the following table:

<table>
<thead>
<tr>
<th>Inspection type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zig Zag Scanning</td>
<td></td>
</tr>
<tr>
<td>Up and Down Scanning</td>
<td></td>
</tr>
<tr>
<td>Spiral Scanning</td>
<td></td>
</tr>
<tr>
<td>Scanning by zones</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Representation of different inspection types

For simplicity reasons we will only be using first two inspection types.

In the first case (i.e. Zig Zag Scanning), the wall will hold three visual markers to delimit its edges. First two Aruco visual markers will be placed on the sides of the wall to indicate the moment in which the drone has to start going up and change its direction. Then a third visual marker will be placed at the top of the wall to indicate the end of the inspection task.

In the second case (i.e. Up and Down Scanning), the settings will be the same except for the position of the visual markers. In this case, the first two Aruco markers will be placed at the top and the bottom of the wall, and the end of the inspection task will be indicated by the third aruco placed at the right side of the wall.

1.4. Scenarios

1.4.1. Machine needs
According to the mission specifications discussed above, we will list several needs that might be interesting for our inspection purposes and have to be taken into account for the system design:
The drone does not recognize the flaw properly

It can be related to several causes. The flaw might not have been trained, there is visibility issues (the drone needs more light), the flaw is too big/small,...

The drone has low battery

We need to handle this possibility by allowing other drones to approach the first one and take over it.

The drone can not fly stably

In a real scenario, we could be flying in front of an outside wall, there could be wind or other causes of instability

The drone does not have the needed instrumentation

Not all the drones can be equipped with the same instrumentation. In this case, as we are implementing a multi-agent system, the drone has to be able to call better instrumented agents

The drone loses its localization references

There might be cases in which for any reason, the drone got lost. The operator then has to relocate it in the scenario to continue the mission.

### Table 2: Machine needs

#### 1.4.2. Definition of Scenarios

The following table shows a set of possible scenarios involving the human-machine interaction model described in this project. Using those scenarios, we will implement the system behaviors

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low visibility and mission delegation</td>
<td>Robot R1 finds a dark area which does not allow the proper recognition. The operator orders R1 to turn on the light and continue with the mission. Then, the drone has a low battery charge. The operator transfers the mission to robot R2.</td>
</tr>
<tr>
<td>Distributed specialized tasks</td>
<td>Robot R1 finds a hole. The operator orders to delimitate the hole zone. The drone asks for painter drone help. Painter drone draws a circle around the hole. The first drone finishes the mission.</td>
</tr>
<tr>
<td>Reconstruction of large fissure</td>
<td>Robot R finds a large fissure. The operator orders to change the inspection strategy to up/down. The drone starts moving taking several pictures of the fissure. Once finished, it makes a reconstruction of the fissure and</td>
</tr>
</tbody>
</table>
classifies it.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zoom out for large fissure</td>
<td>Robot R finds a large fissure. The operator orders to zoom out. The robot gets a better (complete) view and classifies it.</td>
</tr>
<tr>
<td>Lost position</td>
<td>Robot R loses its position with respect to the wall. The operator relocates the drone in the inspection area using movement orders and orders to continue the mission.</td>
</tr>
<tr>
<td>New object to recognize</td>
<td>Robot R finds an unknown imperfection that cannot be classified following the trained imperfections. The drone asks the operator what to do, and the operator decides to create a new class stain.</td>
</tr>
</tbody>
</table>

Table 3: Definition of scenarios

2. AEROSTACK

2.1. System Implementation

In order to implement the cooperation approach for surface inspection missions described above, we used and extended the software framework Aerostack (www.aerostack.org) [Sanchez-Lopez et al., 2016]. Aerostack is a general software framework for aerial robotics that provides different software components and a combination scheme for building the software architecture for autonomous operation of an aerial robotic system. For example, Aerostack provides software components with perception algorithms, SLAM algorithms, controllers, mission plan interpretation methods, multi-robot communication and a general graphical user interface for human-robot interaction.

![Block diagram with the main software components](Molina et al., 2017)

Figure 4: Block diagram with the main software components [Molina et al., 2017].
Figure 7 shows a block diagram with the main software components of our implementation. In the figure, blue blocks correspond to processes provided by Aerostack and orange blocks are new processes that we programmed and added to Aerostack for inspection problems. For instance, we programmed the process called surface defect recognizer implementing the FHCE algorithm, which will be well explained later, using the images captured by the front camera of the drone. In this implementation of FHCE we also used simple routines provided by the OpenCV library (e.g., for representation of images, pixel manipulation, etc.).

Aerostack provides a library of behaviors such as general motion behaviors (take off, land, go to point, etc.). We extended this library with specific behaviors useful for inspection missions. For example, we implemented several inspection strategies such as the zig zag strategy or the up and down strategy and added other behaviors such as the behavior zoom, to move closer to or further from the surface.

2.2. Mission definition and Task specification in Python

To allow the interpretations of human orders and the semi-autonomy of the robots, it is mandatory to define a structured mission for the drone to perform. The idea is to encompass all possible actions of the drone in a single mission file specifying possible events during the execution of the main drone behavior.

All in all, we will be dealing with a set of cases, one for each command linked to different functions specifying movement behaviors for the robot. Hence, the main functions will be those related to “move” commands, but others such as a memorize position or go to that memorized position will be also contemplated. Figure 5 shows a flowchart of the main structure of the mission file:
Tricky functions:

CONTINUEMISSION can be considered as a tricky function. For instance, if we imagine a scenario in which the operator interrupts the normal performance of an action, the drone has to know what to “keep doing” after the interruption. For that reason, we need to separate the drone’s movements into two kind of movement behaviors, namely:

- **Main movement behaviors**: The ones that the drone will be performing autonomously, i.e. **Right**, **Left**, **Up** and **Down** in order to carry out **zig zag** and **up and down** movement strategies.

- **Secondary movement behaviors**: The ones specified by the human for a certain reason, for instance, **zooming in** or **turning the lights on**.

Then, all in all, the main behaviors of the drone can be interrupted by secondary behaviors. To implement this, we will be building a finite-state machine composed by 4 states (“going up”, “going down”, “going right” and “going left”) within two different inspection strategies and which will be catenated to configure the main behaviors. In addition to that, we will use interruption points configuring the secondary behaviors that will keep the main state to know how to continue the mission after the interruption. Figure 6 summarizes with a diagram the finite state machine design:

![Figure 6: Finite-state machine structure](image-url)
3. USER COMMAND LINE INTERPRETER

3.1. The idea

Based on the previous collaborative scheme, we designed a human-robot interaction model described in [Molina et al., 2017] and [Molina et al., 2018] considering messages in categories according to the theory of speech acts [Searle, 1969, 1975; Austin, 1975]. We consider different illocutionary acts to distinguish the intention of the messages, and other subcategories defined by different schemes: DAMSL (Dialogue Act Markup In Several Layers) [Core, Allen, 1995], KQML [Labrou, Finin, 1997], Move Coding Scheme [Carletta et al., 1997], etc.

In particular we use the following categories: (1) assertive messages (for example, the robot informs the operator the completion of a task), and (2) directive messages: messages which cause the receiver to perform a particular action. Within directive messages, we distinguish between two categories: action directives (requests for action) and information requests.

3.2. Speech acts theory

In linguistics, speech acts are statements with performative function in language and communication. According to [Bach K., 2014], “almost any speech act is really the performance of several acts at once, distinguished by different aspects of the speaker's intention: there is the act of saying something, what one does in saying it, such as requesting or promising, and how one is trying to affect one's audience”.

With those statements in mind, we can classify speech acts according to three levels:

- **Locutionary Acts**: Simple act of speaking
- **Illocutionary Acts**: Speak uttering an intention
- **Perlocutionary Acts**: Speak influencing others

Given that statement, we can design our system by means of a division of speech acts into Perlocutionary and Illocutionary acts:

**Perlocutionary Acts**:
In our case they will express orders:
- Start Mission
- Change Mission Strategy
- Move …
- Go Home
- Call Painter
- Turn light On
- …

**Illocutionary Acts**:
- Sending current Position
3.3. Robot assisted by operator

Robots may have recognition abilities for certain defects but, sometimes, certain defects are difficult to classify automatically. In addition, unexpected changes of the environment (e.g., shadows, wind, etc.) may require attention from the operator to decide the appropriate response.

This happens because robots have partial knowledge and are not completely self-sufficient. Therefore, in the presence of uncertainty, a robot may ask the operator for assistance. In this case, the robot works like the field technician (i.e., it is skilled, but may need help) and the operator is like the expert (i.e., she or he can provide assistance when needed) as it is considered in human collaborative control [Fong et al., 2002].

3.4. Operator assisted by robot

The operator may ask the aerial robot to perform an inspection mission, specifying the area to cover and the exploration strategy. In this case, the relation between operator and robot follows a hierarchical authority (as supervisor-subordinate schema) in which the operator delegates a set of tasks to the robot. The robot executes the inspection mission and when it recognizes a defect, it stores the information and notifies it to the operator. Then the drone continues with the inspection.

During the development of the mission, the operator observes the robot behavior. This observation is based on information messages sent from the robot to the operator that confirm that the mission is being developed as expected. The operator can interrupt the mission under certain circumstances (for example, to avoid wrong behaviors).

3.5. Robot assisted by other robots

We consider also that the robot may delegate certain specialized tasks to other robots. For example, the robot can transfer part of the mission to other robots because it does not have enough battery charge, or can delegate a certain specialized task that requires specialized actuators (e.g., use a special device to mark the detected defect on the wall).

3.6. Implementation

3.6.1. The user command line interpreter

Provided that our goal is to create an interactive human-robot system, it is mandatory to create an interface for communication. The first step is then to implement a command interpreter.
Along the project we will be using a command line interpreter based on basic order commands that we can send to the robot.

The idea is to implement a new listener process, capable of taking input from the user and transform it into drone actions. The main goal is to create a high-level program that can interact with a complete Python mission file containing all possible behaviors of the robot which may be activated by means of constant message listening or external events (i.e. visual markers). The python program will be subscribed to the order topic from which it will be receiving orders for the drone. Then the mission itself will interpret the orders and make the drone act as a consequence.

Figure 7 shows a diagram of this architecture.

![Diagram of Communication Architecture](image)

In order to make the drone perform the actions specified by the user, we will create a main function in the Python file which will basically be waiting for orders. Hence, the drone will start its mission stopped and will wait for a takeoff. Then we will create new conditions that will make calls to other task functions.

Regarding the main process, it will be a simple interpreter that will publish a string command over the order_topic. The list of commands will be:

<table>
<thead>
<tr>
<th>Behavior</th>
<th>Command Sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>TAKE_OFF</td>
<td>TO</td>
</tr>
<tr>
<td>LAND</td>
<td>L</td>
</tr>
<tr>
<td>GO_HOME</td>
<td>GH</td>
</tr>
</tbody>
</table>
3.6.2. Robot message viewer

Once we have created the message interpreter, it is mandatory to receive the robot’s feedback. To do so, we will be interested in implementing a new behavior named “NOTIFY_OPERATOR” which will receive as argument a message, a string specifying the information to be sent to the operator.

In our Python mission, we will make a call to “NOTIFY_OPERATOR” in order to check its argument. Then, we will publish that argument on a topic named notification_topic to which our UCLI will be subscribed. Notifications will be then seen on the user command line interpreter to make the human-machine interaction easier. The architecture is shown on figure 8.
Figure 8: Robot notification architecture.
Chapter 3.

SURFACE IMPERFECTIONS RECOGNITION
1. INTRODUCTION

1.1. Needs

One of the required skills of an autonomous robot for surface inspection tasks is the ability to detect abnormal marks in a surface and classify the images in the corresponding category. For this purpose, it is possible to use computer vision algorithms.

Here, the most important point is to make a profitable extraction of patterns of such flaws in order to be able to characterize them properly. The idea is to implement an algorithm capable of separating the flaws from the background and label them depending on their nature.

1.2. Imperfections definition

Here we will define the flaws lexicon with its visual correspondence, i.e. the flaws and its “name”. We will define several flaws even if, for simplicity reasons, only fissures and holes will be used along the project. The following table shows the lexicon with visual correspondence for each defined flaw.

<table>
<thead>
<tr>
<th>Lexicon</th>
<th>Visual Anchorage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fissure</td>
<td>![Fissure Image]</td>
</tr>
<tr>
<td>Hole</td>
<td>![Hole Image]</td>
</tr>
<tr>
<td>Peeling</td>
<td>![Peeling Image]</td>
</tr>
</tbody>
</table>
2. FREQUENCY HISTOGRAM OF CONNECTED ELEMENTS

2.1. Introduction

In this work, we have used the method based on frequency histogram of connected elements (FHCE) [Maravall, Patricio, 2003]. This method is useful to treat the image pixel by pixel and characterize the different types of flaws of the surface.

FHCE uses the concept of neighborhood, i.e. for a given pixel \( p(i,j) \) of an image, its neighborhood is formed by a set of pixels which distances to \( p \) are not greater than two integer values \( r, s \) and is defined as \( \varphi_{(i,j)}^{r,s} \). A connected element \( (T) \) is the neighborhood selected such as the intensity \( I \) of a pixel \( q(k,l) \) is a subset of a given grayscale range \([T - \varepsilon, T + \varepsilon]\):

\[
C_{(i,j)}(T) = \varphi_{(i,j)}^{r,s} : I(k,l) \subset [T - \varepsilon, T + \varepsilon], \forall (k,l) \in \varphi_{(i,j)}^{r,s}
\]

Given the previous definitions, \( H(T) \) is defined as the sum of all the connected elements for each pixel of an image on different gray levels \( T \) where \( T \) is greater than 0 and inferior than the maximum intensity minus one:

\[
H(T) = C_{(i,j)}(T) \quad 0 \leq T \leq I_{\text{max}} - 1
\]

Following that idea, what we need is to extract from each video frame the number of pixels relative to the flaws. So we will be building the FHCE with several images of different flaws in order to see which are the patterns followed by the imperfections.
2.2. The algorithm

Having in mind the previous statements, we have to start by defining the neighborhood shape. As we are talking about holes and fissures we thought about a linear and a square kernel. Then we discovered that the square kernel could be generalized to fit both the hole and fissure needs which is why we decided not to differentiate between imperfections. Our square kernel will be formed by a 3x3 matrix of neighboring pixels where the center pixel is the target.

![Figure 9. Shape of the selected kernel.](image)

Once we have the neighborhood selected, we have to compute the standard deviation and the mean between the pixels of the neighborhood. Given a certain threshold on the standard deviation chosen by trial and error, we will decide whether to assign to each pixel its real value or its neighborhood mean value. Finally, we need to compute the FHCE which will tell us the intensity values that are more present in the image i.e. the pixels where there is an interesting element (maybe a flaw).

Given the histogram, we can define the threshold for the intensity values that fits our needs in order to separate the flaws from the background.

2.3. Special cases

As we will be iterating over each pixel of the image matrix, there are several special cases to take into account in order to avoid out of bounds exceptions. For instance, all pixels belonging to the image surroundings set cannot use a square kernel. A set of conditions will be disposed to handle those exceptions shown below:

![Figure 10. top left-hand corner, top and top right-hand corner pixels.](image)
2.4. Imperfections highlighting and output

Once we have selected the best kernel to use, we need to compute the FHCE. For that purpose, we will be treating each neighborhood to obtain its standard deviation $\sigma$ and its mean value. If $\sigma$ is lesser than a certain threshold, we will assign to the pixel the mean value of all pixels of the neighborhood, else, we will maintain its grayscale value. The threshold will be chosen later on the document while checking the best results. The result obtained is shown below:

![Image result after selecting the connected elements](image-url)
Even if it’s difficult to appreciate it on the image above, several zones of the picture have taken the same intensity value, i.e. we reduced the scale variation in order to differentiate better between similar pixels with respect to the rest. Hence we can see that all the regions of the image have a better delimiting by their grayscale value and then the picture is better segmented.

Once this has been done, we have to compute the FHCE. To do that, we will create a vector of grayscale values and we will count for each value its appearance in the image. That way, we will be able to plot the histogram according to the frequency of appearance of each gray value. This will give us information related to the characterization of the flaws with respect to the image.

![Figure 14. FHCE for a hole](image)

Taking a look to the histogram we can try to separate the elements in the image. The first thing we can see are two predominant areas, one on the range (170, 200) and the other on the range (100, 170). those two areas correspond to the two ranges of predominant intensity values on the image, i.e. a brighter part of the wall on the right (170, 200) and a darker part of the wall (probably caused by a shadow) on the left (100, 170). We can notice that the sum of the pixels of those two parts of the image will cover almost the whole picture. Once we have identified those elements of the image, we can imagine that the hole and the stain will be covered by the range (0, 100), so as a first approach we will say that the hole (darker element in the image) will correspond to the range (0, 80) and the stain will correspond to the range (80, 100). Later on experimentation and results part we will see how to tune those values to fit better our needs.
After this process, we can say that we have extracted the important flaw (the hole) from the image and we can try to characterize it in order to be able to delimit it by a rectangle later. In this case, we will use white pixels to characterize the element in the image depending on the grayscale values range we chose to delimit the flaw. This step is summarized on the following pseudocode of the algorithm and the results are shown below according to [Molina et al., 2018].

![Image: Original image vs Image with flaw delimited](image)

**Algorithm 1. Defect recognition**

**Input:** digital image $I$ (a still picture)

1. $[k_1, k_2] \leftarrow$ interval of gray values for anomalies (based on the frequency histogram analysis)
2. for each pixel $(i, j)$ in input image $I$
3. $g_{i,j} \leftarrow$ gray value of pixel $(i, j)$
4. if $(g_{i,j} \in [k_1, k_2])$ then substitute the value of pixel $(i, j)$ by the distinctive color $s$
5. $R_1 \leftarrow$ rectangles obtained as contour of images of color $s$
6. $R_2 \leftarrow$ rectangles $r_i \in R_1$ such that $area(r_i) \in [k_3, k_4]$
7. $D \leftarrow$ classified rectangles $r_i \in R_2$ according to shape conditions
8. return($D$)

**2.5. Denoising and recognition**

As we can see in the images above, even if we have been able to perform a quite good separation of the flaws from the background, we still have elements in the image which can be confusing for the recognizer and give us bad recognition results. To solve this problem, we have to apply a denoising strategy to avoid false positives while maintaining a correct recognition ratio.

To do that, we will be performing several tasks based on the heuristic analysis of the imperfections which aims to answer to the following questions:

- What is a fissure?
- What is a hole?
By answering to those two questions we could set the basis to design our denoiser. All in all, a fissure is a linearly-shaped imperfection (either horizontal or vertical), and a hole is a circular or square-shaped flaw. In addition, if we compare our system to a real scenario in which a human performs an inspection of the wall, a small imperfection could be considered as a “too far to recognize” imperfection or a “not important” imperfection, and a big flaw would imply the wall to be rebuilt. Taking that into account, we have to set some limitations on the dimensions of the flaw. Then, our main objective is to avoid any object on the image which doesn’t apply to those two definitions, which means avoiding any too big or too small object on the image (depending on our specifications of “big object” and “small object”) or any object which doesn’t fit the pre-defined shapes.

To do that, we will be using two different functions from the OpenCV library, namely `findContours()` and `contourArea()`. If we assign white values to the potential flaws and black values to the rest of the image, we will be creating high contrast images that will allow us to find the contours of those potential flaws.

![Figure 16. Applying high contrast to images](image)

![Figure 17. Recognition results before denoising](image)

Hence, we will be able to see that there are several zones which are considered as a flaw while they are not. Here is where we can use a function which will separate the imperfections from other elements depending on the specified area (in pixels). Taking this into account, we can specify an area greater than “x” pixels and lesser than “y” pixels (The process on how to
find $x$ and $y$ will be explained later on the document) to separate the true flaws from the fake ones.

Some other actions (related to our concrete scenario) have been taken to avoid false positives. For instance, to delete the blue cap from the image, we filtered all the blueish pixels.

### 2.6. Collaboration Human-robot on flaw detection

To make the process more reliable, we decided to implement an assistance request behavior in which the idea of a collaborative system is clearly shown. We decided to add a new step to the recognition of flaws using a process called the belief manager that handles the recognition done by our system and shows the results to the operator so that this one can confirm or reject the recognition.

To do that, once the system has detected a flaw, the drone is stopped and asked to wait for a response. Meanwhile, the recognition results are sent to the belief manager that stores the flaw with its position (in world coordinates) and its category, and shows the belief as an uncertain recognition to the operator. In case the recognition is confirmed, the belief manager will keep track on its memory of the flaw in that position so that it’s not treated again. In case the recognition is rejected, the recognized element will also be stored, but in this case, it will be removed from memory as an actual flaw to prevent the system to recognize again a false imperfection.
Following with the same logic, we need to handle the cases in which the image has two different flaws. In that concrete case, the system will recognize both imperfections but will start tracking one of them highlighting it in blue. The imperfection highlighted will be tracked till the operator confirms or rejects it, and once this has been done, the system will start tracking the other imperfection in the image and start the process again.

Figure 20. Recognizing several flaws at once
Chapter 4.

SYSTEM INTEGRATION
1. THE COMPLETE SYSTEM

1.1 Final structure

In this section we aim to describe the complete final structure of the whole system. All in all the integrated system is compounded by the collaborative components (i.e. the user interface, the notification system, all the components from the Aerostack framework), and the flaws recognition system.

Below we can see a diagram of the whole final structure:

![Diagram of the final structure](image)

Figure 21. Final structure of the system

As we can see, everything is a built in part of the whole solution which would serve as a flaw recognizer for aerial inspection tasks.

1.2 Main difficulties and solution

Along the whole project we have been facing several complications during the development of all the components. Apart from the programming difficulty itself, we noticed that the integration of all the components brought us difficulties as Aerostack framework has been evolving a lot due to the migration to a new architecture.

In addition, we faced some technical problems related to the performance of our machines on the recognition system part of the work. We were performing flaws recognition on each frame of the video stream and the graphic card of the computer we were using wasn’t enough powerful. Those problems were solved by investing on new computers.
Another difficulty we faced was the testing on real environments. Even if we had the space to perform some flying tests on a real scenario with a wall full of flaws, the space wasn’t always available. Plus, the university performed some maintenance tasks on the building hiding most of the imperfections we were using and we got blocked on the performance of tests.

2. EXPERIMENTATION AND RESULTS

2.1. Contemplated scenarios

The experiments we will be performing will be both simulated and real flight tests. The drone has to be able to fit, while performing testing tasks, the needs listed above:

- **The drone obtains good recognition results with respect to the defined patterns for each imperfection**: Given a heuristical definition of the imperfections following certain patterns, the robotic agent has to be able to detect a potential flaw on the façade and classify it correctly.

- **In case there is a flaw which is not recognized, there is a way to take control of the drone and facilitate its recognition task (zoom in/out,...)**: The drone has to be capable of detecting an imperfection on the wall even if it isn’t catalogued. If this is the case, the quadcopter will communicate with the human agent showing an image of the flaw so that he can take a decision in relation to the cataloging of the element, classifying it as a known flaw or generating a new class which would need to be trained later.

- **Communication between human agent and robot is fluid**: The human agent has to be able to take control of the machine at any moment. To assure that, we have to be sure there is no lag in the communication channel, that all the notifications sent by the robot are received by the operator, and that all the commands work properly.

- **The drone has a certain degree of autonomy being able to self-localize in the scenario**: In case the operator is not controlling the drone, the robot has to be able to self-localize and continue the mission by itself taking into account previously given settings.

- **All processes are robust to exceptions so that crashes on runtime are avoided**: All the processes have to be tested several times and taking into account several scenarios to avoid unexpected crashes. In a real case scenario, the drone should not be at risk or put people at risk.

- **At the end of the experimentation phase, the obtained success ratio in recognition (where the success is measured in terms of the number of flaws found with respect to the existent flaws) is higher than 80%**: This implies the
localization of the most severe imperfections which could cause a collapse of the structure. In addition the ratio of false positives with respect to correct classifications is acceptable: After the wall inspection, it will be mandatory to review the classification made by the drone to ensure that it is correct and/or rectify potential errors or unexpected difficulties.

2.2. Recognition tests and results

To run the tests related to flaws recognition and its accuracy, we need to prepare first of all a dataset of imperfections containing several images of fissures and holes. This dataset will be divided in two parts, namely, the design part and the evaluation part.

To do that we recorded several videos of imperfections on a wall and we extracted the frames from those videos generating a dataset composed by almost 1600 different images divided into a set of positive coincidences for both fissures and holes flaw types, a set of negative coincidences for both flaw types, four sequences of images for fissures and three sequences of images for holes.

The positive sets were used to test the correct recognition of the flaws on each image. Negative sets were used to test the percentage of false positives generated by the recognizer on images not containing the evaluated flaw. Finally the sequences were used to test whether the system would be able to identify the flaw on several moments of the whole video stream so that we can ensure that no flaw would be avoided if the drone passes in front of it.

The design part was composed by the 60% of the images of the dataset for each flaw allowing us to decide which range of grayscale values we could use to get better results, which flaw dimensions would be better for our dataset, which threshold should be used after running the FHCE as the minimum standard deviation and which value we should use to differentiate between fissures and holes. Then we used the evaluation dataset (composed by the rest of images of the whole dataset to evaluate the results after having decided the better parameters for our system.

In addition, we labeled all the images of the dataset manually to be able to run the recognition tests. To do that, we wrote two programs in python language. The first one was aimed to allow a user to draw a rectangle around the flaw in each image, and save the image with the characteristics of this rectangle (length and center) on its name. The output of the recognizer was also the images with the characteristics of the rectangle surrounding the imperfection on the name of the file. The second python program was aimed to compare the characteristics of the surrounding rectangles on both labelled dataset and recognition output telling us the results as a percentage of correctly classified flaws.

Another thing we noticed while preparing the dataset, was the dispersion among flaws of the same type, and we found that the flaws in each one of the images are very different in terms of imperfection position and size. The following images allow us to visualize this dispersion.
For simplicity on the explanation, we will name the parameters to test as follows:

- **k1**: This parameter is used to decide the threshold of the standard deviation between pixels to be considered for the same neighborhood. $k1 \in [0, 240]$ evaluating steps of 30. This would make us perform 8 tests to find the best value to use.

- **k2**: This parameter states the upper value of the range of grayscale values considered as imperfections. The lower value is 0 and $k2 \in [50, 130]$ evaluating steps of 10. This would make us perform 9 tests to find the best value to use.

- **k3**: This parameter states the lower value of the area (in pixels) to consider a flaw as a correct recognition in our dataset. $k3 \in [40, 400]$ evaluating steps of 40. This would make us perform 10 tests to find the best value to use.

- **k4**: This parameter states the upper value of the area (in pixels) to consider a flaw as a correct recognition in our dataset. $k4 \in [1000, 12000]$ evaluating steps of 1000. This would make us perform 12 tests to find the best value to use.
- k5: This parameter is used to compare the sides of the surrounding rectangle of the flaw and allow us to make the difference between fissures and holes. $k_5 \in [10, 100]$ evaluating steps of 10. This would make us perform 10 tests to find the best value to use.

All in all, we have to perform 49 different tests on both positive and negative design datasets for fissures and holes and then find the best configuration of parameters to run a single test on evaluation dataset for both positive and negative images of fissures and holes and all 5 sequences of images. This would make a total amount of 105 tests to be run among 1594 images. Each one of the tests on a design dataset of 186 images approximately takes an average of 35 seconds to be performed so all in all we have to invest 4 hours and 5 minutes approximately to perform all the tests and get the results.

To deal with this problem, we wrote several scripts to be ran autonomously for each parameter in order to avoid spending that amount of time in front of the computer.

After running all those tests, we found that the best values of the parameters applied to our dataset are:

- $k_1 = 30$
- $k_2 = 90$
- $k_3 = 120$
- $k_4 = 11000$
- $k_5 = 40$

Applying those values to the evaluation data set we obtained the following results on the images:

<table>
<thead>
<tr>
<th>Truth</th>
<th>Prediction</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Holes</td>
<td>Fissures</td>
</tr>
<tr>
<td>Holes</td>
<td>89%</td>
<td>32%</td>
</tr>
<tr>
<td>Fissures</td>
<td>20%</td>
<td>70%</td>
</tr>
</tbody>
</table>

Table 6: Results of the recognition on evaluation dataset

We see that 89% of the holes were recognized as such and 70% of the fissures were recognized as such. In addition, the results show, that blocking the recognition of fissures gives us a 20% of false positives with type “hole” and blocking the recognition of holes gives us a 32% of false positives with type “fissure”.
Then we obtained the following results on flaw sequences for the positive dataset:

<table>
<thead>
<tr>
<th>Sequence type</th>
<th>Sequence number</th>
<th>Flaws recognized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fissures</td>
<td>1</td>
<td>43/50</td>
</tr>
<tr>
<td>Fissures</td>
<td>2</td>
<td>50/50</td>
</tr>
<tr>
<td>Fissures</td>
<td>3</td>
<td>36/50</td>
</tr>
<tr>
<td>Fissures</td>
<td>4</td>
<td>75/86</td>
</tr>
<tr>
<td>Holes</td>
<td>1</td>
<td>47/50</td>
</tr>
<tr>
<td>Holes</td>
<td>2</td>
<td>50/50</td>
</tr>
<tr>
<td>Holes</td>
<td>3</td>
<td>18/50</td>
</tr>
</tbody>
</table>

Table 7: Results of the recognition on real flaw sequences

As we can see, in almost all the sequences we get more than 70% of success on recognition except for the last holes sequence in which we get only 36% of success. In this last case, the results are due to the fact that we are dealing with a sequence of images in which the flaw is very small with respect to the rest of the image. Here we would solve the problem by means of the collaborative architecture where the human would tell the drone to zoom into the flaw to see it better, but, in any case, the drone would detect it without the help of the human at some point during its trajectory which means that the imperfection would never be skipped.

Below we can see the table showing the results of false positive recognition on flaws sequences.

<table>
<thead>
<tr>
<th>Sequence type</th>
<th>Sequence Number</th>
<th>Flaws recognized</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fissures</td>
<td>1</td>
<td>41/50</td>
</tr>
<tr>
<td>Fissures</td>
<td>2</td>
<td>15/50</td>
</tr>
<tr>
<td>Fissures</td>
<td>3</td>
<td>50/50</td>
</tr>
<tr>
<td>Fissures</td>
<td>4</td>
<td>2/50</td>
</tr>
<tr>
<td>Holes</td>
<td>1</td>
<td>39/50</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------</td>
<td>-----</td>
<td>---</td>
</tr>
<tr>
<td>Holes</td>
<td>2</td>
<td>0/50</td>
</tr>
<tr>
<td>Holes</td>
<td>3</td>
<td>3/50</td>
</tr>
</tbody>
</table>

Table 8: Results of the recognition on fake flaws sequences to test false positives

In this case, we can see that apparently we get very bad results on the first and the third sequence of fissures and the first sequence of holes. The truth is that those concrete sequences contain imperfections of the other type (i.e. fissures sequences contain holes and holes sequence contains fissures) which makes the recognizer find imperfections on the images. Then, we can say that, what appears to be a fail on false positive flaws recognition, is indeed a success on real imperfections recognition. In addition, the rest of the sequences show that most of the times (more than 70%), if there is no supplementary imperfection, the fail percentage is due to the recognizer not finding anything.

### 2.3. Simulated flight tests and results

First of all, several simulated tests have been run to see the performance of the command line interpreter. Mainly, we have tried to use all the commands in a simulated scenario using the human-machine interface from Aerostack. We created a simple map in which the drone is situated in front of a 10 meters long wall at a two meters distance from it as shown below:

Figure 22. Configured map for tests
Then, using the UCLI process, we will be able to control the drone. Our mission will consist on doing a takeoff and starting a zigzag. Then at the very beginning, we will stop the mission and resume it. At a certain point, we will use a fake command to show what happens when the user makes a mistake and then demand the drone to zoom in to show the available commands and zoom out. The following step will be to ask the drone to move left, stop it and demand for moving up. We will ask to the robot to show us several notifications, such as its position and finally, we will demand to terminate the mission, the quadcopter will then go to the initial point and land.

The complete command sequence is shown in the table below:

<table>
<thead>
<tr>
<th>Command</th>
<th>Usage</th>
</tr>
</thead>
<tbody>
<tr>
<td>TO</td>
<td>The drone will do a takeoff</td>
</tr>
<tr>
<td>SM</td>
<td>The drone will start the mission (Default ZigZag)</td>
</tr>
<tr>
<td>S</td>
<td>The drone will stop</td>
</tr>
<tr>
<td>CM</td>
<td>The drone will resume the mission</td>
</tr>
<tr>
<td>ZU</td>
<td>Fake command to get an error message</td>
</tr>
<tr>
<td>ZI</td>
<td>The drone will zoom in</td>
</tr>
<tr>
<td>HELP</td>
<td>We will get a list of commands</td>
</tr>
<tr>
<td>ZO</td>
<td>The drone will zoom out</td>
</tr>
<tr>
<td>ML</td>
<td>The drone will start moving left</td>
</tr>
<tr>
<td>S</td>
<td>The drone will stop</td>
</tr>
<tr>
<td>MU</td>
<td>The drone will start moving up</td>
</tr>
<tr>
<td>SP</td>
<td>We will receive a notification of the position of the drone</td>
</tr>
<tr>
<td>FM</td>
<td>The drone will go to the point targeted as “home” and will land</td>
</tr>
</tbody>
</table>

Table 9: Command sequence for simulation

After the tests we have found several problems related to the configuration of the mission and some inconsistencies between functions. For instance in a python mission, we can execute a behavior or just activate it. The first case will execute the behavior and will not liberate the
execution of the program till the behavior has been completed. The second one will activate the behavior and leave it in “background” so that the rest of the program will continue its normal execution. Hence, almost all the behaviors have to be called using the function activate and allowing the program to continue listening for commands. There is only a few behaviors that have to be executed and not just activated, behaviors such as takeoff and land, which have to terminate before doing anything else.

In addition, several bugs related to other processes have been found and solved during the execution of the tests. Bugs related to the trajectory planning, or the behaviors themselves. Also, a few bugs were found related to the UCLI, for instance, notifications were not being received or the commands were not properly sent to the drone.

Once resolved all those problems, the mission was executed as expected. Below we show a picture of the HMI while the drone performs the mission, and a picture of the UCLI:

![Figure 23. Mission execution (HMI vision)](image-url)
Regarding the simulation of the flaw recognition process, we will use the drone’s camera on a flat surface with lines and “hole shaped” elements and see whether the recognition results are good or bad.

At the very beginning we obtained the recognition results according to the first definition of the flaw recognizer. We observed that there were too much noisy elements considered as flaws which weren’t at all. Heuristically, we redefined the recognition process erasing from the image all the elements (or almost all) which were not imperfections. The results are shown in the pictures below.
2.4. Real Flight scenarios, tests and results

Along this section, we recover the table described earlier in this document (table 3) where several real scenarios are described and explained in detail to characterize the kind of situations in which the proposed system might be helpful. The table shows a set of experimental scenarios that were considered to cover different situations. These experiments serve as a proof of concept for the kind of human-robot collaborative scenario presented in this document. Even so, not all the considered experiments have been carried out and are left as future research objectives.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low visibility and mission delegation</td>
<td>Robot R1 finds a dark area which does not allow the proper recognition. The operator orders R1 to turn on the light and continue with the mission. Then, the drone has a low battery charge. The operator transfers the mission to robot R2.</td>
</tr>
<tr>
<td>Distributed specialized tasks</td>
<td>Robot R1 finds a hole. The operator orders to delimitate the hole zone. The drone asks painter robot R2 for help. R2 draws a circle around the hole. The first drone finishes the mission.</td>
</tr>
<tr>
<td>Reconstruction of large fissure</td>
<td>Robot R1 finds a large fissure. The operator orders to change the inspection strategy to up/down. The drone starts moving taking several pictures of the fissure. Once finished, it makes a reconstruction of the fissure and classifies it.</td>
</tr>
<tr>
<td>Zoom out for large fissure</td>
<td>Robot R finds a large fissure. The operator orders to zoom out. The robot gets a better (complete) view and classifies it.</td>
</tr>
<tr>
<td>Lost position</td>
<td>Robot R loses loses its position with respect to the wall. The operator relocates the drone in the inspection area using movement orders and orders to continue the mission.</td>
</tr>
<tr>
<td>New object to recognize</td>
<td>Robot R finds an unknown imperfection that cannot be classified following the trained imperfections. The drone asks the operator what to do, and the operator decides to create a new class stain.</td>
</tr>
</tbody>
</table>

Table 7: Real scenarios for flight tests

Figure 26 shows the trajectory followed by the robot in one of the flight experiments. This example corresponds to an aerial platform AR Drone 2.0 performing a indoor inspection to find holes and fissures on a wall as shown on figure 27. The whole trajectory is covered in 1 minute and 19 seconds. The robot develops autonomously a horizontal exploration starting from the takeoff point (2, 2, 0) and landing at the same point after the exploration. There are two points where the robot zooms in and out to have closer views of the wall: point (2, 3, 1) and point (6, 3, 1).
As the graph shows, the drone is able to self-localize in the map and follow a predefined path. This path was delimited by Aruco visual markers on both sides of the wall and on its top left corner to characterize the end of the mission. This experiment proves that the proposed approach of a collaborative system used in aerial inspection missions is a good starting point.

Image 26: Example exploration trajectory followed by the drone [Molina et al., 2017]

Image 27: Parrot AR Drone 2.0 performing real flight tests
3. CONCLUSIONS OF THE PROJECT AND FUTURE RESEARCH

Along this document a human-robot cooperative approach has been presented as a solution for aerial inspection missions. Here, a mixed initiative involving several ways of connecting the human operator with the drone have been proposed to build a paradigm of the communication between robots and humans composed by a flexible dialogue. The project presented in this work aims to give a generic solution to an inspection problematic by means of two interaction modes based on supervision and assistance. But the proposed solution is intended to be used in other situations; hence, the idea of a collaborative inspection system can be extrapolated to new different scenarios.

Throughout the document the solution proposed has proven to answer positively to the hypothesis raised in the introduction. It can be said then that drones with a built-in camera can be used as a device for structural semi-supervised imperfections recognition in inspection missions.

In the present paper, a software system has been shown to validate this model using the Aerostack framework allowing us to carry out several real flight experiments. In addition, comparing this system to a tele-operated solution, the idea presented in this document gives more autonomy to the robotic agent reducing the cognitive load of the human operator.

As opened lines for improvement, more behaviors could be created to extend the approach. Plus, the agent-agent collaboration is still pending to be implemented and it would give for sure more richness to the project. As another possibility, the imperfections dataset could be enhanced with more pictures both of existing and new imperfections. Finally, computing load and performance could be largely enhanced on the recognition side of the project.
References


ANNEX

Aerostack repository:
https://github.com/Vision4UAV/Aerostack

Published Papers:
  - Video: https://www.youtube.com/watch?v=HExy26XQdeE
- Sensors: http://www.mdpi.com/1424-8220/18/3/893/htm

Videos:
- Flight simulation with User Command Line Interpreter:
  drive.google.com/open?id=0BwYIiG7Li4JZBeWVsvMWNVOTl1cUE/view?usp=sharing
- Collaborative system with two imperfections:
  drive.google.com/file/d/0BwYIiG7Li4JZBeWVsvMWNVOTl1cUE/view?usp=sharing
- Rejection of non-imperfection and acceptance of hole:
  drive.google.com/file/d/0BwYIiG7Li4JZBeWVsvMWNVOTl1cUE/view?usp=sharing
- Rejection of non-imperfection and acceptance of fissure:
  drive.google.com/file/d/0BwYIiG7Li4JZBeWVsvMWNVOTl1cUE/view?usp=sharing
- Drone flight:
  drive.google.com/file/d/0BwYIiG7Li4JZBeWVsvMWNVOTl1cUE/view?usp=sharing